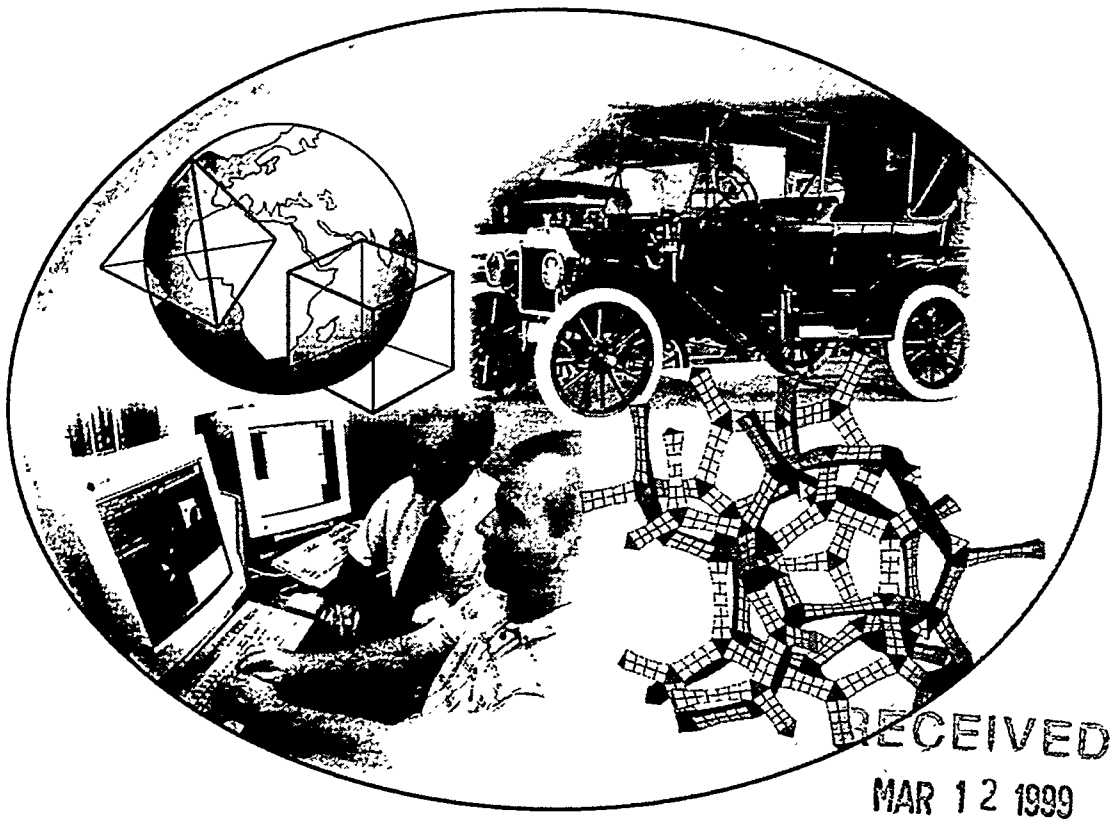
Proceedings

SAND 98-2250

7th INTERNATIONAL MESHING ROUNDTABLE '98



Sponsored by:
Sandia National Laboratories

Hosted by:
Ford Motor Company

SANDIA REPORT
SAND 98-2250
Unlimited Release
Printed October 1998

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interest of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by Sandia Corporation, the United States Government, or the other authors.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Additional copies can be ordered from
Sandia National Laboratories
PO Box 5800, MS0441
Albuquerque, NM 87185-0441

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Proceedings

7th INTERNATIONAL MESHING ROUNDTABLE '98

October 26-28, 1998
Dearborn, Michigan

Sponsored by:
Sandia National Laboratories

Hosted by:
Ford Motor Company

Introduction

The goal of the 7th International Meshing Roundtable is to bring together researchers and developers from industry, academia, and government labs in a stimulating, open environment for the exchange of technical information related to meshing and general pre-processing techniques. In the past, the Roundtable has enjoyed significant participation from each of these groups from a wide variety of countries.

The Roundtable will consist of technical presentations from contributed papers, an invited speaker, and an invited panel of experts discussing topics related to the development and use of automatic mesh generation tools. In addition, we will again feature a "Meshing Maestro" competition and poster session at Ford's Automotive Hall of Fame to encourage discussion and participation from a wide variety of mesh generation tool users.

Papers are being sought that present original results on meshing and pre-processing techniques. In addition to our core topics in meshing related algorithms, we are also interested in obtaining technical papers that relate analysis and application solution to the mesh generation process. Potential topics include but are not limited to:

- Volume and surface mesh generation techniques
- Anisotropic mesh generation
- Hybrid meshing approaches
- Mesh optimization and mesh quality control
- Adaptive mesh refinement techniques
- Geometry decomposition and clean-up techniques
- Industrial robustness and complex geometries
- Technical issues related to working with million element meshes
- Assemblies of automatically generated meshes
- Theoretical or novel ideas with practical potential
- Technical presentations from industrial researchers
- Application sessions
- Issues with CAD/CAM
- Mesh visualization

The 1998 Roundtable is steered by a committee taken from private industry, universities, and government laboratories.

Roundtable information can be found on the World Wide Web at URL
http://www.mcs.anl.gov/~freitag/7IMR/

Front Cover: Mitzie Bower, Sandia National Laboratories

COMMITTEE

Conference Host(s):

Phil Tuchinsky, Host
Ford Motor Company
Ford Research Laboratory
Mail Drop 2122 SRL
P.0. Box 2053
Dearborn, Michigan
Phone: (313)
E-Mail: ptuchins@ford.com

Harley Wattrick, Host
Ford Motor Company
Powertrain Operations
Dearborn, Michigan
Phone: (313) 323-2893
E-Mail: hwattric@ford.com

Conference Chairman:

Lori Freitag, Chair
Argonne National Laboratory
9700 S. Cass Ave., MCS221/C236
Argonne IL 60439
Phone: (630) 252-7246
Fax: (630) 252-5986
http://www.mcs.anl.gov/~freitag/
E-Mail: freitag@mcs.anl.gov

Conference Coordinator:

Tammy Eldred
Sandia National Laboratories
P.O. Box 5800, MS 0841
Albuquerque, NM 87185-0475
Phone: (505) 844-0180
Fax: (505) 844-8251
E-Mail: tjeldre@sandia.gov

Paper Submission:

David White, Paper Submissions
Sandia National Laboratories
P.O. Box 5800, MS 0441
Albuquerque, NM 87185-0475
Phone: (505) 284-4562
Fax: (505) 844-9297
E-Mail: drwhite@sandia.gov
http://sass577.endo.sandia.gov/~drwhite/

Conference Committee:

Ted Blacker
Fluent Inc.
500 Davis Street, Suite 600
Evanston, IL 60201
Phone: (847) 491-0200 x315
Fax: (847) 869-6495
E-Mail: ted@fluent.com

Ed D'Azevedo
Oak Ridge National Laboratory
P.O.Box 2008, Bldg. 6012
Oak Ridge, TN 37831-6367
Phone: (423) 576-7925
Fax: (423) 574-0680
http://www.epm.ornl.gov/~efdazedo/
E-Mail: ed@ornl.gov

Kenji Shimada
Carnegie Mellon University
Department of Mechanical Engineering
Pittsburgh, PA 15213-3890
Phone: (412) 268-3614
Fax: (412) 268- 3348
http://hpme12.me.cmu.edu/faculty1/shimada/
E-Mail: shimada@cmu.edu

Shanghua Teng
Department of Computer Science
University of Illinois at Urbana-Champaign
2109 DCL, mc 258
1304 W Springfield
Urbana, IL 61801
Phone: (217) 244-5972
E-Mail: steng@cs.uiuc.edu

VISIT OUR WEB SITE:
http://www.mcs.anl.gov/~freitag/7IMR/

~ Agenda ~

Sunday, October 25th

6:00 - 9:00 p.m.	~ Reception & Registration ~ Refreshments will be served. (Firestone Room - Third floor of Dearborn Inn)

Monday, October 26th

7:30 a.m.	REGISTRATION (Salon IV, First Floor Ballroom, Dearborn Inn)
8:30 - 9:45 a.m.	WELCOME & KEY NOTE ADDRESS
9:45 - 10:00 a.m.	BREAK - Complimentary Continental Breakfast
10:00 - 11:30 a.m.	SESSION 1 (Single Session) Plenary I
11:30 - 12:45 a.m.	LUNCHEON - Catered at Hotel
12:45 - 2:00 p.m.	SESSION 2 (Parallel Session) Session 2A - Quadrilateral Meshing Session 2B - Tetrahedral Meshing
2:00 - 2:15 p.m.	BREAK - Complimentary Refreshments
2:15 - 3:30 p.m.	SESSION 3 (Parallel Session) Session 3A - Hybrid Volume Meshing Session 3B - Adaptive Techniques
3:30 - 3:45 p.m.	BREAK - Setup for Panel Discussion
3:45 - 5:00 p.m.	BIRDS-OF-A-FEATHER SESSION
5:30 - 8:00 p.m.	POSTER SESSION AND RECEPTION Automotive Hall of Fame, Dearborn
8:00 p.m.	DINNER - open. Suggestions provided to local restaurants

Tuesday, October 27th

8:15 a.m.	OPENING REMARKS
8:30 - 10:00 a.m.	MINI-TUTORIALS
10:00 - 10:15 a.m.	BREAK - Complimentary Continental Breakfast
10:15 - 11:45 a.m.	SESSION 4 (Single Session) Plenary II
11:45 - 1:30 p.m.	LUNCH - No host Suggestions provided to local restaurants
1:30 - 2:45 p.m.	SESSION 5 (Parallel Sessions) Session 5A - Triangular Meshing Session 5B - Hexahedral Meshing
2:45 - 3:00 p.m.	BREAK - Complimentary Refreshments
3:00 - 4:15 p.m.	SESSION 6 (Parallel Sessions) Session 6A - Surface Meshing Session 6B - Mesh Quality
4:15 - 4:30 p.m.	BREAK - Set up for Panel Discussion
4:30 - 5:45 p.m.	PANEL DISCUSSION
7:00 - 11:00 pm	BANQUET & AWARDS Henry Ford Museum

Wednesday, October 28th

8:15 a.m.	OPENING REMARKS
8:30 - 9:45 a.m.	SESSION 7 (Parallel Session) Session 7A - Novel Meshing Approaches Session 7B - Practical Industrial Experiences
9:45 - 10:00 a.m.	BREAK - Complimentary Continental Breakfast
10:00 - 12:00 a.m.	OPEN PROBLEMS FORUM - ENDING REMARKS

Table of Contents

Session 2B - Tetrahedral Meshing, continued

Session 3A - Hybrid Volume Meshing

Session 3B - Adaptive Techniques

Mini-Tutorials

Session 4 - Plenary II

Session 5A - Triangular Meshing

Session 5B - Hexahedral Meshing

Session 6A - Surface Meshing

Session 6B - Mesh Quality

Session 7A - Novel Meshing Approaches

Session 7B - Practical Industrial Experiences

Panel Discussion & Birds-of-a-Feather

Keynote Address Abstract

Enhancing CAE Effectiveness: The Next Steps

Dr. Marc Halpern
Director of Research; Engineering, Manufacturing, and Design
D. H. Brown Associates, Inc.
222 Grace Church Street
Portchester, NY 10573

Abstract

Across all industries, companies have embraced the concept of early problem solving or "front-end loading"; to improve product development lead times. As a critical component of the strategy, companies adopt FEA-based simulation tools to construct models of components and products. If these models effectively capture the essence of the problem to be solved, the simulations provide insight to design decisions earlier in the product development process than higher fidelity physical prototypes which take longer to build and cost more.

Historically, model preparation has been a major barrier to fulfilling the promise of finite element analysis. Reliable results were often delivered too late to have a significant impact on design decisions. Fortunately, technological innovation has enabled substantial progress towards improving model preparation time since the early 1990s. Recent innovations promise additional reductions in model preparation times.

Our research over the past year suggests that the next wave of productivity gains will derive from data management that supports finite element analysis requirements. For example, major manufacturers report that they still face substantial challenges making the right data and models available to their analysis specialists in the form they need it. A new wave of technology should facilitate effective reuse of accumulated data, models, and product development knowledge.

This presentation explores these trends. It will highlight progress made, the current status of commercial technology, and the critical issues shaping further advancement of CAE technology.

Session 1
Plenary I

BMSweep: Locating Interior Nodes During Sweeping

M. L. STATEN[1], S. A. CANANN[1,2], S. J. OWEN[1,2]

[1]*ANSYS Inc. 275 Technology Drive, Canonsburg, Pennsylvania, 15317, U.S.A.*

[2]*Department of Civil and Envirnomental Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A.*

1.0 Abstract

BMSweep is a new algorithm to determine the location of interior nodes while volume sweeping. BMSweep uses background mesh interpolation to locate interior nodes while sweeping two and one half dimensional volumes. Three dimensional volumes can be swept using BMSweep after being decomposed into two and one half dimensional volumes. The interpolation method provides for quality element creation while allowing the volume boundary to vary.

KEYWORDS: mesh generation, sweep, background mesh, hexahedra, mapped meshing, 2.5D.

2.0 Introduction

For a variety of reasons, hexahedral elements are often prefered over tetrahedral elements for use in finite element analysis[1,2]. However, unlike tetrahedral meshes, hexahedral meshes are much more constrained and, therefore, much more difficult to generate[3]. Sweeping is a technology that has received significant research in the past few years and is a method of meshing two and one half dimensional volumes with an all hex mesh [4-9]. A two and one half dimensional volume is a volume that has a topologically constant cross section along a single axis known as the "sweep axis." (Figure 1a).

The procedure used in most sweeping methods is to first identify a source area and a target area at opposite ends of the sweep axis. The source area is meshed with a quadrilateral mesh. This quadrilateral mesh is then swept through the volume towards the target area using a specified number of layers. Each quadrilateral on the source area forms a single hexahedral element in each of the layers during sweeping. If the source area is meshed with triangles, wedges (i.e. triangular prisms) are formed instead of hexahedra.

The method of sweeping can be modified to mesh more complicated geometries by allowing for multiple source and/or multiple targets [4-6]. Blacker[4] presents the Cooper Tool which is a variation of sweeping. The Cooper Tool takes a volume and divides it up into "barrels". Each barrel has a single source area and a single target area. Each barrel is meshed separately, ensuring that the boundary mesh of each barrel conforms to any adjacent barrels. Once each barrel is meshed, the final mesh is simply the combination of the elements in all of the barrels. Liu and Gadh [6] also use sweeping to mesh some of their BLOBs (Basic LOgical Bulk). Similar to the Cooper Tool, Liu and Gadh [6] decompose the volume into simpler shapes which are sent to either a sweeper or a volume mapped mesher.

One problem with the current sweeping technology is how to determine the location of the new nodes created on the interior of each barrel. Shih and Sakurai [7] present a method to determine these interior node locations; however,

their method requires that both the source and target areas be planar and parallel to each other. Mingwu and Benzley [5] present another method of determining these interior node locations, but their algorithm requires a time-consuming smoothing operation after the hexahedral elements are formed[10]. In addition, even with smoothing, the quality of the resulting elements can be poor for complex models. Blacker [4] indicates that the Cooper Tool determines the interior node locations using a least square weighted residual method, however, the algorithm is not documented.

This paper presents an alternate method for determining the location of these interior nodes via background mesh interpolation. Hereafter, this algorithm will be called Sweeping via Background Mesh Interpolation (BMSweep). BMSweep provides the following benefits:

- Is General enough to sweep any two and one half dimensional volume.
- Is Orientation insensitive.
- Is computationally inexpensive.
- Does not require flat source and target areas.
- Does not require that the source and target areas be parallel.
- Does not require smoothing after the initial placement of the interior nodes.
- Does not require the source and target areas to have the same shape. (While both the source and target areas must have the same number of loops, the areas can have different surface area and shape.)
- Does not require a constant cross section shape along the sweep direction, however, constant cross section topology is required.

3.0 The Sweeping Algorithm

Before BMSweep begins to sweep through a volume, the following four assumptions are made:

1. The geometry to be swept is a two and one half dimensional volume either unaltered or created by decomposing the volume. Volume decomposition may be done manually or by employing an automated method [4,6,8,13].
2. The source area is meshed with a quadrilateral , quad/tri mixed or triangle mesh [11,12].
3. The target area is meshed with a projection of the source area mesh. Both source and target meshes must have the same mesh topology.
4. All side areas are meshed with a regular, gridded, quadrilateral, mapped mesh [13]. In addition, the mapped meshes on the side areas must conform from one side area to the next. The nodes on the side area mapped meshes are organized into "ribs". A rib is a continuous line of edges extending from a node on the source area, through a single node on each layer and terminating at a node on the target area [4]. Figure 1a shows a two and one half dimensional volume and Figure 1b shows what the boundary ribs may look like. With the side areas mapped meshed, the boundary ribs are defined before invoking BMSweep. Interior ribs begin at the source area interior nodes and terminate at the corresponding target area nodes. The objective of BMSweep is to determine the location of the nodes on the interior ribs.

With these assumptions, BMSweep proceeds using the following steps:
1. Generate a background mesh. (see section 3.1)
2. Calculate interpolation information using the background mesh. (see section 3.2)
3. Using the background mesh and interpolation information, calculate the location of each interior node and place it on the corresponding rib. (see section 3.3)
4. Connect the nodes created in step 3 to form the new elements. (see section 3.4)

These four steps are described in the following sections.

3.1 Generation of the Background Mesh

BMSweep requires a background mesh to provide a framework for computing the locations of the nodes on the interior ribs for each layer of the mesh. The background mesh is generated by tesselating the source area's boundary nodes. Since tesselation is more conveniently and robustly done in 2D, the parametric coordinates of the source area boundary nodes may be used. For this implementation, a 2D Bowyer/Watson[16] Delaunay triangulation algorithm is used to tesselate the nodes. For example, Figure 1c shows the mesh on the source area of the volume in Figure 1a. Figure 1d shows the source area boundary nodes projected to parametric space and tesselated. For the volume in Figure 1a, Figure 1d serves as the background mesh.

The background mesh must be created using only the boundary nodes of the source area. No interior points should be added to the background mesh. Although there are interior nodes on the source area that could be inserted into the background mesh, these nodes should not be inserted. This is because the background mesh will elevated to each layer during the sweep (see section 3.3) and none of the intermediate layers have nodes yet that correspond to the source area interior nodes.

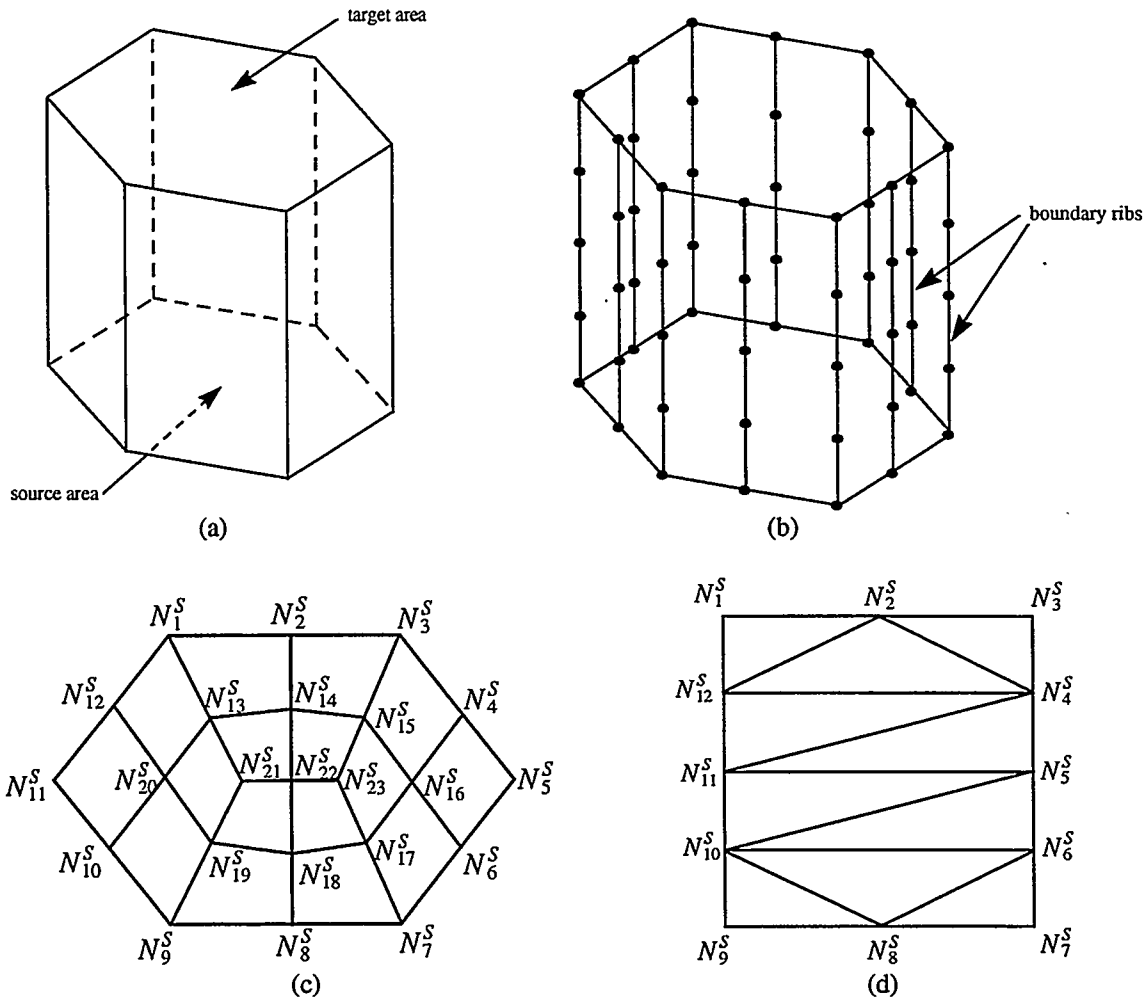


Figure 1 Example of background mesh for simple 2 ½ dimensional volume

3.2 Calculating Interpolation Information

Interpolation data is needed at each interior source area node, including:

1. Which triangle in the background mesh contains it (see section 3.2.1),
2. Its Barycentric coordinates in both the source and target areas (see section 3.2.2), and
3. Its offset distance with respect to the source and target areas. (The offset distance is the distance from an interior node to the background mesh triangle containing the node in 3d space. The offset distance is zero if both the source and target areas are flat.) (see section 3.2.3)

3.2.1 Triangle determination

With the background mesh in 2D parametric space, the 2D parametric coordinates of each interior source area node N_i^S is used to determine which background mesh triangle, T_i contains it. Once T_i is determined, a pointer to T_i is stored with N_i^S for future reference. It is likely that a single background mesh triangle will contain more than one interior source area node. In this case, each N_i^S contained in a particular triangle stores a pointer to the same background mesh triangle.

For example, Figure 2 shows the background mesh in parametric space from Figure 1 with the source area mesh projected onto it (shown with dotted lines). The triangles that contain each of the interior source area nodes (N_{13}^S - N_{23}^S) are determined using a 2D triangle node location algorithm [14,15] . For example, triangle T_{13} contains node N_{13}^S, triangle T_{17} contains node N_{17}^S, and so forth.

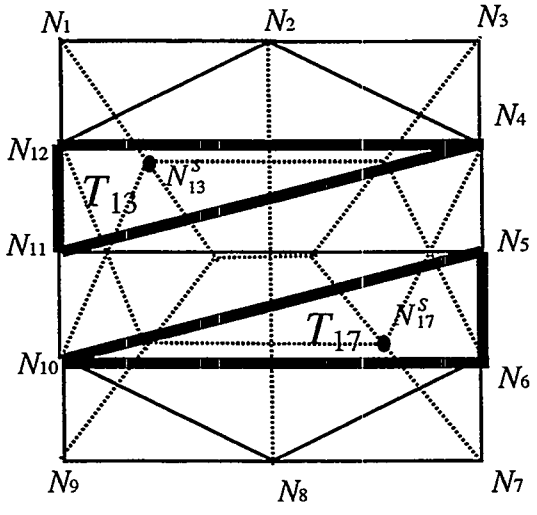


Figure 2 Background mesh with source area mesh connectivity (dotted lines) projected to parametric space

3.2.2 Barycentric Coordinates

Barycentric coordinates with respect to the background mesh triangles are used to locate each interior rib node. Two sets of barycentric coordinates are needed for each source area interior node N_i^S:

1. $\{a_i^S, b_i^S, b_i^S\}$: Barycentric coordinate of the uv location of N_i^S with respect to the source area uv locations of the corner nodes of the background mesh triangle T_i, and
2. $\{a_i^T, b_i^T, c_i^T\}$: Barycentric coordinates of the uv location of the cooresponding target area node N_i^T with respect to the target area uv locations of the corner nodes of the background mesh triangle T_i.

At each layer, the barycentric coordinates used to locate the new interior point are a linear interpolation of $\{a_i^S, b_i^S, c_i^S\}$ and $\{a_i^T, b_i^T, c_i^T\}$.

If the shape of the source and target areas are identical, then these two sets of barycentric coordinates will be identical. If, however, the shape of the source area varies at all from the shape of the target area, these barycentric coordinates could be quite different. In fact, if the source and target areas have drastically different shapes, then the background mesh triangle, T_i, after being elevated to the target area, may not contain N_i^T at all, causing one or more of the barycentric coordinates $\{a_i^T, b_i^T, c_i^T\}$ to be negative. This is perfectly acceptable. The barycentric coordinates of the N_i^S and N_i^T must be with respect to the same background mesh triangle, T_i since each interior node is located using a linear interpolation of $\{a_i^S, b_i^S, c_i^S\}$ and $\{a_i^T, b_i^T, c_i^T\}$.

3.2.3 Offset Distance

If either the source area or target area is not flat, then offset distances are needed to locate the interior nodes. Offset distance interpolation data is needed to catch source and target area curvature that is not captured by the background mesh. Since only boundary nodes are used to generate the background mesh, the background mesh only catches the curvature that is present on the boundary of the source and target areas (see Figure 9).

The offset distance is the distance, in 3d space, from an interior node to the background mesh triangle, T_i, containing the node. Two offset distances, d_i^S and d_i^T, are needed for each interior source area node. Let $\mathbf{P}_i^S$ be the location of the source area interior node, N_i^S, and $\mathbf{P}_i^T$ be the location of the target area node, N_i^T, corresponding to N_i^S. Let $(\mathbf{P}_i^{S1}, \mathbf{P}_i^{S2}, \mathbf{P}_i^{S3})$ be the source area xyz corner node coordinates of the background mesh triangle T_i. Let $(\mathbf{P}_i^{T1}, \mathbf{P}_i^{T2}, \mathbf{P}_i^{T3})$ be the corresponding target area xyz corner node coordinates of the background mesh triangle T_i. Offset distances d_i^S and d_i^T are determined using Equations 1 & 2. If the source area is flat, d_i^S will be zero. Likewise if the target area is flat, d_i^T will be zero.

$$d_i^S = \left\| (a_i^S \bullet \mathbf{P}_i^{S1} + b_i^S \bullet \mathbf{P}_i^{S2} + c_i^S \bullet \mathbf{P}_i^{S3}) - \mathbf{P}_i^S \right\| \quad [1]$$

$$d_i^T = \left\| (a_i^T \bullet \mathbf{P}_i^{T1} + b_i^T \bullet \mathbf{P}_i^{T2} + c_i^T \bullet \mathbf{P}_i^{T3}) - \mathbf{P}_i^T \right\| \quad [2]$$

3.3 Locating the Nodes on each Layer

Once the interpolation information is calculated, the location on each layer, $\mathbf{P}_i^L$, of each interior node, N_i^L, can be determined. For each new interior node N_i^L, four steps are required to determine its location. The first step is to linearly interpolate the Barycentric coordinates and offset distances calculated from the source and target areas. That is, let $\{a_i^L, b_i^L, c_i^L\}$ be the linear interpolation of $\{a_i^S, b_i^S, c_i^S\}$ and $\{a_i^T, b_i^T, c_i^T\}$ for node i on layer L. Also, let d_i^L be the linear interpolation of d_i^S and d_i^T. If the layers during the sweep are evenly spaced, the interpolation parameter can simply be the layer number. Otherwise a distance along the sweep path over the total sweep distance can be used as the interpolation parameter.

The second step is to elevate the background mesh triangle, T_i, containing the corresponding source area interior node to the current layer L. This is done by retrieving the corresponding boundary nodes at layer L from the boundary ribs that originate at the nodes defining T_i on the source area (see Figure 3). Let $(\mathbf{P}_i^{L1}, \mathbf{P}_i^{L2}, \mathbf{P}_i^{L3})$ be the locations of these three nodes defining the triangle T_i at layer L.

The third step is to define $\vec{\mathbf{V}}_i^L$ which is used with the offset distance d_i^L to specify the location of N_i^L to compensate for source and target area curvature. $\vec{\mathbf{V}}_i^L$ is calculated using the following:

$$\vec{\mathbf{V}}_i^L = a_i^L \bullet \vec{\mathbf{V}}_i^{L1} + b_i^L \bullet \vec{\mathbf{V}}_i^{L2} + c_i^L \bullet \vec{\mathbf{V}}_i^{L3} \quad [3]$$

where $[\vec{\mathbf{V}}_i^{L1}, \vec{\mathbf{V}}_i^{L2}, \vec{\mathbf{V}}_i^{L3}]$ are unit vectors in the direction of the sweep at $(\mathbf{P}_i^{L1}, \mathbf{P}_i^{L2}, \mathbf{P}_i^{L3})$ (see Figure 3).

The final step is to use the following equation to determine the location, P_i^L, of interior node, N_i^L, on layer L of an interior rib.

$$\mathbf{P}_i^L = (a_i^L \bullet \mathbf{P}_i^{L1} + b_i^L \bullet \mathbf{P}_i^{L2} + c_i^L \bullet \mathbf{P}_i^{L3}) + d_i^L \bullet \vec{\mathbf{V}}_i^L \quad [4]$$

Equation 4 calculates the final location of the interior nodes. No smoothing is required. Equation 4 interpolates the location of the interior nodes directly from the location of the boundary nodes on the current layer L. The locations of the nodes on the layers directly above and below a particular layer have no effect on the node locations of that layer. As a result, the locations of the nodes on each layer twist, turn, and rotate along with the boundary nodes of that layer completely independent of the volume's global orientation. This allows BMSweep to sweep volumes that have dramatic twists, turns and changes in cross-sectional area (see examples in section 4.0).

As the interior nodes are created, they are stored in order of increasing layers on each interior rib so as to facilitate element creation.

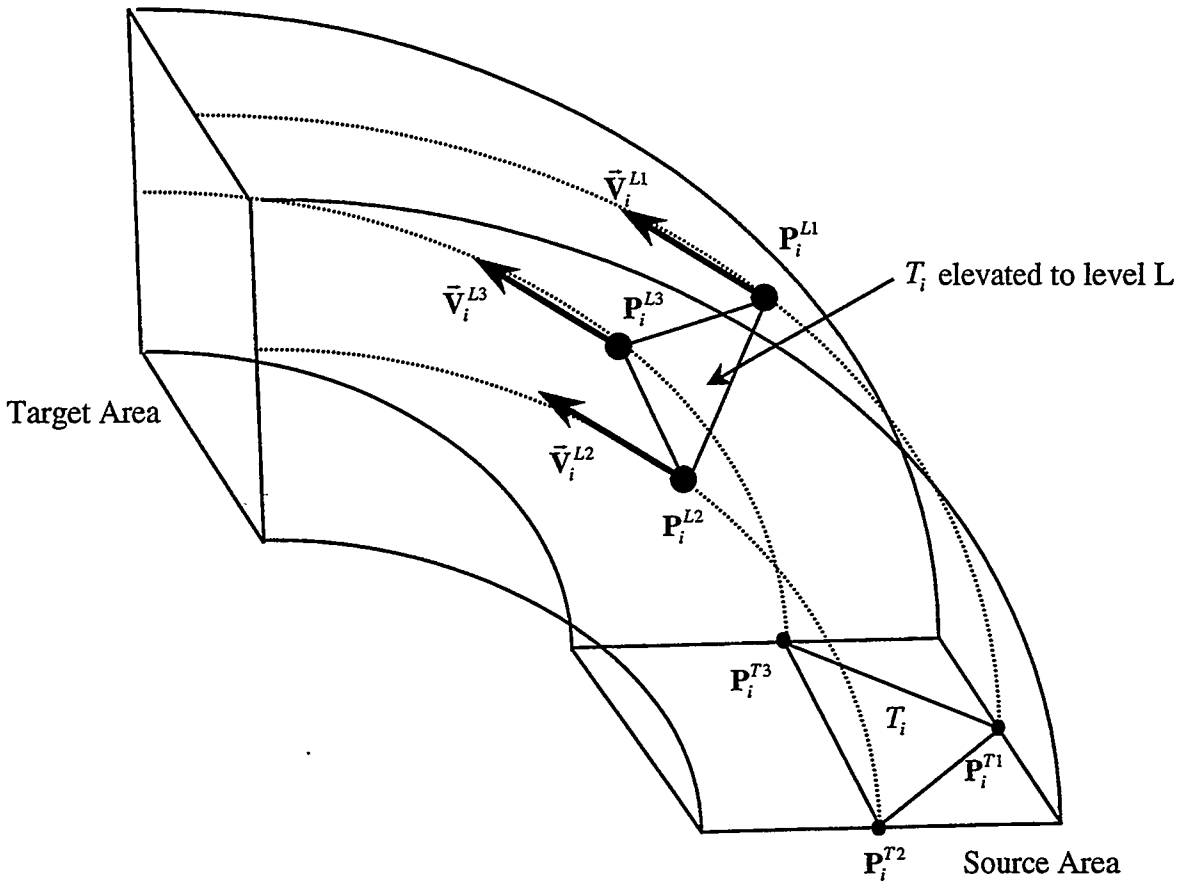


Figure 3 Vectors used to calculate node locations at layer L

3.4 Creating the new Elements.

Once all of the interior nodes have been located and placed on the interior ribs, the formation of the elements can begin. Each facet on the source area is individually processed by determining the nodes on the next layer that are on the same rib as each facet node. This process is repeated for each source area facet on each layer until all volume elements are created.

4.0 Examples

Figure 4 through Figure 11 are examples of volumes swept using BMSweep.

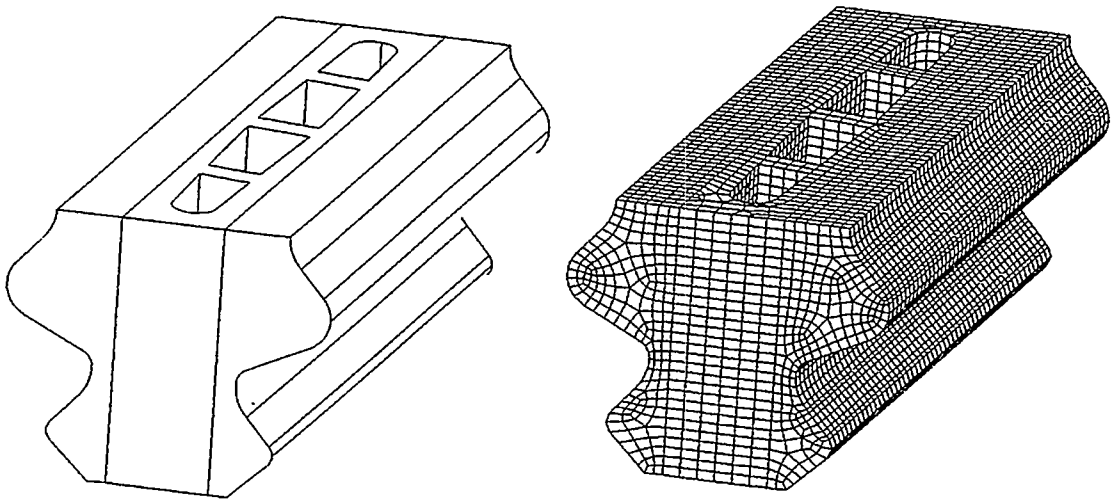


Figure 4 *BMSweep sweeping three adjacent volumes in different directions. CAD model courtesy of Pratt & Whitney*

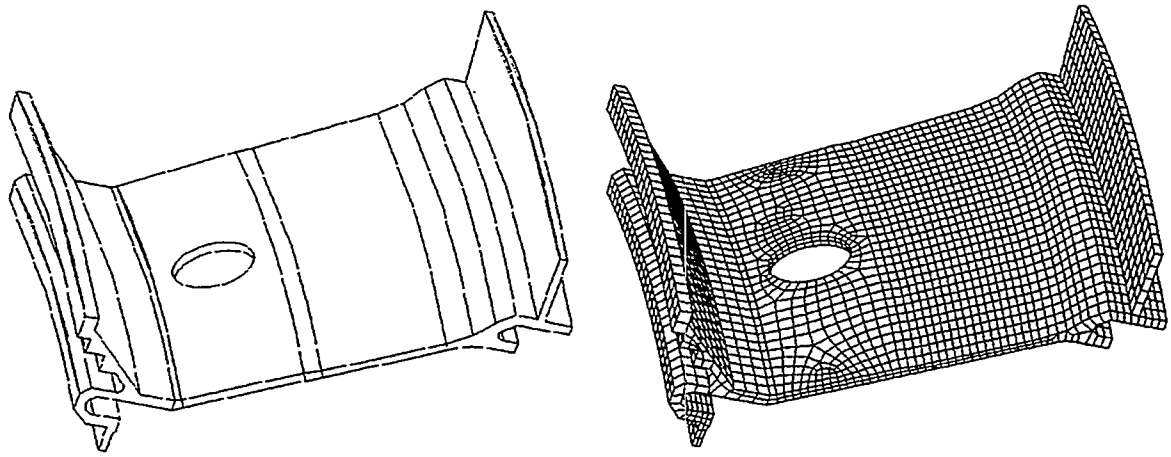


Figure 5 *BMSweep sweeping six adjacent volumes in different directions . CAD model courtesy of Pratt & Whitney*

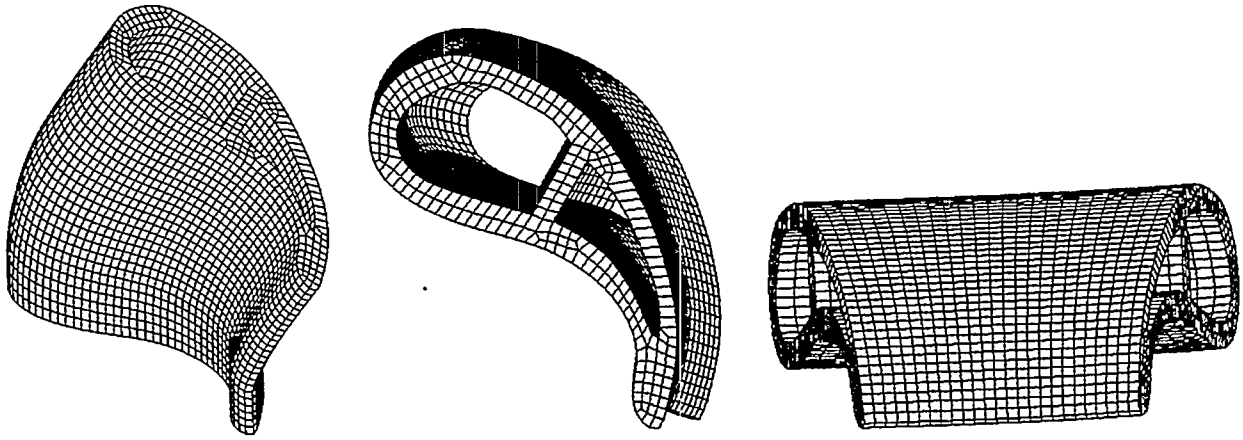


Figure 6 *BMSweep sweeping volume with irregular sweep direction*

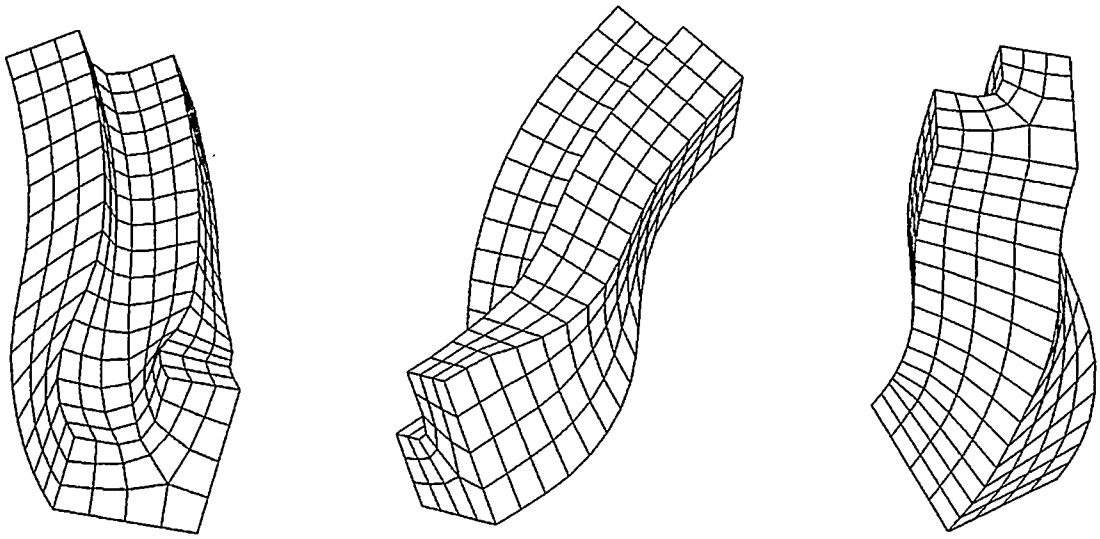


Figure 7 BMSweep sweeping volume with twisted and curved sweep direction

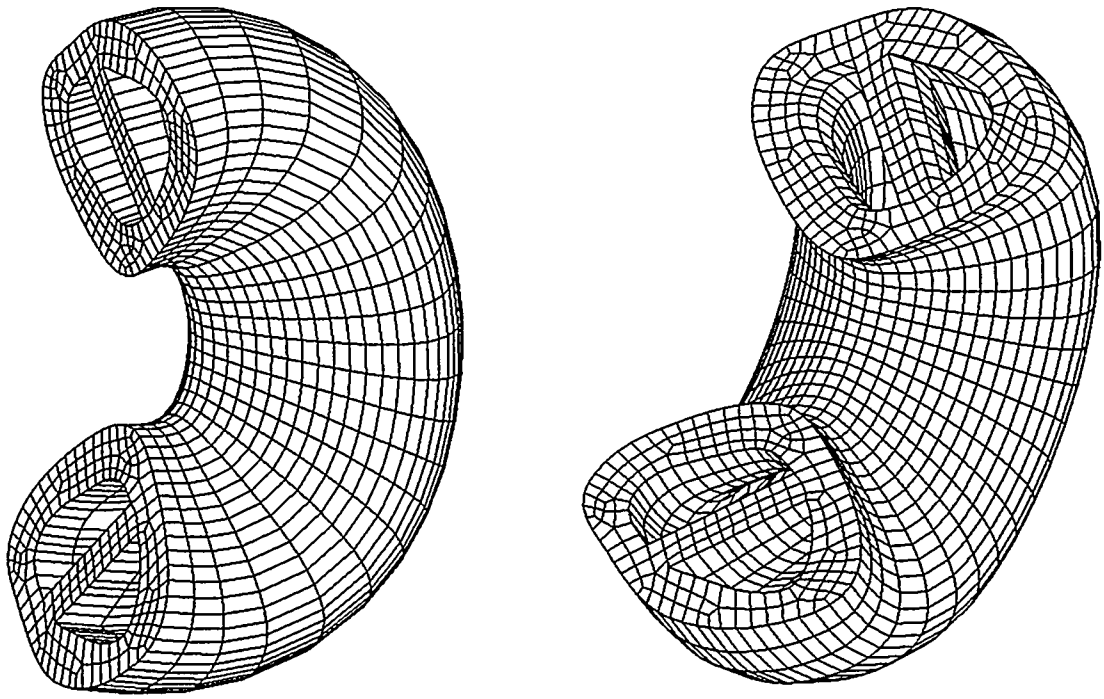


Figure 8 BMSweep sweeping thin walled volume with curved sweep direction

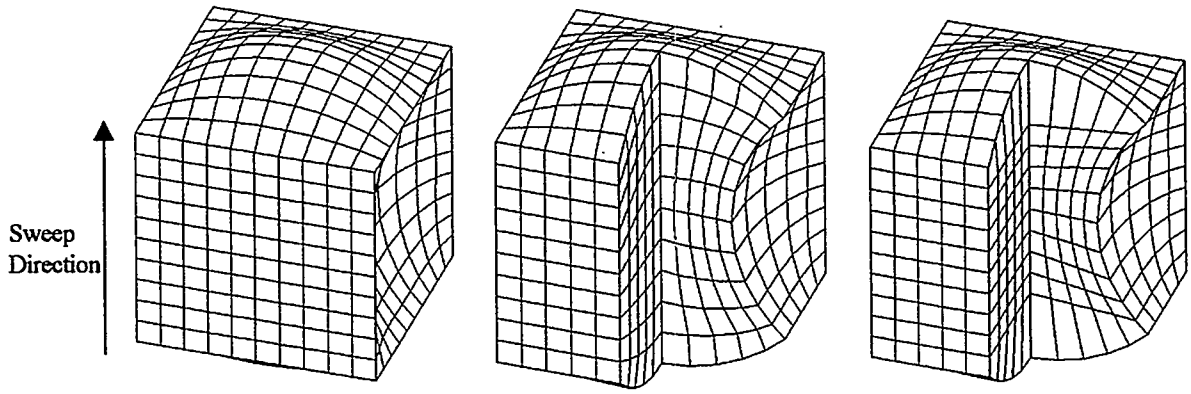


Figure 9 BMSweep sweeping volume using offset distance interpolation

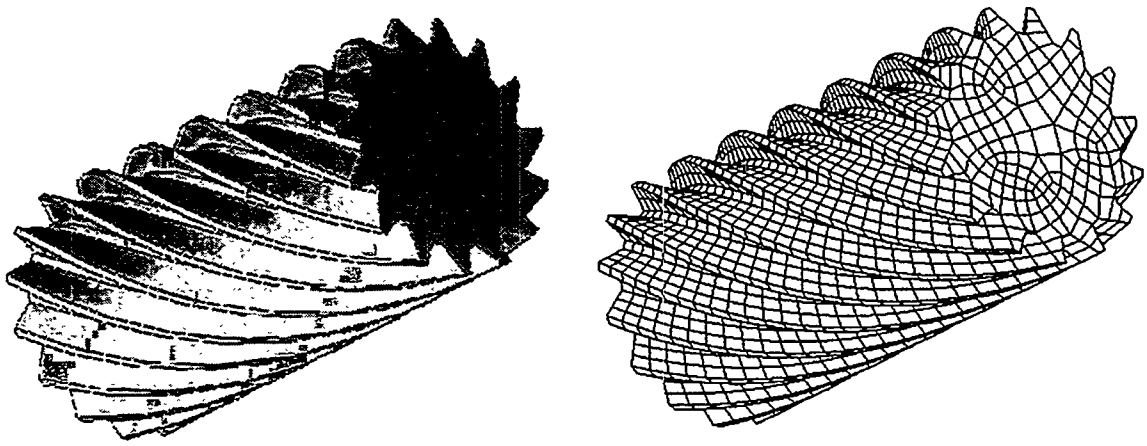


Figure 10 BMSweep sweeping volume with 150 degree rotation in sweep direction

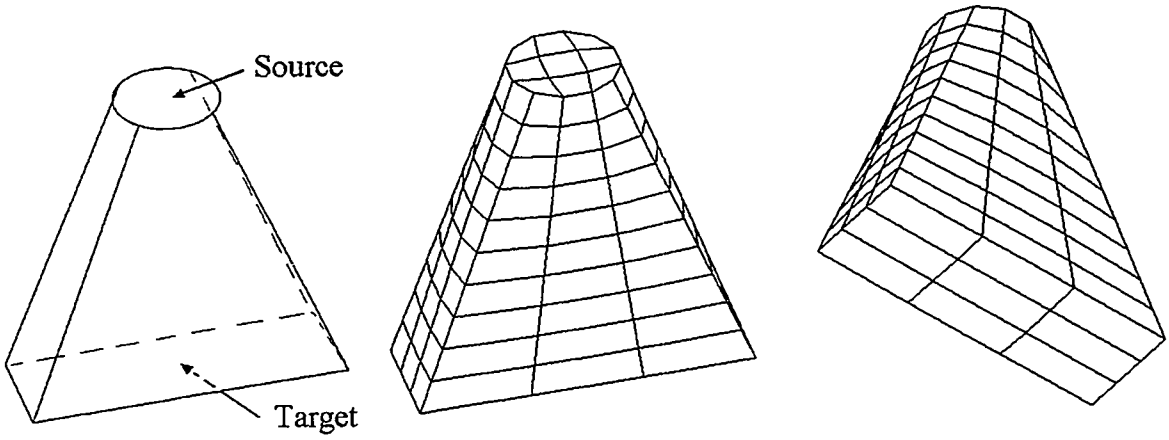


Figure 11 BMSweep sweeping volume with circular source and rectangular target

5.0 Conclusions

BMSweep, a method for determining interior node locations during sweeping, has been presented. BMSweep uses background mesh interpolation to locate the interior nodes. BMSweep is orientation insensitive and meshes two and one half dimensional volumes without requiring flat or parallel faces. In addition, the cross section of the sweep can twist, turn, and rotate freely. The shape of the cross section can also vary along the sweep path as long the cross section remains topologically constant. The algorithm has been extensively tested for robustness and has proven able to mesh a large variety of complex models. Coupled with volume decomposition, BMSweep can be used to mesh a large class of three dimensional volumes.

6.0 References

[1] Cifuentes, A. O. and A. Kalbag, 1992, "A Performance Study of Tetrahedral and Hexahedral Elements in 3-D Finite Element Structural Analysis." *Finite Elements in Analysis and Design*, Vol. 12, pp. 313-318.

[2] Benzley, S. E.; Perry, E.; Merkley, Karl; Clark, B. and Sjaardema, G., 1995, "A Comparison of All-Hexahedral and All-Tetrahedral Finite Element Meshes for Elastic and Elasto-Plastic Analysis," *Proceedings, 4th International Meshing Roundtable, Sandia National Labratories*, October 1995, pp. 179-191.

[3] Mitchell, S. A., 1996, "A Characterization of the Quadrilateral Meshes of a Surface Which Admit a Compatible Hexahedral Mesh of the Enclosed Volume." *Proceedings, 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS '96), Lecture Nodes in Computer Science 1046*, Springer, pp. 465-476.

[4] Blacker, T., 1996, "The Cooper Tool," *Proceeding, 5th International Meshing Roundtable 96*, pp. 13-29.

[5] Mingwu, L. and Benzley, S. E., 1996, "A Multiple Source and Target Sweeping Method for Generating All Hexahedral Finite Element Meshes," *Proceedings, 5th International Meshing Roundtable 96*, pp. 217-225.

[6] Liu, S. S. and Gadh, R., 1997, "Automatic Hexahedral Mesh Generation by Recursive Convex and Swept Volume Decomposition," *Proceeding, 6th International Meshing Roundtable 97*, pp. 217-231.

[7] Shih, B. and Sakurai, H., 1997, "Shape Recognition and Shape-Specific Meshing for Generating All Hexahedral Meshes," *Proceedings, 6th International Meshing Roundtable 97*, pp. 197-209.

[8] White, D. W., Mingwu, L., and Benzley, S. E., 1995, "Automated Hexahedral Mesh Generation by Virtual Decomposition," *Proceeding, 4th International Meshing Roundtable 95*, pp. 165-176.

[9] Chiba, N., Nishigaki, I., Yamashita, Y., Takizawa, C., and Fujishiro, K., 1996, "An Automatic Hexahedral Mesh Generation System Based on the Shape-Recognition and Boundary-Fit Methods," *Proceedings, 5th International Meshing Roundtable 96*, pp. 281-290.

[10] Canann, S. A., Tristano, J. R., and Staten, M. L., 1998, "An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes," *Proceedings, 7th International Meshing Roundtable 98.*

[11] Tristano, J. R., Owen, S. J., and Canann, S. A., 1998, "Advancing Front Surface mesh Generation in Parametric Space Using Riemannian Surface Definition," *Proceedings, 7th International Meshing Roundtable 98.*

[12] Owen, S. J., Staten, M. L., Canann, S. A., and Saigal, S., 1998, "Advancing Front Quadrilateral Meshing Using Triangle Transformations," *Proceedings, 7th International Meshing Roundtable 98.*

[13] White, D. R., 1996, "Automatic, Quadrilateral and Hexahedral Meshing of Pseudo-Cartesian Geometries using Virtual Decomposition," Master's Thesis, Brigham Young University, August 1996.

[14] Mirante, A. and Weingarten, N., 1982, "The radial sweep algorithm for constructing triangulated irregular networks," *IEEE Computer Graphics and Applications*, Vol. 2, No. 3, May, pp. 11-21.

[15] Lee, D. T. and Schacter, B. J., 1980, "Two algorithms for constructing a Delaunay triangulation," *International Journal of Computer and Information Sciences*, Plenum Press, New York, London, Vol. 9, No. 3, June, pp. 219-242.

[16] Watson, D. F., 1981, "Computing the Delaunay tesselation with application to Voroni Polytopes," Computer Journal, Vol. 24, pp. 167-172.

On Optimal Bilinear Quadrilateral Meshes

Ed D'Azevedo*

Abstract.

The novelty of this work is in presenting interesting error properties of two types of asymptotically "optimal" quadrilateral meshes for bilinear approximation. The first type of mesh has an error equidistributing property where the maximum interpolation error is asymptotically the same over all elements. The second type has faster than expected "super-convergence" property for certain saddle-shaped data functions. The "super-convergent" mesh may be an order of magnitude more accurate than the error equidistributing mesh. Both types of mesh are generated by a coordinate transformation of a regular mesh of squares. The coordinate transformation is derived by interpreting the Hessian matrix of a data function as a metric tensor. The insights in this work may have application in mesh design near known corner or point singularities.

keywords. anisotropic mesh, quadrilateral mesh, mesh generation, super-convergence.

1 Introduction

This paper presents the theoretical effectiveness of two types of "optimal" bilinear quadrilateral meshes. The novelty of this work is in presenting interesting error properties of two types of asymptotically "optimal" quadrilateral meshes for bilinear approximation. The first type of mesh has an error equidistributing property where the maximum interpolation error is asymptotically the same over all elements. The second type has faster than expected "super-convergence" property for certain non-convex saddle-shaped data functions. The "super-convergent" mesh may be an order of magnitude more accurate than the error equidistributing mesh. Both types of meshes are generated by a coordinate transformation of a regular mesh of squares. The coordinate transformation is derived by interpreting the Hessian matrix of a data function as a metric tensor. This work is a basic study on optimal meshes with the intention of gaining insight into the more complex meshing problem in surface approximation and finite element analysis especially near corner or point singularities.

For simplicity, we consider the problem of interpolating a given smooth data function with continuous piecewise bilinear quadrilaterals over a domain to satisfy a given error tolerance. A mesh that achieves this error tolerance with the *fewest* elements is defined to be optimally efficient. Intuitively, one would expect smaller and denser elements in regions where the function has sharp peaks or large variations.

Provably optimal triangular meshes [2, 4] have been produced by anisotropic mesh transformation. Anisotropic mesh transformation is emerging as an effective technique for unstructured grid generation where the vertex distribution is highly non-uniform. The central idea is to control the element shapes and sizes by specifying a symmetric metric tensor that measures the approximation error. The metric tensor determines the corresponding anisotropic transformation. The anisotropic mesh is then the image of a uniform mesh of optimal shape elements under the anisotropic transformation. Simpson [9] gives a survey on anisotropic meshes. Nadler [6], D'Azevedo and Simpson [3, 4], and D'Azevedo [2] have studied *local* anisotropic transformation for generating optimally efficient triangular meshes. Numerous works such as Borouchaki [1], Peraire [7], and Shimada [8], have used the Hessian matrix as a metric tensor for anisotropic mesh generation. In this paper we apply a similar analysis to bilinear approximation on quadrilateral patches.

An outline of the paper follows. In §2, we present a simple local quadratic model for error analysis and introduce the coordinate transformation to the "isotropic" space. In §3 we show a square over the isotropic space is the most efficient shape to minimize the ratio of Error/Area. A regular mesh of squares over the isotropic space would correspond to an optimally efficient mesh in the original space. Section 4 states a classical result in differential geometry on the conditions for finding the anisotropic transformation $[\tilde{x}(x,y), \tilde{y}(x,y)]$ for a general data function. Results of numerical experiments are presented in §5 to demonstrate the error equidistributing property and the effectiveness of the super-convergent meshes.

*Computer Science and Mathematics Division, Oak Ridge National Laboratory, P. O. Box 2008, Oak Ridge, TN 37831–6367. Work was funded in part by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corp.

2 Quadratic model

We shall consider a local analysis where we assume the data function $f(x,y)$ in the neighborhood of (x_c,y_c) is well approximated by its quadratic Taylor expansion,

$$\begin{aligned} f(x,y) &= f(x_c+dx, y_c+dy) \\ &\approx f(x_c,y_c)+\nabla f(x_c,y_c)[dx,dy]+\frac{1}{2}[dx,dy]H[dx,dy]^t\,. \end{aligned} \tag{1}$$

The function is convex if $\det(H)>0$ and saddle-shaped if $\det(H)<0$. The key insight in [2] is in interpreting the Hessian matrix H in (1) as a symmetric metric tensor. Let the symmetric Hessian matrix be diagonalizable as

$$\begin{aligned} H &= Q^t\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}Q = S^t\begin{bmatrix}1 & 0\\ 0 & \epsilon\end{bmatrix}S, \quad \text{where } \epsilon=\text{sign}(\det(H)), \\ S &= \begin{bmatrix}\sqrt{|\lambda_1|} & 0\\ 0 & \sqrt{|\lambda_2|}\end{bmatrix}Q, \quad \text{and } Q \text{ is orthogonal, } Q^tQ=I\,. \end{aligned} \tag{2}$$

Note that transformation S is essentially a rotation to align eigenvectors along the coordinate axes then followed by a simple scaling. Under this transformation S, the expression $[dx,dy]H[dx,dy]^t$ reduces to $(d\bar{x})^2+\epsilon(d\bar{y})^2$, where $[\bar{x},\bar{y}]^t=S[x,y]^t$. Over the transformed space $(\bar{x}(x,y),\bar{y}(x,y))$, the Hessian matrix is reduced to a simple form (2) with no preference for any direction. We shall call this transformed space the "isotropic" space. We shall use a quadratic data function to derive a simple model for deriving the maximum interpolation error over a bilinear quadrilateral patch.

3 Quadrilateral patch

The bilinear interpolant over a quadrilateral element is given by the isoparametric formulation (commonly used in finite element analysis) over the normalized (p,q)-space on the unit square, $0\le p,q\le 1$. Basis functions are

$$\begin{aligned} \phi_1(p,q)&=(1-p)(1-q), & \phi_2(p,q)&=p(1-q),\\ \phi_3(p,q)&=pq, & \phi_4(p,q)&=(1-p)q\,, \end{aligned} \tag{3}$$

that satisfy $\phi_i(x_j,y_j)=\delta_{ij}$, and sum to one, $1=\sum_{i=1}^{i=4}\phi_i(p,q)$.

Mapping from (p,q) to the original (x,y)-space is by

$$\begin{aligned} x(p,q) &= x_1\phi_1(p,q)+x_2\phi_2(p,q)+x_3\phi_3(p,q)+x_4\phi_4(p,q)\\ y(p,q) &= y_1\phi_1(p,q)+y_2\phi_2(p,q)+y_3\phi_3(p,q)+y_4\phi_4(p,q) \end{aligned} \tag{4}$$

that maps vertex $(0,0)$ to (x_1,y_1), vertex $(1,0)$ to (x_2,y_2), $(1,1)$ to (x_3,y_3) and $(0,1)$ to (x_4,y_4). The bilinear interpolant (over (p,q)-space) is given by

$$p_b(x(p,q),y(p,q)) = \sum_{i=1}^{i=4} f(x_i,y_i)\phi_i(p,q)\,. \tag{5}$$

The error function for quadratic interpolation over a *parallelogram* can be shown by direct algebraic expansion (see Appendix A) to be

$$\begin{aligned} E_Q(p,q) &= p_b(x(p,q),y(p,q))-f(x(p,q),y(p,q))\\ &= \mathcal{E}_Q-\frac{1}{2}\left(\mu_1(p-p_c)^2+\mu_2(q-q_c)^2\right)\,, \end{aligned} \tag{6}$$

with centroid at $[p_c,q_c]=[\frac{1}{2},\frac{1}{2}]$,

$$[u_x,u_y]=[x_2-x_1,y_2-y_1], \quad [v_x,v_y]=[x_4-x_1,y_4-y_1],$$

$$
\begin{aligned}
\mathcal{E}_Q &= E_Q(p_c, q_c) = \frac{1}{8}(\mu_1 + \mu_2)\ , \\
0 &= \frac{\partial}{\partial p} E_Q(p_c, q_c) = \frac{\partial}{\partial q} E_Q(p_c, q_c)\ , \qquad (7) \\
\mu_1 &= [u_x, u_y] H [u_x, u_y]^t, \quad \mu_2 = [v_x, v_y] H [v_x, v_y]^t\ .
\end{aligned}
$$

For a convex function ($\det(H) > 0$), μ_1 and μ_2 are positive, hence the maximum error is attained at the centroid $[p_c, q_c]$.

For the case of a general convex quadrilateral, the error expression is more complicated. However, we can show a square over the isotropic space is of optimal shape by minimizing the efficiency ratio (Error/Area). Since the isoparametric bilinear interpolant (5) exactly fits linear functions [5], the error attained at the centroid (x_c, y_c) can be written as

$$
\begin{aligned}
\mathcal{E}_M &= \frac{1}{4}\left(\sum_{i=1}^{i=4} \tfrac{1}{2}[x_i, y_i] H [x_i, y_i]^t\right) - \tfrac{1}{2}[x_c, y_c] H [x_c, y_c]^t \qquad (8) \\
&= \frac{1}{8}\left(\sum_{i=1}^{i=4} \left([x_i, y_i] H [x_i, y_i]^t - [x_c, y_c] H [x_c, y_c]^t\right)\right) \\
[x_c, y_c] &= [(x_1 + x_2 + x_3 + x_4)/4, (y_1 + y_2 + y_3 + y_4)/4]\ . \qquad (9)
\end{aligned}
$$

This expression can be further simplified over the isotropic space where H is the identity,

$$
\begin{aligned}
\mathcal{E}_M &= \frac{1}{8}\left(\sum_{i=1}^{i=4} ((\tilde{x}_i^2 + \tilde{y}_i^2) - (\tilde{x}_c^2 + \tilde{y}_c^2))\right) \\
&= \frac{1}{8}\left((\tilde{x}_1^2 + \tilde{x}_2^2 + \tilde{x}_3^2 + \tilde{x}_4^2) - 4\tilde{x}_c^2 + (\tilde{y}_1^2 + \tilde{y}_2^2 + \tilde{y}_3^2 + \tilde{y}_4^2) - 4\tilde{y}_c^2\right) \\
&= \frac{1}{8}(L_1^2 + L_2^2 + L_3^2 + L_4^2), \quad \text{with } L_i^2 = (\tilde{x}_i - \tilde{x}_c)^2 + (\tilde{y}_i - \tilde{y}_c)^2,
\end{aligned}
$$

where $[\tilde{x}_i, \tilde{y}_i]^t = S[x_i, y_i]^t$ and $[\tilde{x}_c, \tilde{y}_c]^t = S[x_c, y_c]^t$ are the corresponding coordinates over the isotropic space. The area of this transformed convex quadrilateral is (see Figure 1)

$$
\text{Area} = \frac{1}{2}(L_1 L_2 \sin(\theta_1) + L_2 L_3 \sin(\theta_2) + L_3 L_4 \sin(\theta_3) - L_4 L_1 \sin(\theta_1 + \theta_2 + \theta_3))\ .
$$

Since the isotropic transformation S in (2) is a rotation followed by a rescaling of coordinate axis, the area of quadrilateral over the isotropic space is scaled by $\sqrt{|\lambda_1 \lambda_2|} = \sqrt{|\det(H)|}$ (intrinsic to H). By calculus, we can show this ratio of $\mathcal{E}_M$/Area is minimized and *attained* by a square with $L_1 = L_2 = L_3 = L_4$ and $\theta_1 = \theta_2 = \theta_3 = \pi/4$. Hence the most efficient shape among all *general* convex bilinear quadrilaterals is a square over the isotropic space with an efficiency ratio of 1/4.

If $f(x, y)$ is saddle-shaped ($\det(H) < 0$), the error expression for a parallelogram is still

$$
E_Q(p, q) = \frac{1}{8}(\mu_1 + \mu_2) - \frac{1}{2}(\mu_1 (p - p_c)^2 + \mu_2 (q - q_c)^2)\ .
$$

Under the anisotropic transformation S,

$$
\mu_1 = \tilde{u}_x^2 - \tilde{u}_y^2, \quad \mu_2 = \tilde{v}_x^2 - \tilde{v}_y^2\ , \quad \begin{bmatrix} \tilde{u}_x & \tilde{v}_x \\ \tilde{u}_y & \tilde{v}_y \end{bmatrix} = S \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \qquad (10)
$$

For a square over the isotropic space, we have

$$
[u_x, u_y] = [L, 0], \quad [v_x, v_y] = [0, L], \quad \mu_1 = L^2, \quad \mu_2 = -L^2\ ,
$$

$$
E_Q(p, q) = -\frac{1}{2}(L^2 (p - \tfrac{1}{2})^2 - L^2 (q - \tfrac{1}{2})^2) = \frac{L^2}{2}((q - \tfrac{1}{2})^2 - (p - \tfrac{1}{2})^2)\ .
$$

The maximum error is $L^2/8$ and attained at $(p, q) = (\frac{1}{2}, 1)$ or $(\frac{1}{2}, 0)$.

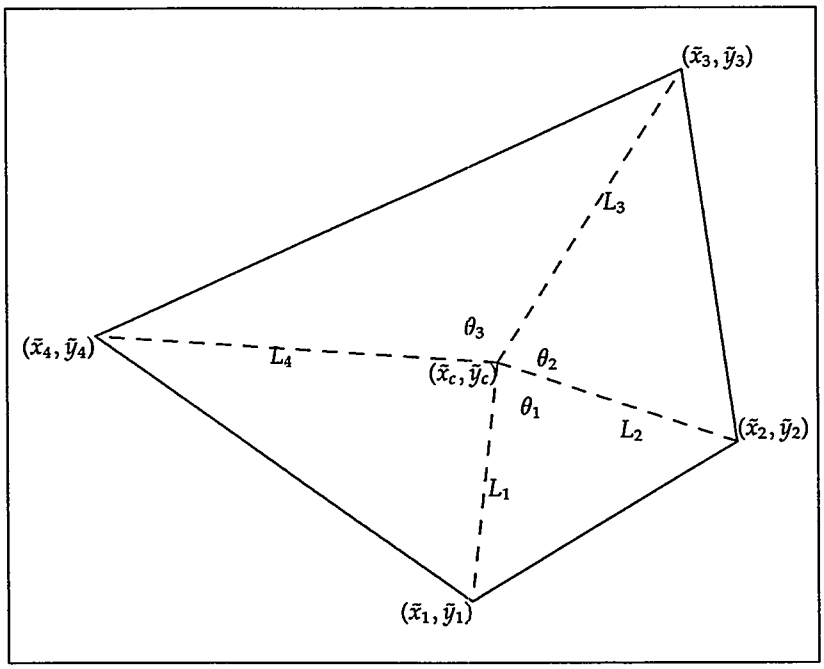


Figure 1: Convex quadrilateral over isotropic space.

Note that both μ_1 and μ_2 vanish for

$$[\tilde{u}_x, \tilde{u}_y] \quad = \quad [L, L], \quad [\tilde{v}_x, \tilde{v}_y] = [-L, L] \quad , \tag{11}$$

which correspond to a square rotated by $\pi/4$. The above indicates an "exact fit" ($E_Q(p,q) = 0$) if $\mu_1 = \mu_2 = 0$. This suggests bilinear approximation has higher than expected accuracy and the simple quadratic model is inadequate to fully capture the error properties in this case.

To summarize, a square over the isotropic space in any orientation is of optimal shape for the convex ($\det(H) > 0$) case, and a square rotated by $\pi/4$ is the optimal shape for the saddle-shaped ($\det(H) < 0$) case. A regular square mesh over the isotropic space would correspond to a error equidistributing mesh where each patch incurs the same maximum error. For a saddle-shaped data function $\det(H) < 0$, a regular mesh of squares rotated $\pi/4$ would have higher than expected accuracy.

4 Differential Geometry

The *constant* Hessian Matrix $H = \{h_{ij}\}$ in (1) determines the coordinate transformation S that maps $[\tilde{x}, \tilde{y}]^t = S[x, y]^t$ so that

$$[dx, dy]H[dx, dy]^t = d\tilde{x}^2 + \epsilon d\tilde{y}^2 \, .$$

For more general functions, we may view the Hessian matrix $H(x, y)$ as a metric tensor for measuring the interpolation error $[dx, dy]H[dx, dy]^t$. Thus we need to determine $[\tilde{x}(x, y), \tilde{y}(x, y)]$, a *continuous* transformation that *globally* satisfies $[dx, dy]H[dx, dy]^t = d\tilde{x}^2 + \epsilon d\tilde{y}^2$ for infinitestimals $[dx, dy]$. The transformation $[\tilde{x}(x, y), \tilde{y}(x, y)]$ should satisfy

$$h_{11}dx^2 + 2h_{12}dxdy + h_{22}dy^2 = \left(\frac{\partial \tilde{x}}{\partial x}dx + \frac{\partial \tilde{x}}{\partial y}dy\right)^2 + \epsilon\left(\frac{\partial \tilde{y}}{\partial x}dx + \frac{\partial \tilde{y}}{\partial y}dy\right)^2 \, ,$$

$$
\begin{aligned}
h_{11} &= \frac{\partial^2}{\partial x^2} f(x,y) = \left(\frac{\partial \tilde{x}}{\partial x}\right)^2 + \epsilon \left(\frac{\partial \tilde{y}}{\partial x}\right)^2 , \\
h_{12} &= \frac{\partial^2}{\partial x \partial y} f(x,y) = \frac{\partial \tilde{x}}{\partial x}\frac{\partial \tilde{y}}{\partial y} + \epsilon \frac{\partial \tilde{y}}{\partial x}\frac{\partial \tilde{x}}{\partial y} , \\
h_{22} &= \frac{\partial^2}{\partial y^2} f(x,y) = \left(\frac{\partial \tilde{x}}{\partial y}\right)^2 + \epsilon \left(\frac{\partial \tilde{y}}{\partial y}\right)^2 .
\end{aligned} \tag{12}
$$

The conditions for finding the anisotropic coordinate transformation $[\tilde{x}(x,y), \tilde{y}(x,y)]$ are given by a classical result in differential geometry for characterizing a "flat" space [10]: that the Riemann-Christoffel tensor formed from the metric tensor H is identically zero. In this case, a sufficient condition is for $H = \{h_{ij}\}$ to satisfy

$$
K_1 h_{11} + K_2 h_{12} + K_3 h_{22} = 0 \tag{13}
$$

for some constants K_1, K_2, K_3. In particular, (13) is satisfied by harmonic functions ($h_{11} + h_{22} = 0$). The coordinate transformation $[\tilde{x}(x,y), \tilde{y}(x,y)]$ may be found by solving an initial value ordinary differential equation. The details for computing the anisotropic coordinate transformation $[\tilde{x}(x,y), \tilde{y}(x,y)]$ are described in [2].

5 Numerical Experiments

In this section, we demonstrate the effectiveness of a super-convergent mesh for interpolation over bilinear quadrilaterals on several harmonic functions. To clearly illustrate the error equidistributing properties, only elements entirely interior to the unit square are generated to simplify the presentation.

Example 1. A logarithmic singularity at $(x_0, y_0) = (0.5, -0.2)$,

$$
f(x,y) = \ln((x-x_0)^2 + (y-y_0)^2)/2, \quad \det(H) = -((x-x_0)^2 + (y-y_0)^2)^{-2} .
$$

Coordinate transformation is

$$
\tilde{x}(x,y) = \arctan(y-y_0, x-x_0), \quad \tilde{y}(x,y) = \ln((x-x_0)^2 + (y-y_0)^2)/2 .
$$

Example 2. A near singularity at $(x_0, y_0) = (0.5, -0.2)$,

$$
f(x,y) = \frac{(x-x_0)^2 - (y-y_0)^2}{((x-x_0)^2 + (y-y_0)^2)^2}, \quad \det(H) = -36((x-x_0)^2 + (y-y_0)^2)^{-4} .
$$

Coordinate transformation is

$$
\tilde{x}(x,y) = \sqrt{6}\left(1 - \frac{x-x_0}{(x-x_0)^2 + (y-y_0)^2}\right), \quad \tilde{y}(x,y) = \sqrt{6}\frac{y-y_0}{(x-x_0)^2 + (y-y_0)^2} .
$$

Example 3. A more severe near singularity at $(x_0, y_0) = (0.5, -0.2)$,

$$
f(x,y) = \frac{((x-x_0)^2 + (y-y_0)^2)^2 \div 8(x-x_0)^2(y-y_0)^2}{((x-x_0)^2 + (y-y_0)^2)^4}, \quad \det(H) = -400((x-x_0)^2 + (y-y_0)^2)^{-6} .
$$

Coordinate transformation is

$$
\tilde{x}(x,y) = \sqrt{5}\left(1 + \frac{(y-y_0)^2 - (x-x_0)^2}{((x-x_0)^2 + (y-y_0)^2)^2}\right), \quad \tilde{y}(x,y) = 2\sqrt{5}\frac{(x-x_0)(y-y_0)}{((x-x_0)^2 + (y-y_0)^2)^2}
$$

Example 4. Potential flow around a corner at $(x_0, y_0) = (0.5, 0.5)$ where $n = \pi/\alpha = 16/31$, $\alpha = 2\pi - \pi/16$ is the angle of corner, and $\theta = \arctan(y, x)$,

$$
f(x,y) = ((x-x_0)^2 + (y-y_0)^2)^{n/2}\cos(n\theta), \quad \det(H) = -\frac{57600}{923521}((x-x_0)^2 + (y-y_0)^2)^{-46/31} .
$$

Table 1: Summary of results for Example 1.

	Minimum error	Median error	90 percentile	Maximum error	Number of elements
Mesh I	3.56e-04	3.56e-04	3.56e-04	3.56e-04	918
Mesh I	8.90e-05	8.90e-05	8.90e-05	8.90e-05	3841
Mesh I	2.22e-05	2.22e-05	2.22e-05	2.22e-05	15674
Mesh II	3.44e-06	3.44e-06	3.44e-06	3.44e-06	923
Mesh II	4.30e-07	4.30e-07	4.30e-07	4.30e-07	3847
Mesh II	5.37e-08	5.37e-08	5.37e-08	5.37e-08	15695

Table 2: Summary of results for Example 2.

	Minimum error	Median error	90 percentile	Maximum error	Number of elements
Mesh I	1.30e-02	1.30e-02	1.30e-02	1.30e-02	920
Mesh II	1.27e-04	1.79e-04	3.18e-04	6.93e-04	921

Coordinate transformation is

$$[\tilde{x}(x,y),\tilde{y}(x,y)] = \frac{\sqrt{15}}{2}((x-x_0)^2+(y-y_0)^2)^{4/31}\left[\sin(8\theta/31),\cos(8\theta/31)\right] .$$

The results of the experiments are summarized in Figures 2, 3, 4, 5 and in Tables 1, 2, 3 and 4. Mesh I is generated by a regular mesh of squares over the isotropic space. Mesh II is generated by a regular mesh of squares but with the $\pi/4$ rotation over the isotropic space to capture the super-convergent behavior. Both Mesh I and Mesh II have similar element size, element shape and density and differ mainly in the $\pi/4$ rotation. The error equidistributing meshes (Mesh I) are displayed in Figures 6, 8, 10 and 12. The super-convergent meshes (Mesh II) are displayed in Figures 7, 9, 11 and 13. The error profiles in 2, 3, 4 and 5 clearly show significant improvement in accuracy of Mesh II over Mesh I. The almost level error profile for Mesh I indicates an equidistribution of interpolation error evenly over all elements as predicted by our simple error model.

Note that Example 1 produces a simple radially symmetric mesh with a regular angular partition. Even in this simple case, a $\pi/4$ rotation yields substantial improvement in approximation accuracy.

Results on Table 1 and Table 3 show the expected $O(h^2)$ convergence rate for Mesh I. A four-fold increase of elements leads to a four-fold decrease in error. Results for Mesh II demonstrate a higher than $O(h^2)$ convergence. A four-fold increase of elements leads to an eight fold decrease in error. This suggests $O(h^3)$ convergence behavior for Mesh II.

In summary, we have derived a simple error model for bilinear approximation over a parallelogram. We used this model to motivate the generation of super-convergent meshes using an anisotropic coordinate transformation of a regular mesh of squares. The numerical experiments clearly demonstrate the effectiveness of the super-convergent mesh for certain non-convex data functions. The insight gained here might have application to mesh design near known point or corner singularities.

Table 3: Convergence test on Example 3.

	Minimum error	Median error	90 percentile	Maximum error	Number of elements
Mesh I	1.51e+00	1.51e+00	1.52e+00	1.56e+00	255
Mesh I	4.54e-01	4.54e-01	4.54e-01	4.60e-01	916
Mesh I	1.13e-01	1.13e-01	1.14e-01	1.15e-01	3837
Mesh I	2.84e-02	2.84e-02	2.84e-02	2.85e-02	15685
Mesh II	2.36e-02	4.06e-02	9.66e-02	5.09e-01	259
Mesh II	3.69e-03	6.69e-03	1.63e-02	9.64e-02	918
Mesh II	4.52e-04	8.29e-04	2.04e-03	1.44e-02	3834
Mesh II	5.53e-05	1.03e-04	2.54e-04	1.92e-03	15682

Table 4: Summary of results for Example 4.

	Minimum error	Median error	90 percentile	Maximum error	Number of elements
Mesh I	4.21e-4	4.21e-4	4.22e-4	4.26e-4	576
Mesh II	5.90e-6	9.90e-6	1.90e-5	3.97e-5	575

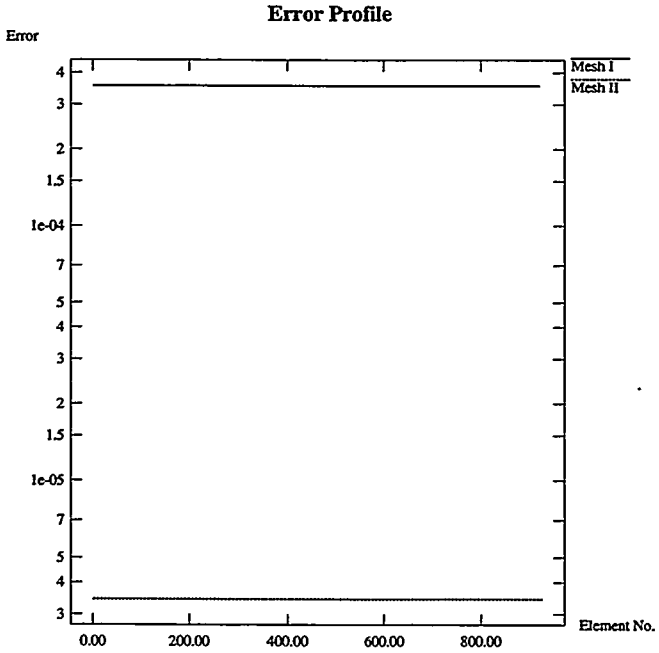


Figure 2: Error profiles for Example 1.

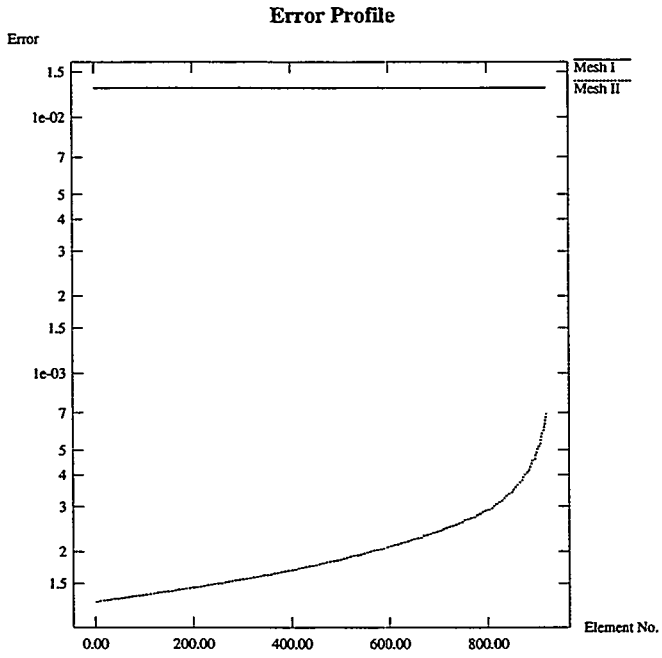


Figure 3: Error profiles for Example 2.

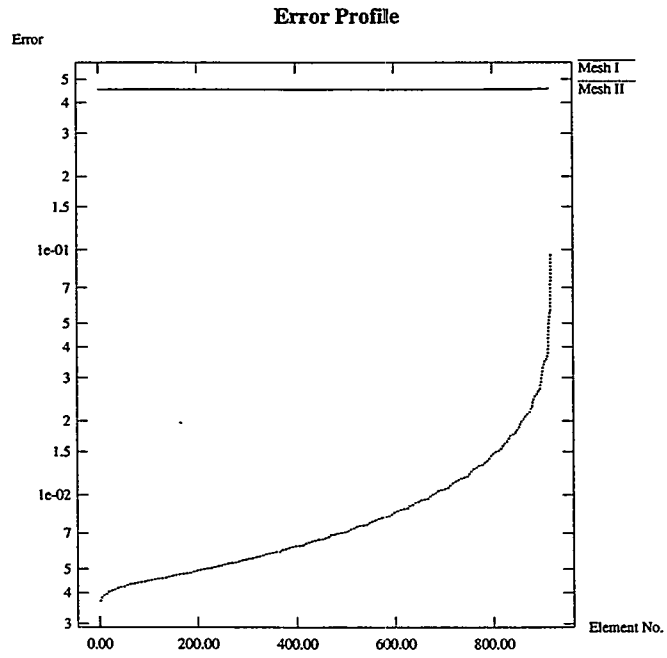


Figure 4: Error profiles for Example 3.

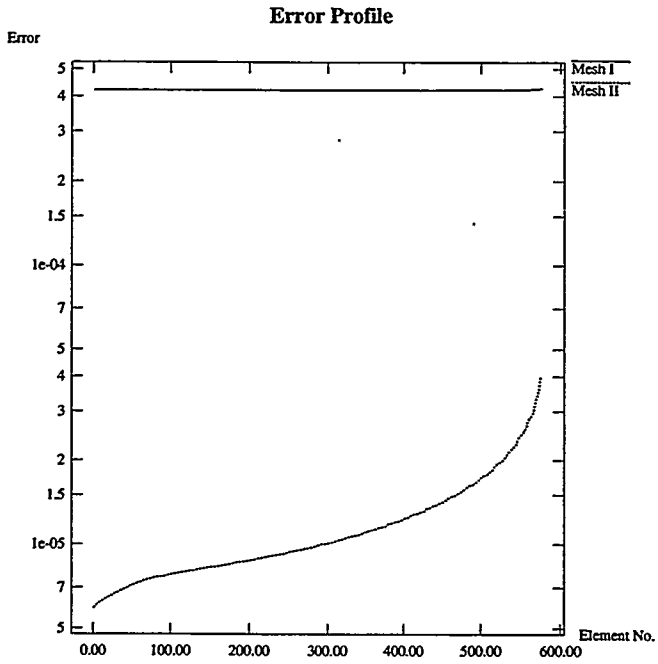


Figure 5: Error profiles for Example 4.

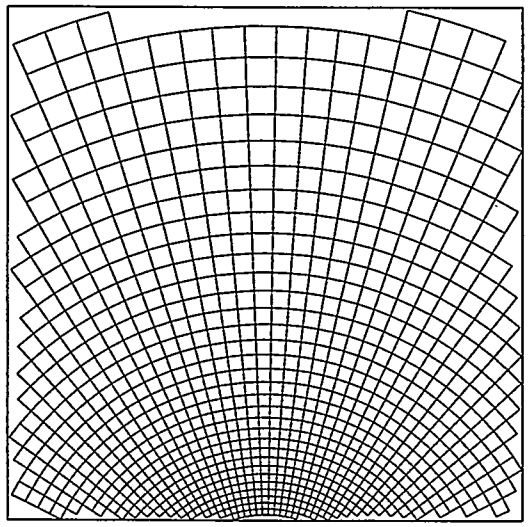


Figure 6: Mesh I for Example 1.

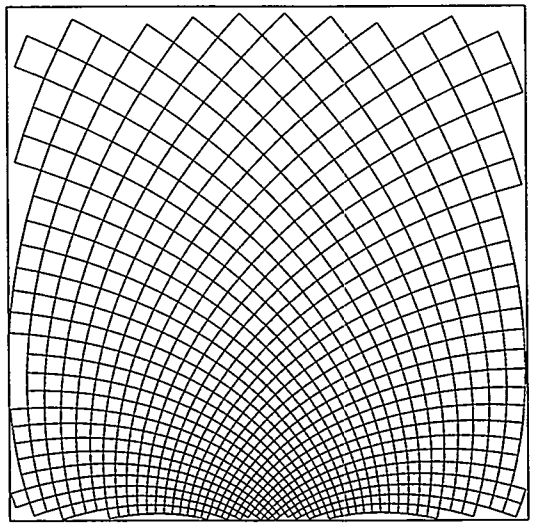


Figure 7: Mesh II for Example 1.

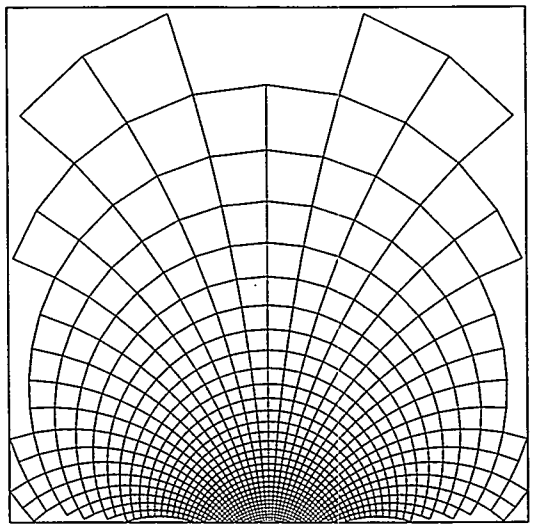


Figure 8: Mesh I for Example 2.

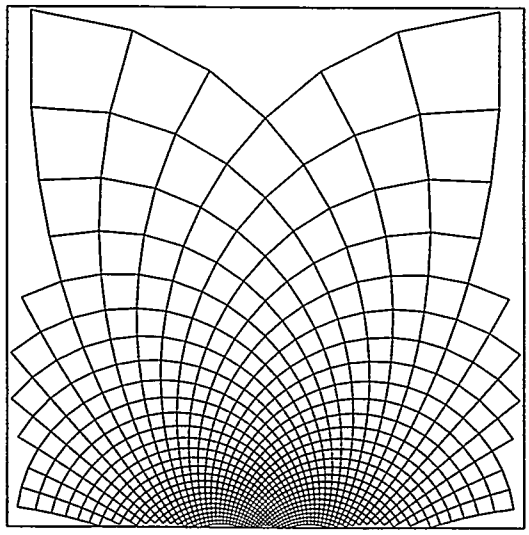


Figure 9: Mesh II for Example 2.

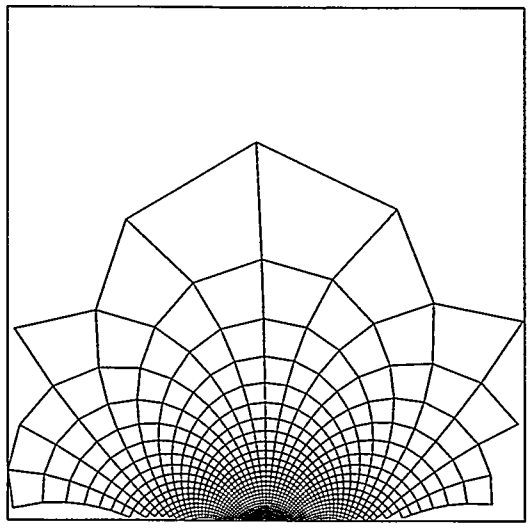


Figure 10: Mesh I for Example 3.

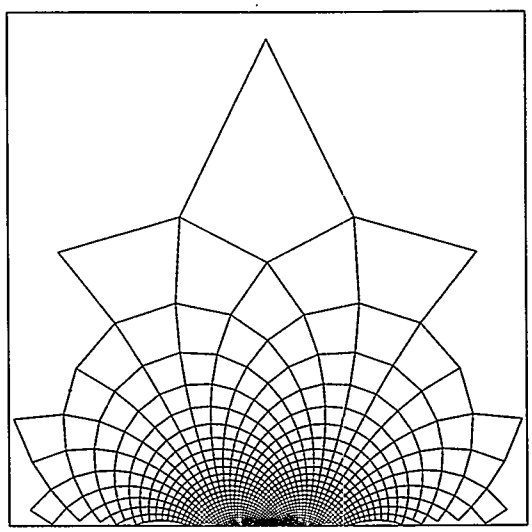


Figure 11: Mesh II for Example 3.

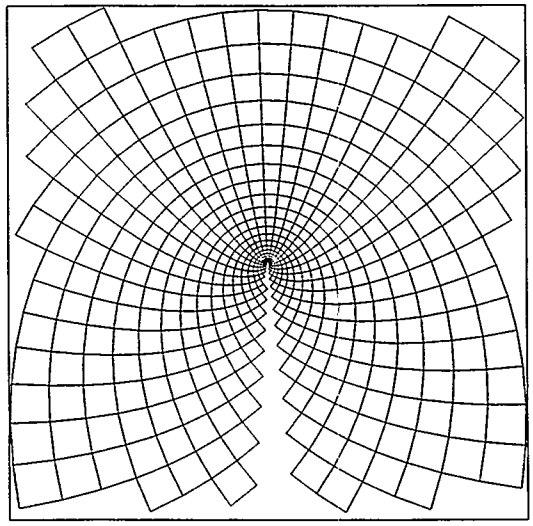


Figure 12: Mesh I for Example 4.

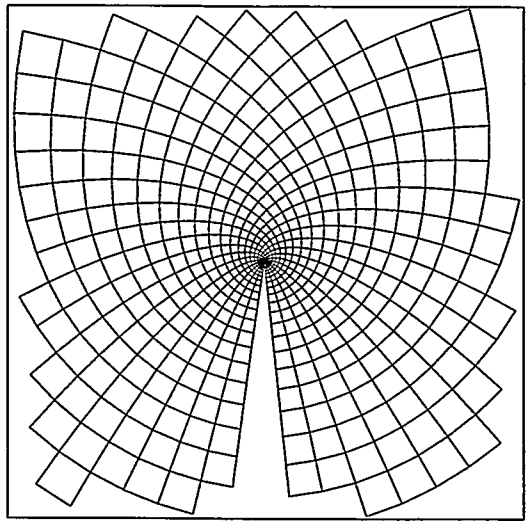


Figure 13: Mesh II for Example 4.

Appendix A

In this section, we show the error function for quadratic interpolation over a parallelogram is given by (6) using only simple algebraic expansion. Let the data function be

$$f(x,y) = \frac{1}{2}[x,y]H[x,y]^t + [g_1,g_2][x,y]^t + c \tag{14}$$

and the affine isoparametric transformation be

$$\begin{bmatrix} x(p,q) \\ y(p,q) \end{bmatrix} = T\begin{bmatrix} p \\ q \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \; T = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} = \begin{bmatrix} x_2 - x_1 & x_4 - x_1 \\ y_2 - y_1 & y_4 - y_1 \end{bmatrix}. \tag{15}$$

Then the interpolation error can be shown to be

$$\begin{aligned} E_Q(p,q) &= p_b(x(p,q),y(p,q)) - f(x(p,q),y(p,q)) \\ &= \mathcal{E}_Q - \frac{1}{2}\left(\mu_1(p-p_c)^2 + \mu_2(q-q_c)^2\right) , \end{aligned} \tag{16}$$

with centroid at $[p_c,q_c] = [\frac{1}{2},\frac{1}{2}]$,

$$\begin{aligned} \mathcal{E}_Q &= E_Q(p_c,q_c) = \frac{1}{8}(\mu_1+\mu_2) , \\ \mu_1 &= [u_x,u_y]H[u_x,u_y]^t, \quad \mu_2 = [v_x,v_y]H[v_x,v_y]^t \end{aligned}$$

Let the data function over (p,q)-space be written as

$$\begin{aligned} \tilde{f}(p,q) &= f(x(p,q),y(p,q)) \\ &= \frac{1}{2}[p,q]\tilde{H}[p,q]^t + [\tilde{g}_1,\tilde{g}_2][p,q]^t + \tilde{c} \end{aligned}$$

$$\text{where } \tilde{H} = T^tHT = \begin{bmatrix} \tilde{h}_{11} & \tilde{h}_{12} \\ \tilde{h}_{12} & \tilde{h}_{22} \end{bmatrix} \quad \text{and} \tag{17}$$

$$[\tilde{g}_1,\tilde{g}_2] = \left([g_1,g_2] + [x_1,y_1]H\right)T , \tag{18}$$

$$\tilde{c} = c + [g_1,g_2][x_1,y_1]^t + \frac{1}{2}[x_1,y_1]H[x_1,y_1]^t .$$

The function values at the four interpolating corners are

$$\begin{aligned} f_1 &= \tilde{f}(0,0) = \tilde{c} , \quad f_3 = \tilde{f}(1,1) = \frac{1}{2}(\tilde{h}_{11}+\tilde{h}_{22}+2\tilde{h}_{12}) + \tilde{g}_1 + \tilde{g}_2 + \tilde{c} , \\ f_2 &= \tilde{f}(1,0) = \frac{1}{2}\tilde{h}_{11} + \tilde{g}_1 + \tilde{c} , \quad f_4 = \tilde{f}(0,1) = \frac{1}{2}\tilde{h}_{22} + \tilde{g}_2 + \tilde{c} . \end{aligned} \tag{19}$$

By (5) and (16) (note the vanishing of linear and constant terms),

$$\begin{aligned} E_Q(p,q) &= \left(\sum_{i=1}^{i=4} f_i\phi_i(p,q)\right) - \tilde{f}(p,q) \\ &= \frac{1}{2}\left(p(1-q)\tilde{h}_{11} + pq(\tilde{h}_{11}+\tilde{h}_{22}+2\tilde{h}_{12})\right. \\ &\qquad \left. +(1-p)q\tilde{h}_{22} - (p^2\tilde{h}_{11} + q^2\tilde{h}_{22} + 2pq\tilde{h}_{12})\right) \\ &= \frac{1}{2}\left(p\tilde{h}_{11} + q\tilde{h}_{22} + 2pq\tilde{h}_{12} - p^2\tilde{h}_{11} - q^2\tilde{h}_{22} - 2pq\tilde{h}_{12}\right) \\ &= \frac{1}{2}\left(p(1-p)\tilde{h}_{11} + q(1-q)\tilde{h}_{22}\right) \\ &= \frac{1}{8}(\tilde{h}_{11}+\tilde{h}_{22}) - \frac{1}{2}(\tilde{h}_{11}(p-\frac{1}{2})^2 + \tilde{h}_{22}(q-\frac{1}{2})^2) . \end{aligned} \tag{20}$$

From (15) and (17), we have $\tilde{h}_{11} = \mu_1$ and $\tilde{h}_{22} = \mu_2$; hence the error function has the form given in (16).

References

[1] Houman Borouchaki, Paul Louis George, Frederic Hecht, Patrick Laug, and Eric Saltel. Delaunay mesh generation governed by metric specifications. Part I. Algorithms. *Finite Elements in Analysis and Design*, 25:61–83, 1997.

[2] E. F. D'Azevedo. Optimal triangular mesh generation by coordinate transformation. *SIAM J. Sci. Statist. Comput.*, 12(4):755–786, 1991.

[3] E. F. D'Azevedo and R. B. Simpson. On optimal interpolation incidences. *SIAM J. Sci. Statist. Comput.*, 10:1063–1075, 1989.

[4] E. F. D'Azevedo and R. B. Simpson. On optimal triangular meshes for minimizing the gradient error. *Numer. Math.*, 59:321–348, 1991.

[5] A. R. Mitchell and R. Wait. *The Finite Element Method in Partial Differential Equations.* Wiley-Interscience Publication, 1977.

[6] E. Nadler. Piecewise linear best l_2 approximation on triangulations. In C. K. Chui, L. L. Schumaker, and J. D. Ward, editors, *Approximation Theory V*, pages 499–502, Boston, 1986. Academic Press.

[7] J. Peraire, M. Vahdati, K Morgan, and O. C. Zienkiewics. Adaptive remeshing for compressible flow computations. *J. Comput. Phys.*, 72:449–466, 1987.

[8] Kenji Shimada. Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles. In *Proceedings 6th International Meshing Roundtable 1997, October 1997, Park City, Utah*, 1997. Also available as Sandia Report SAND 97-2399 UC-405.

[9] R. B. Simpson. Anisotropic mesh transformations and optimal error control. *Applied Numerical Mathematics*, 1992. Speical issue as the proceedings of the US Army sponsored Workshop for Adaptive Methods for Partial Differential Equations, Rensselaer Polytechnical Institute (accepted).

[10] I. S. Sokolnikoff. *Tensor Analysis, Theory and Applications to Geometry and Mechanics of Continua.* John Wiley, New York, second edition, 1964.

Solving Difficult Grid Related Problems Utilizing the Volume Grid Manipulator

Stephen J. Alter*
Lockheed Martin Engineering & Sciences
Hampton, Virginia 23681
e-mail: s.j.alterlarc.nasa.gov

Abstract. *This article presents a set of methods to solve difficult problems in grid generation as related to Computational Fluid Dynamics (CFD). Throughout the evolution of a CFD simulation, modification of existing volume grids may be required to improve flow domain capture, resolve characteristic gradients, and enhance the accuracy of computed aerodynamic and thermodynamic parameters. The processes identified in this paper make extensive use of the Volume Grid Manipulator (VGM) multi-purpose grid manipulation code which contains an interpretor style language. The VGM software enables the build up of simple processes, or manipulations, to develop complex techniques that can be used to solve a wide variety of grid related problems. The VGM code also provides a user friendly environment for the solution of difficult grid related problems. Coupling the rich command language with the journaling or script generation capability afforded by the VGM code, enables the use of complex solution techniques to solve a variety of grid problems as well as provide quick turnaround. To demonstrate the powerful manipulative capabilities of the VGM code, a volume grid of a simple shape with multiple flow features will be adapted, a wake volume grid will be appended to a volume grid of an X33 Venturestar, and the volume grid of another X33 Venturestar will be expanded to ensure flow capture while removing negative volume cells.*

keywords. topology, structured grid-generation, parametric studies, grid-adaption

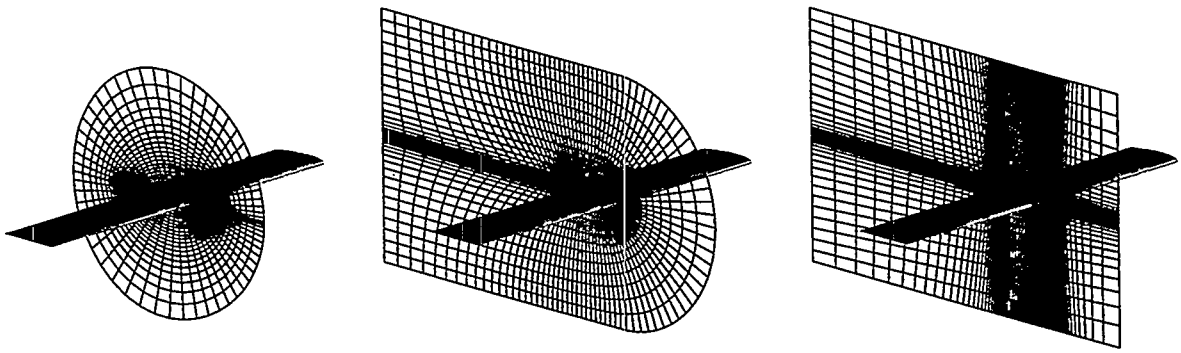
1 Nomenclature

I,ξ	streamwise computational direction measured from nose to tail of body
J,η	circumferential computational direction measured from top to bottom of body
K,ζ	computational direction normal to body surface
R	gas constant
ρ	density
$\frac{(S_2-S_1)}{R}$	entropy change between points
ΔS_i	distance between points (i,j,k) and $(i-1,j,k)$
ΔS_j	distance between points (i,j,k) and $(i,j-1,k)$
ΔS_k	distance between points (i,j,k) and $(i,j,k-1)$
X,Y,Z	Cartesian coordinates

2 Introduction

Iterative schemes have been used to solve fluid problems for a multitude of applications for nearly a century. Historically, complex fluid flow equations were simplified to algebraic and ordinary differential representations to simplify the analysis. As computer technology has progressed, the simplifications of the fluid flow equations have been reduced leading to the solution of partial differential equations. For most of the applications of the fluid flow equation models, structured grids have been employed. Structured grids are so named because there is an ordering to the points that discretize the geometry and volume on which the computation is performed. Three-dimensional surfaces can be represented in two computational coordinates, and volumes are represented in three-computational coordinates. The ordering of the points for surfaces and volumes is done such that at each computational coordinate there is an equal number of points in the second coordinate. For example, in a surface grid of 15 by 35 points, at each of the 15 points there are 35 points that locate specific physical coordinate points on the surface. Structured grids in the computational domain take on the appearance of square cells for surfaces and cubes for volumes because there is always one computational unit between points. The ordering of the grid points that define a configuration in a

*Senior Aeronautical Engineer, Senior Member AIAA



(a) O-grid (b) C-grid (c) H-grid

Figure 1: Structured grid topologies for a wing.

structured grid provide easy application of boundary conditions as well as computation of derivatives that comprise the various fluid flow equations.

The ordering of the points that discretize and define a surface or volume grid is called topology. The topology of a structured grid provides the ordering of defining grid points or domain discretizing points such that the pertinent geometrical features in the computation are captured. For example, an O-grid topology, Fig. 1a, is typically used to model a three-dimensional wing, a C-grid topology, Fig. 1b, could be used on the same wing attached to a fuselage, and an H-grid topology, Fig. 1c, could for the same purposes.

Until recently, most configurations and geometries analyzed with structured grids have been simple in shape and form, requiring only a single block. As the geometries being analyzed are increasingly complex, the use of multiple simple blocks to define the discretized volume of the modeled fluid flow has become common. Multiple block decompositions of a flow domain provide a simple solution to the most difficult obstacle in structured grid development, topology. By breaking the domain into simpler segments (blocks), the development of the ordering (topology), is manageable. The blocks can be overlapped where there are regions of grid points in one block that occupy the same region of another block, abutted where the gird points match point-to-point at their interfaces, or patched where the point-to-point matching is a subset of one grid to another, i.e., a 3 to 2 point matching. Additionally, structured grids of any topology can be tailored for Euler or Navier-Stokes computations.

Structured grids used for solving Euler or Navier-Stokes flow models have several common features. These include the necessity to capture gradients that significantly affect surface properties, capture of the affected flow domain as it reacts to the geometry being analyzed, and the highest fidelity possible. High fidelity of structured grids is characterized by point-to-point spacings changing at a rate close to unity, skewness is kept to a minimum throughout the domain, and orthogonal grid lines traversing from the wall of the configuration to the outer domain limits. Although structured grid generation technology is mature, there are a number of challenges that need solving, to improve the productivity in using such grids. These most common challenges include reductions in the resources of computers and labor to do:

- grid adaption that produces high fidelity grids that capture those flow features that significantly affect the accuracy of the computation;
- wake generation for existing flow domains where separation can play a key role in accuracy of the computation; and
- proper flow domain capture.

All flows require grid adaption to some extent for the resolution of flow variable gradients. The adaption allows for the efficient placement of grid points in the volume grid by removing resolution of the flow domain where gradients are non-existent or benign as compared to other regions. Structured volume grid adaption in three dimensions is a difficult task. Anisotropic adaption, which moves grid points with respect to the flow gradient, usually requires laborious iterative techniques to properly capture strong flow field gradients while retaining grid quality throughout the domain[1]. A simpler approach is to utilize isotropic grid adaption which moves points along a grid line, but capture of flow gradients can be difficult as there is no dependency on the adjacent grid lines[2]. Further complications arise in the selection of the proper flow variables to be used in computing the flow gradients to be resolved. Use

of multiple flow variables can be difficult to incorporate into a single grid adaption. In any case, maintaining grid quality while adapting to flow field gradients is nearly impossible in a batch operation because of flow variable choices and the use of interactive operations is extremely time consuming.

Analysis of the wake region of vehicles moving through a fluid, in both high and low speed flows is important with respect to aerodynamic computations. Development of appropriate grids for wake regions for hypersonic vehicles is difficult due to engine nozzles, control surfaces and servicing apparatus located in the base region. Hence multiple block decompositions are generally employed. The difficulty in constructing the multiple blocks with adjacent zones is compounded by the need to have grid line slope, cell size, and point-to-point stretching continuity at the interfaces. Development of the grid point distributions usually requires the extraction of grid point spacings at a boundary and the spacing gradient approaching the boundary in all computational directions. Combining these parameters with a requirement of slope continuity at the boundaries, makes the development of high fidelity wake volume grids next to impossible without the use of software to rapidly compute these quantities.

A common obstacle encountered during the evolution of the Computational Fluid Dynamic (CFD) simulation, is the construction of an adequate flow domain for the affected region of fluid surrounding a vehicle. This type of volume grid may arise from adapting the outer boundary with internal transients that result in a false position of the outer reaches of flow effects, or the initial improper sizing of the flow domain. Many techniques exist to expand or contract a volume grid to ensure flow capture, including localized coordinate expansions[3] and grid line extrapolation, to name a few. Use of these two methods can produce a volume grid that adequately captures the flow domain, but they can cause grid lines to cross resulting in the formation of negative cell volumes. However, the two expansion techniques mentioned can produce regions of volume grids that are usable, but combining the usable regions is difficult without writing specific C or Fortran code to perform the merging operation.

This paper will discuss the methods and tools that can be used to solve the three major challenges facing grid generation previously identified. The methods make extensive use of the Volume Grid Manipulator[4] (VGM) multi-purpose grid manipulation code because it offers a wide variety of capabilities and functionality required to solve these difficult problems. Though there may be other challenges to be solved in the structured grid generation arena, the three listed above address the most common encountered obstacles. Solution of these will provide improved productivity and applicability of structured grids to CFD simulations.

3 The Volume Grid Manipulator

All of the complex and difficult issues mentioned in the introduction can be easily solved through the use of the VGM tool. The VGM code has a language structure with 12 commands, listed in table 1, that offer the necessary operations and methods that can be used to perform grid adaption, generate and append wakes onto existing volume grids, and provide adequate flow capture for CFD simulations.

Command	Description
allocate	Create a new block or array variable
blend	Interpolate between existing points in a variable
combine	Regroup volume grids into a single grid system
copydist	Copy a distribution from one grid line to another
for	Looping for a repetitive complex manipulation
quit	End execution
read	Input data
redist	Redistribute a grid line based on a function
set	Equate variables and data or compute data
smooth	Smooth a grid with algebraic or PDE solvers
tfi	Perform Trans-Finite Interpolation on a region/zone
write	Output data

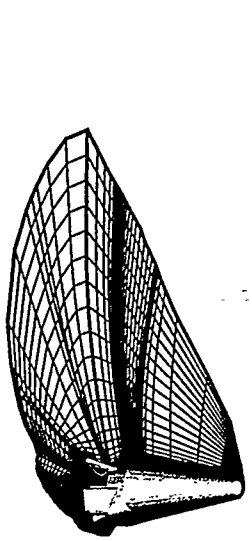
Table 1: VGM command summary.

Although there are only a dozen commands, each command has several options such as physical versus computational domain based parameters, distribution functions, computational direction to be used, and arclength versus normalized arclength interpolants, to name a few. By changing one option on each command, over 1000 specific manipulation instructions result. This command rich language embodies the capability to build up various commands to perform

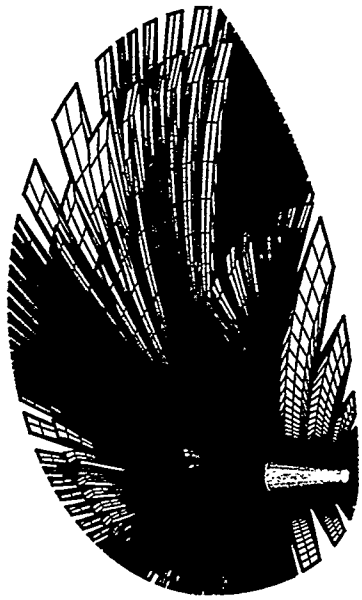
fast and powerful manipulations of grid points, lines, surfaces and volumes (i.e., grid data). While most commands are designed to operate on existing grid data, some manipulations can be used to generate grid points, lines, surfaces and volumes. Examples of the simple manipulations VGM was designed to perform include single to multiple block decompositions and vice versa, insertion of parametric design changes into existing grids, and the conversion of grid data between PLOT3D[5], GRIDGEN[6], TECPLOT[7], and LAURA[2] data sets. Although the VGM code currently supports the input and output of LAURA restart files, extension to other commonly used flow solvers like GASP[8], and TLNS3D[9] is straightforward and can be easily implemented.

Examples of the simple manipulations VGM was designed to perform are coarse grain to fine grain grid adaption, inviscid to viscous grid conversion and topological changes, illustrated in Figs. 2, 3a, and 4, respectively. Notice that the inviscid to viscous grid conversion usually improves grid quality which can be seen by the thinner grid lines in Fig. 3b, where the largest change in grid line character occurs. The grid quality is improved by virtue of the increased orthogonality in the body to outer domain direction. This improvement is provided by a combination of a spline with local corrections to prevent severe overshooting and re-parameterization of the resulting grid line in the event of overlapping points. Typically VGM either retains the existing quality of the grid, or it improves the quality through reduced skewness, enhanced orthogonality, reduced point-to-point stretchings and slope continuity. The illustrated alterations appear to be complex, but they are all accomplished using less than six manipulator command lines. The VGM script used for the most complex of these manipulations, the coarse to fine grain grid adaption, is shown in Fig. 5.

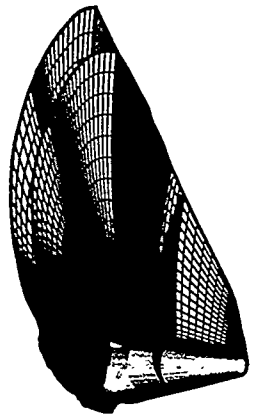
The VGM code is a powerful, multi-purpose grid manipulation tool that allows the use of the VGM language interactively or in a batch mode of operation using a script, thereby making the code applicable towards parametric design studies. The compactness of the VGM language counteracts the lack of a Graphical User Interface for the code. The VGM language scripts, like the one shown in Fig. 5, use easy to read layout for each command, where there is no order to the command arguments for most commands. Ordering of command arguments becomes important when specifying source and destination grids to be manipulated. Unlike Computer Aided Design (CAD) systems, and most grid generation systems, the VGM code also has the unique feature of being able to compute existing grid spacing, spacing gradients and slope continuity angles for any existing grid. This adds to the power of relational grid generation, a key to developing high fidelity surface and volume grids. Some flow regimes require more refinement than others, however, the techniques discussed in this article will be applicable to all fluid speeds. Techniques for solving the three difficult grid related problems mentioned above will be addressed, by adapting a volume grid for a simple sphere-cone-cylinder-flare configuration with an appended wake using a quasi-isotropic scheme, constructing a wake for an existing solution converged X33 Venturestar forebody, and expanding the flow domain of a similar X33 configuration to ensure flow capture. Although these capabilities are not unique to the VGM code, this is the only code that embodies all the necessary tools and operations to perform these complex manipulations.



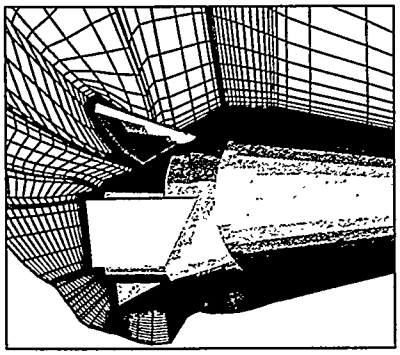
(a) Flow adapted coarse grid



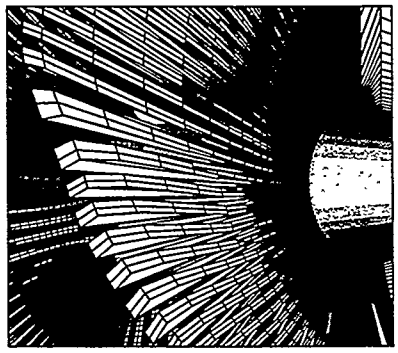
(b) Coarse grid insertion



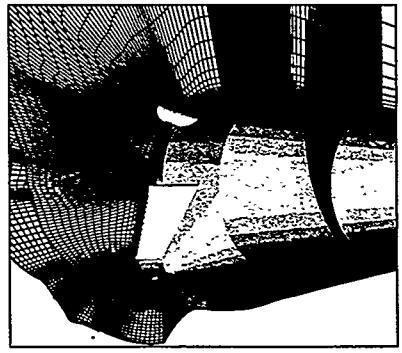
(c) Adapted fine grid



(d) Enlarged view of aft end

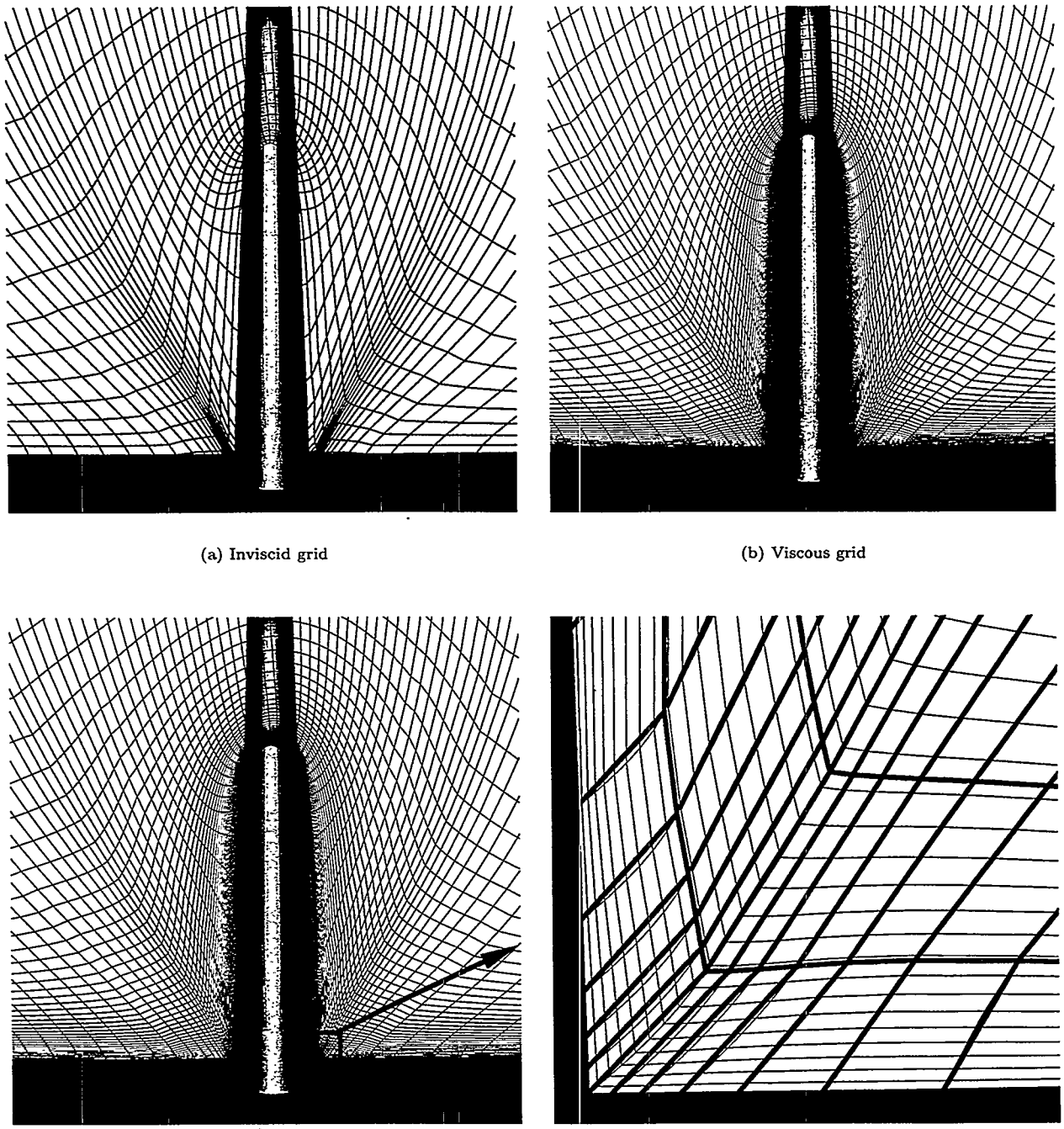


(e) Enlarged view of middle section



(f) Enlarged view of adapted fine grid

Figure 2: Coarse to fine grain grid adaption of a proposed X33 concept.



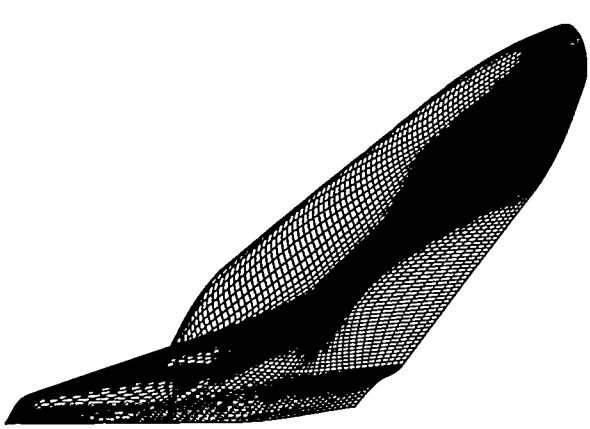
(a) Inviscid grid

(b) Viscous grid

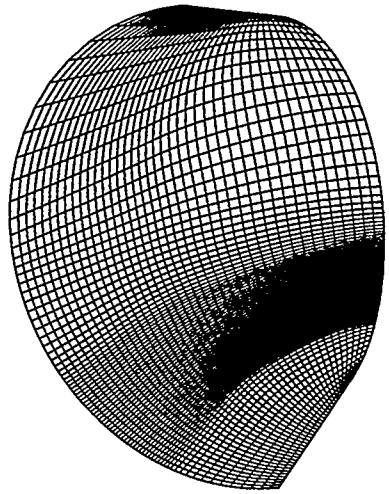
(c) Local region

(d) Orthogonality improvement

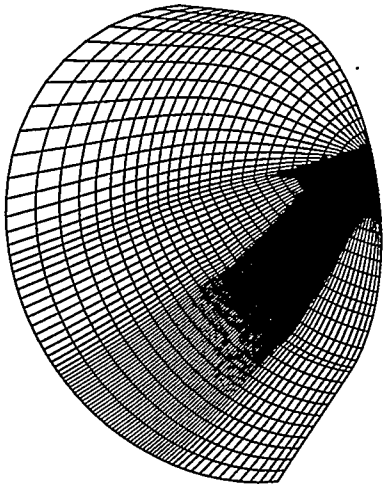
Figure 3: Inviscid to viscous grid conversion on an X33.



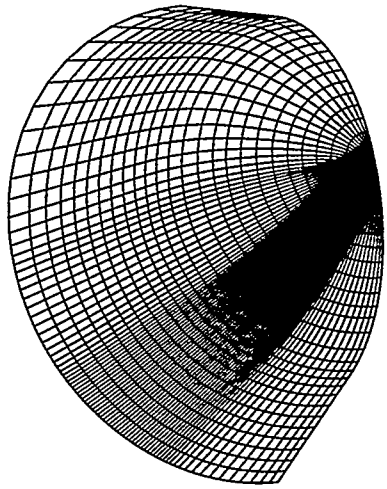
(a) Original surface grid



(b) Original topology on the nose



(c) New topology on the nose



(d) Final smoothed grid

Figure 4: Conversion of a parabolic singularity to a pole boundary on a finless Reusable Launch Vehicle.

```
# Adapt a fine grid of a candidate X33 based on a coarse grid
# solution. First, read in the coarse then fine grids.

read X33_mod-2.g plot3d unformatted single
read X33_mod-0.g plot3d unformatted single

# Generate the distribution function for the entire domain by
# inserting the coarse grid and interpolating to get the fine
# grid distribution functions. First, memory is allocated to
# store the distribution function to be built.

allocate dsk1[153,121,65]

# The current arclength distribution function from the coarse
# grid, which has dimensions of (20 X 16 X 65), in the body to
# outer domain direction is extracted:

set dsk1[1-0:8,1-0:8,1-65] = dska(xyz[1,1,1-0:1,1-0:1,1-65])

# Holding the coarse grid distribution fixed at intervals of
# 8 points in the streamwise and cross-sectional directions,
# interpolate with a spline function to compute the distribution
# function between these fixed K-direction grid lines.

blend dsk1[:8,:8] 1d i-direction parametric interpolation=spline
blend dsk1[:8,:8] 1d j-direction parametric interpolation=spline

# Holding the topologically rectangular volumes separated by
# intervals of 8 points in the I- and J-direction, fill in the
# volume of the distribution function by interpolating with
# Trans-Finite interpolation in 2D (I,J), and marching in the
# K-direction.

blend dsk1[:8,:8] 2d k-direction physical interpolation=tfi \
      xyz[1,1,:8,:8]

# Adapt the fine grid with the inserted and interpolated coarse
# grid distributions.

redist k-direction spline normarc physical points=65 func(dsk1) \
      xyz[2,1] newblock=no

# Output the resulting grid:

write X33_mod-3.g plot3d single xyz[2]
quit
```

Figure 5: Coarse to fine grain grid adaption procedure using VGM.

4 Volume Grid Adaption

One of the most important aspects of performing CFD simulations is the proper use of available computer resources and labor. All flow domains are similar in one respect, there is a region of fluid around the body being analyzed that is affected by the fluid. A wise use of computer resources dictates that this region be the only region in which computations are performed. A priori knowledge of the size and location of the region of fluid affected by a geometry cannot be accurately determined with engineering tools. The size and location of the flow domain is dependent on a fluid model such as equilibrium, non-equilibrium, and chemically-reacting flow as well as the flow conditions such as Mach number and angle of attack. Typically, approximations in fluid models are used to determine the limits of the flow domain. This results in portions of the flow domain that are not affected by the geometry. These regions of no flow activity can be eliminated by moving the discretized domain to capture only the flow domain. Further, during the alteration process, otherwise known as grid adaption, the discretized domain is altered to capture those regions containing strong flow gradients. This procedure significantly increases the efficient use of computer resources.

Grid adaption in two dimensions is simple in comparison to three dimensions. There are as many grid adaption schemes as there are codes to do grid adaption, but most of them use anisotropic movements in the grid to adapt to flow gradients as well as restricting the flow domain to the affected region of fluid[1, 10, 11]. Anisotropic grid movements are the shifting of grid points with respect to the flow gradient, while isotropic grid movement is with respect to the grid line on which the points lie. Anisotropic grid adaption typically destroys grid quality by producing highly skewed cells and point-to-point stretchings that may not be conducive to obtaining solution convergence with the CFD solver being used. Isotropic grid adaption retains the overall grid character but moves points along existing grid lines to capture flow field gradients. Although skewness may be produced, the point-to-point stretchings and slope continuity are retained. Isotropic grid adaption is preferred because two measures of grid quality are preserved while providing adequate resolution of flow field gradients.

The VGM code enables the use of isotropic grid adaption through the manipulation of existing grid lines to capture flow gradients. The process of adapting a volume grid with VGM is to first determine the control points to remain fixed. This is done for one computational direction on a very coarse representation of the flow domain or a coarse grid both on which the flow has been computed and is appropriately captured. These control points are typically the positions of extrema in a flow variable. For example, in the simple sphere-cylinder-cone geometry illustrated in Fig. 6, the grid is comprised of seven blocks, but each set of blocks traversing the body to the outer domain direction can be combined to produce three separate zones. This reduction is performed to appropriately adapt the grid in the direction from the body to the outer boundary (i.e., the k-direction) and improve slope continuity in the other computational directions. The first zone contains blocks 1 and 2, the second zone contains blocks 5 and 3 and the third zone contains blocks 7, 6, and 4.

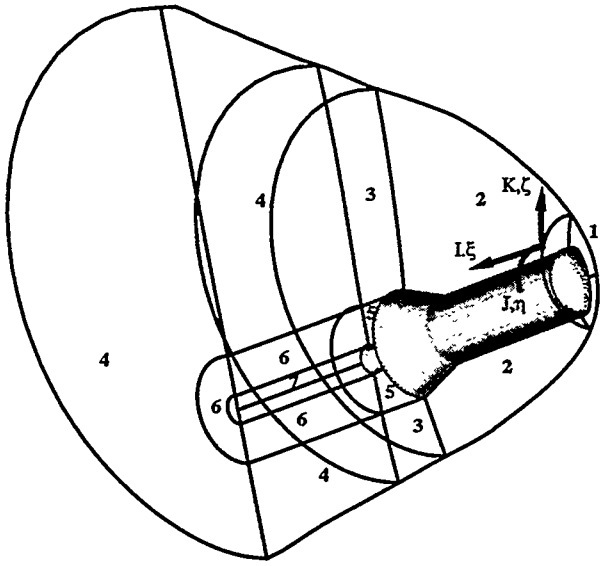


Figure 6: Simple sphere-cylinder-cone geometry with a nozzle embedded in the wake domain and comprised of 7 blocks.

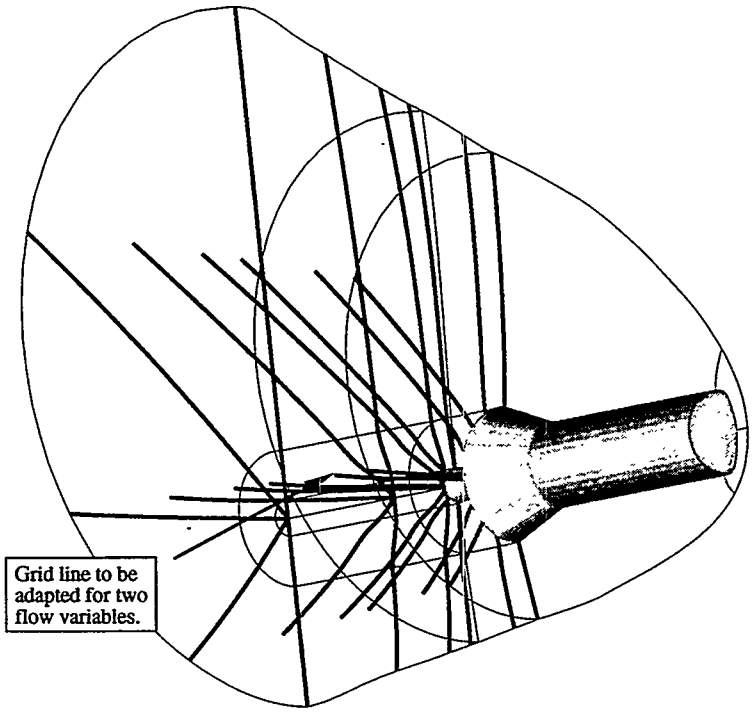


Figure 7: Lines on which grid adaption is based.

For this example, only the wake is adapted as this region is the most complex. Coarsening the grid in the streamwise and cross-sectional directions results in a set of lines in Fig. 7. It is these lines that are evaluated for flow variable gradients, for which control points are computed. The control points will be the fixed points along each of the grid lines.

The control points can be obtained by considering any number of flow variables, with each giving a new set of points to remain fixed, as shown for an interior grid line in the body to outer domain direction in Fig. 8. By using several flow variables, the domain is adapted based on the pertinent flow characteristics without sacrificing the gradients of one variable in favor of another. For the flow domain, the entropy and magnitude of local velocity contours are shown in Fig. 9. Although there are similarities between each variable in the location of the gradients, some matching regions do not overlap, or do not have as strong a gradient between the two chosen variables. Using both significantly improves the quality of the solution through resolution of these various gradients. Additionally, focusing the available grid points on these gradients increases the efficient use of computer resources.

Each of the identified control points along the respective grid line to be adapted serves as end points for redistributing the grid line. The curvilinear segment between each control point will have a new set of points that clusters cells to the ends of the segment, thereby adapting the grid to the control points. Each grid line is redistributed in this manner, noting that the total number of grid points along the grid line does not change. By using the same number of points between the control points on each segment, and by utilizing the block referencing data structure of the VGM code, reduces the labor required for redistribution. Labor is reduced because of the commonality between the segments in point distribution functions and dimension, despite differences in computational length. For the example grid line, the original and new grid point distributions are shown in Fig. 10.

These new grid lines now serve as the basis for the entire domain. The remaining grid lines in the domain, along the computational direction chosen for grid adaption are then adapted by interpolating between the known adapted grid lines. The interpolation can be done linearly, elliptically[12], using a spline, or any combination of these three. Typically the streamwise direction is done first, then the cross-sectional as the dependency is more an issue in controlling grid quality in the streamwise direction. The interpolations result in an adapted volume grid based on the two flow quantities chosen, and is shown in Fig. 11. Notice the overall grid quality remains consistent, the point-to-point stretchings do not exceed a growth rate of 50%, and slope continuity along the streamwise and cross-sectional grid lines is maintained. The growth rate is kept to a minimum by using distribution functions that inherently do not permit excessive point-to-point stretchings, such as those of Vinokur[13] and the LAURA grid adaption code. This

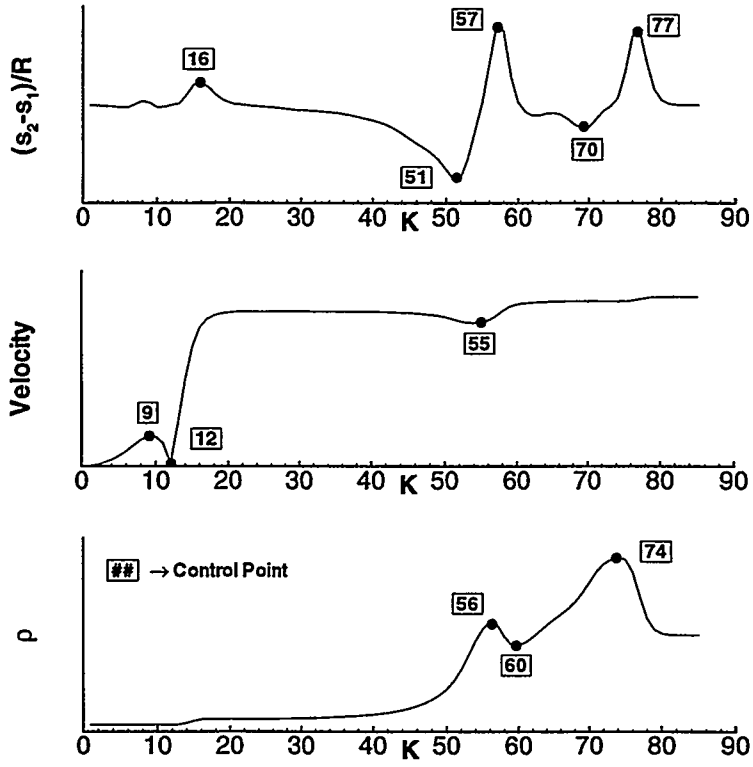
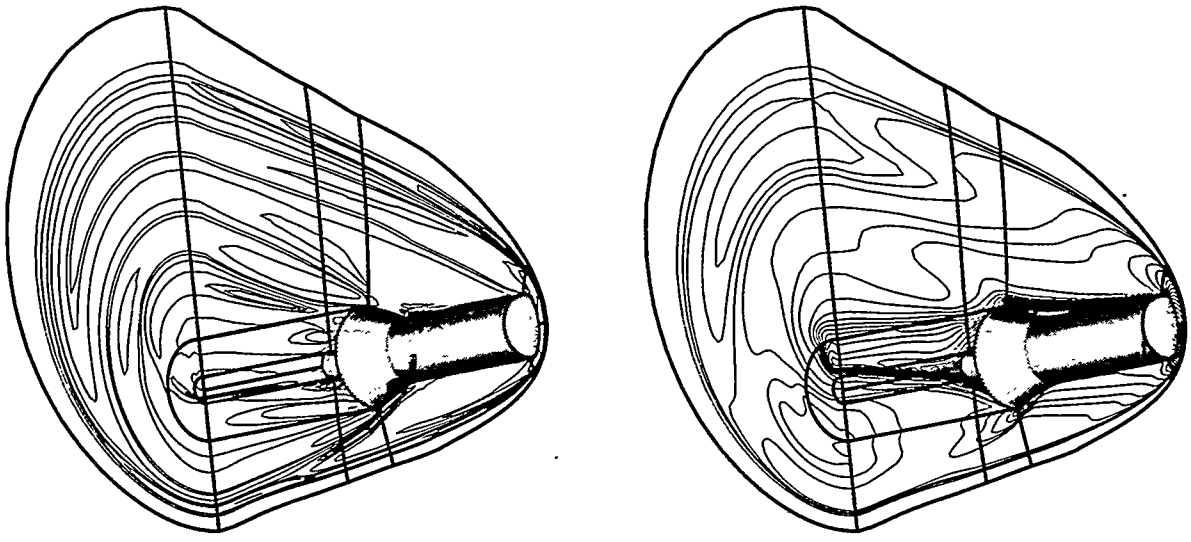


Figure 8: Control points along a grid line for entropy, velocity, and density along an interior grid line.



(a) Entropy contours

(b) Local velocity magnitude contours

Figure 9: Flow variable contours considered in the adaption process.

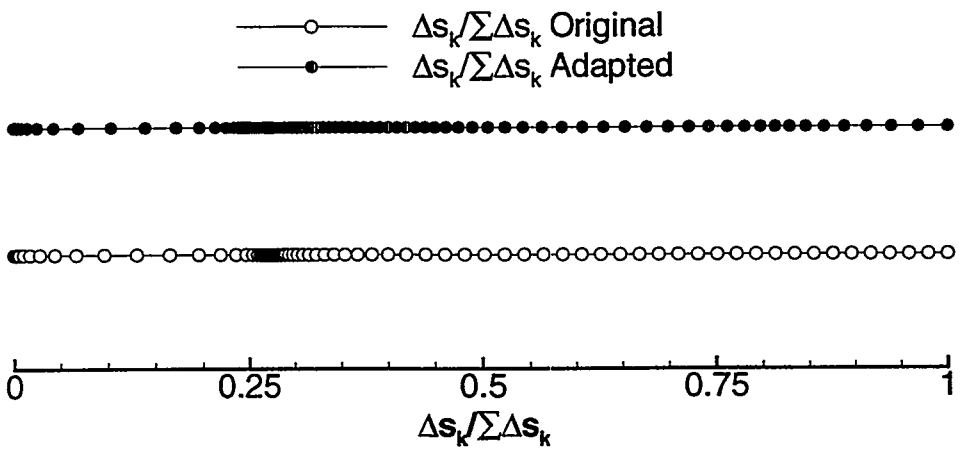
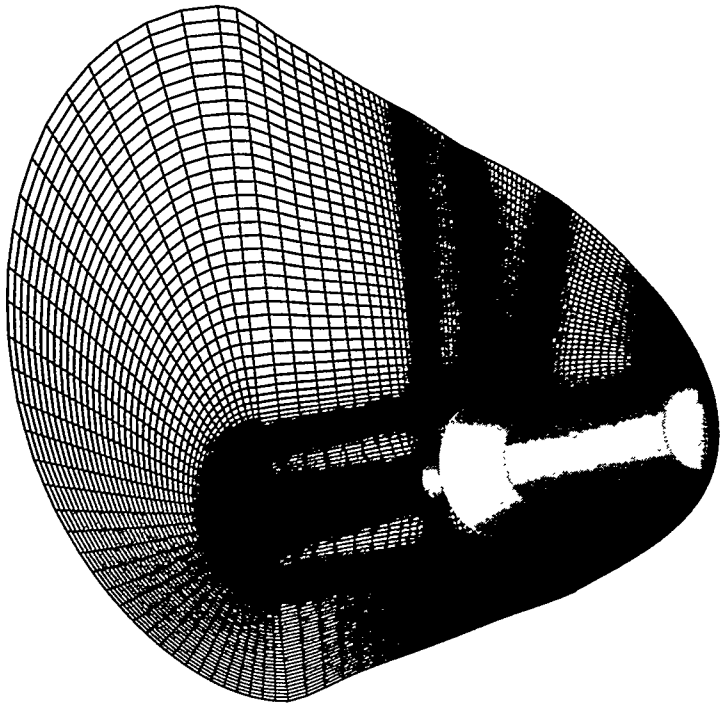


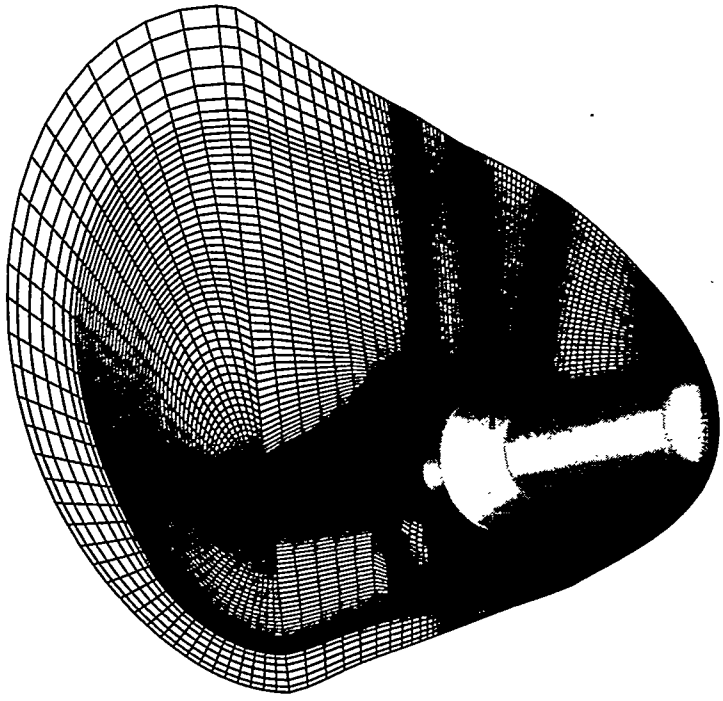
Figure 10: Adapted grid line for entropy and velocity gradients along an interior line.

volume grid is still considered to be of high fidelity, but now is appropriately distributed to capture the important flow gradients. Although only two flow variables are chosen for this adaption, any number of variables can be used. This process has been used on a variety of problems, each resulting in enhanced resolution of flow gradients[14, 15].

The described grid adaption process is simply accomplished by a build up of VGM commands, and is applicable to any volume grid and any flow regime. The process uses algebraic distribution functions and blending or interpolation algorithms. The use of algebraic manipulators significantly reduces the time required to adapt the grid, and their simplicity reduces the labor required to use them. Additionally, these manipulators are independent of flow regime, grid topology, and even grid composition be it overlapped or embedded, patched, or abutting.



(a) Original



(b) Adapted

Figure 11: Adapted wake domain grids using an isotropic scheme.

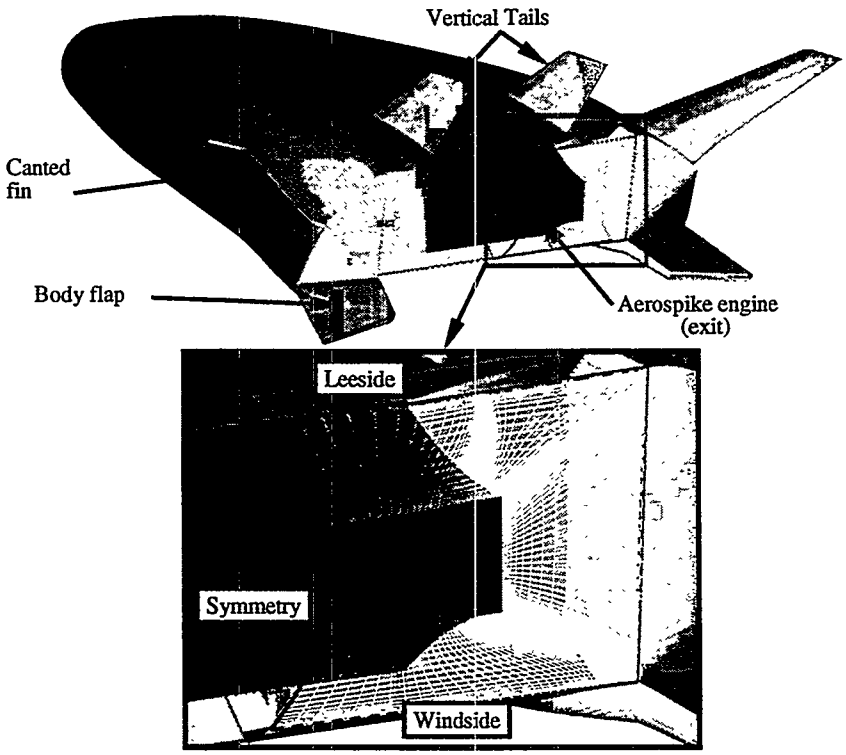


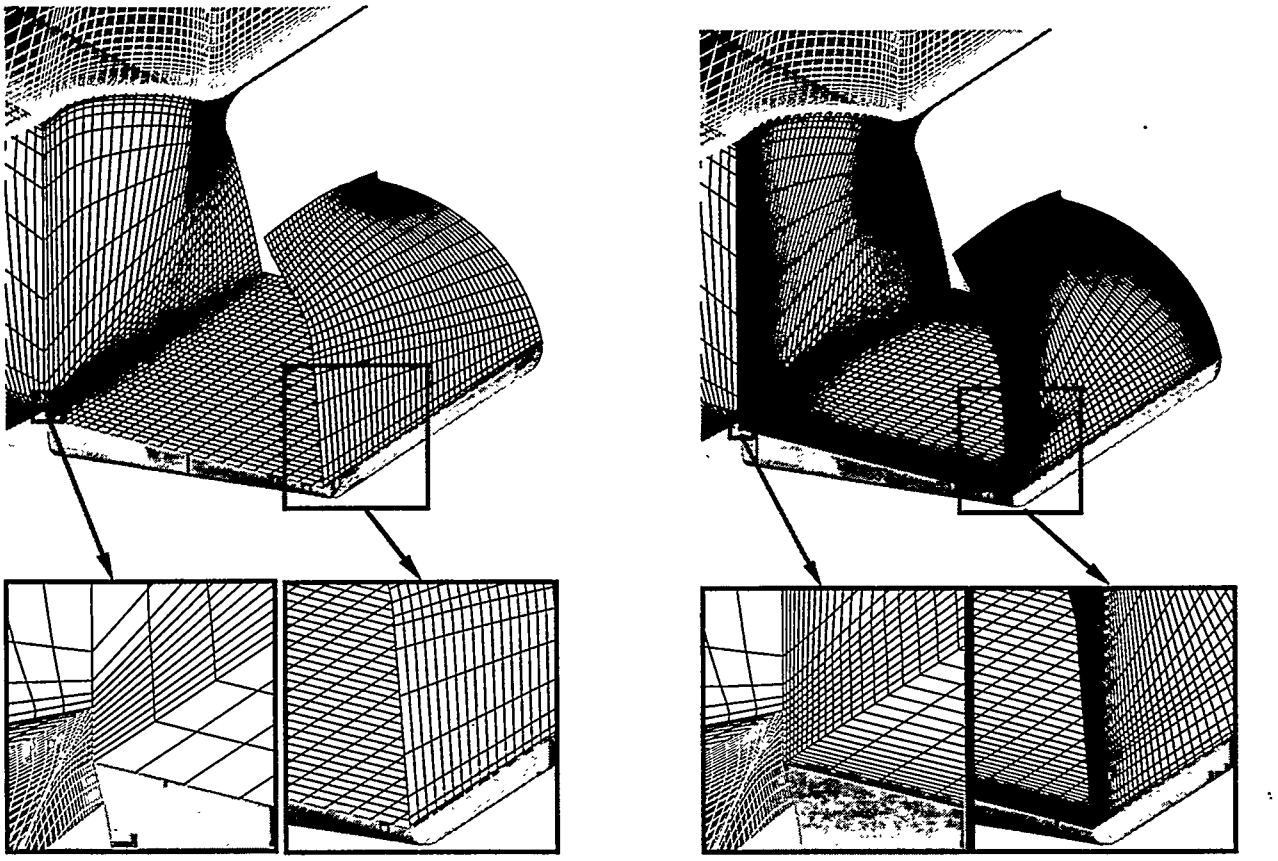
Figure 12: Interface of the O-grid engine block to the wake.

5 Wake Domain Grid Generation

During the evolution of a CFD simulation, a necessity may arise to model the wake region. Analysis of wake regions can be done to study the effects of applying the correct boundary conditions for exit surfaces, the affects of a sting mounting for wind tunnel configurations and the base region contributions to aerodynamics. For the case at hand, the contributions of aerodynamic forces and the thermal environment of the X33 Venturestar dictate the analysis of the wake domain. Generation of a wake volume grid for the solution-converged forebody grid of an X33 Venturestar commences by selecting the topology of the domain to be generated, and the cutoff distance for the wake. The cutoff distance was selected at twice the length of the body. The topology is a multiple block data set comprised of 15 blocks. Each block is generated in a specific order based on the availability of boundary data from newly generated adjacent blocks. Assuming the wall surface grids have been generated, the process of generating these blocks begins by constructing the engine block that encompasses the nozzle of the aerospike engine shown darker shaded in Fig. 12. The topology of this block is an O-grid that traverses from the leeside symmetry plane to the windside, as shown in Fig. 12. The connecting edges at the symmetry plane are generated by projecting the base wall grid at the symmetry planes to the end of the engine nozzle using straight lines. The remaining edges are then constructed as straight line connections at the end of the nozzle, with distributions copied from the opposing face edges. The surface that serves as an interface at the end of the engine nozzle does require the use of a 2D grid generator, but is the only surface that requires the additional computing. The next blocks that are generated are at the end of the vertical tail, and the end of the canted fin.

The remaining blocks that comprise the wake are constructed by generating the block above the bodyflap with straight lines from the base to the end of the flap, and splines on the top of the block. Subsequent blocks of the mid-engine, engine core, mid-tail, and mid-canted-fin are constructed in this order such that the interfaces to each consecutive block serve as known data for consistency in block-to-block matches. The remaining blocks in the wake core are generated identically to the mid section blocks. Grid point spacing continuity is enforced in all blocks by using the known data of previously generated block interfaces.

The final set of wake core blocks is developed iteratively with VGM until monotonicity in point spacing and proper grid clustering for resolving flow gradients about vehicle corners is achieved. Grid point monotonicity spacing can be



(a) Original grid

(b) Grid modified with a new function

Figure 13: Bodyflap block modified to improve grid effectiveness for boundary layer modeling.

obtained through the use of the 8 different distribution functions available in the redistribution command of VGM[16], or by building up a combination of the available distribution functions as well as copies of grid distributions within the available volumes. The build up is simply accomplished by multiplying the contributing distributions by a factor less than unity, such that the sum of the factors is unity. The results of this technique is shown in Figs. 13a and 13b. The initial distributions on top of the bodyflap were not conducive to viscous computations around the corners of the geometry nor the base of the geometry, as shown in Fig. 13. The distributions in the cross-sectional direction on top of the bodyflap, were improved by first reducing the cell sizes to 10% of the original values at the end points, and then doubling the number of points. This change was then propagated to the top of the base using parametric re-mapping[17], to retain the current distribution at the top of the base but account for the change in distribution on top of the bodyflap. The next direction modified was the top of the bodyflap to the top of the base in the body to outer boundary direction, at the end of the bodyflap. Initially this region did not have point-to-point stretchings that would properly capture a boundary layer. The distribution of points is improved by utilizing a copy of the current distribution in combination with a copy of the distribution at the opposing edge on the base. The results of these modifications in grid point spacings to improve boundary layer capture and modeling are shown in Fig. 13b. The result of the wake core grid generation is the set of blocks shown in Fig. 14.

The final block to be generated is the outer domain that uses the wake core interface, and the exit from the forebody domain, as starting surfaces. The domain is constructed by projecting upstream grid lines, or drive curves, on the forebody exit at the symmetry planes and a circular cross-section at the wake exit, as shown in Fig. 15. Drive curves represent the lines that control topology in a structured grid, because these lines "drive" the grid around the configuration. The dashed lines identify the projected lines used in the formation of the wake, while the dotted lines represent the straight line connections to the wake core. The large dots on the exit cross-section are the control points used in constructing the conic section of a circle for closing off the volume. The remaining undefined surface grids are generated with 2DTFI and GRIDGEN2D[6] at the exit, and the volume is generated with 3DTFI. The volume

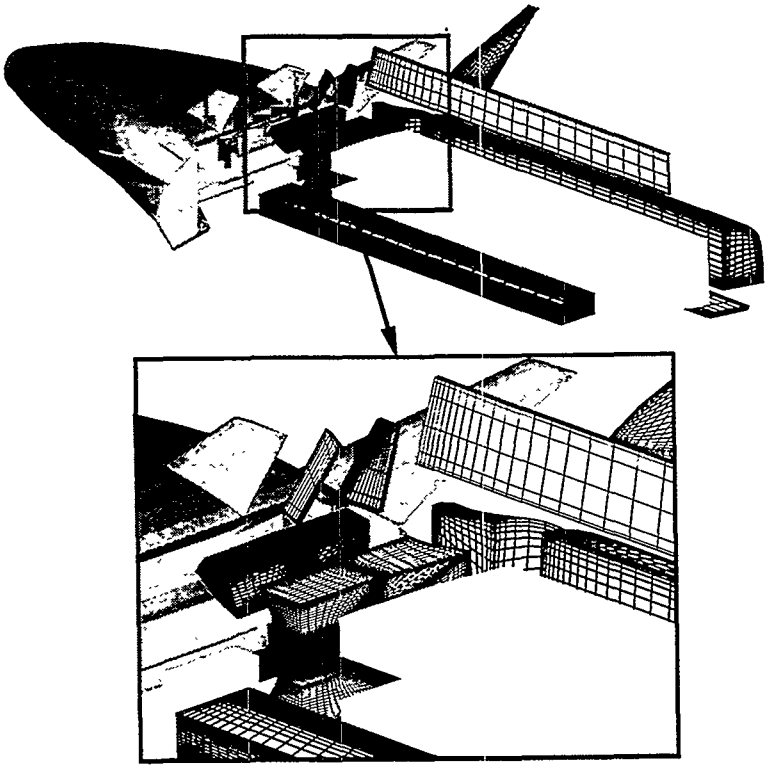


Figure 14: Expanded view of the X33 wake core blocks.

is then improved by running the 3DMAGGS[18] elliptic solver to smooth the grid and obtain orthogonal incidence angles at the interface to the wake core. Representative cross-sectional planes are shown in Fig. 16.

This process initially consumed 18 hours of wall clock time, but a subsequently generated wake took only 2.5 hours because the scripts were already generated, and parameter data from each block was used in the construction. A pseudo parametric grid generation capability is then afforded through the use of the block data as well as the scripts to do the generation. The generation of this domain discretization in a generalized grid generation tool required over 30 hours of wall clock time. Considering the fact that this tool does not have a scripting language, the time to generate each subsequent wake field domain would have required nearly the same amount of time. Therefore, use of the VGM code with its script language and the block data in the construction of the wake flow fields can significantly reduce the time to do parametric studies of wake field phenomenon.

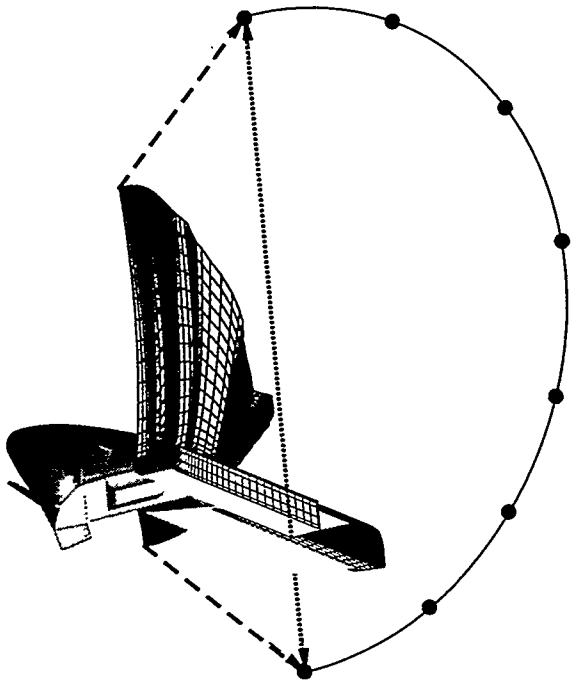


Figure 15: Projected drive curves used in outer block wake construction.

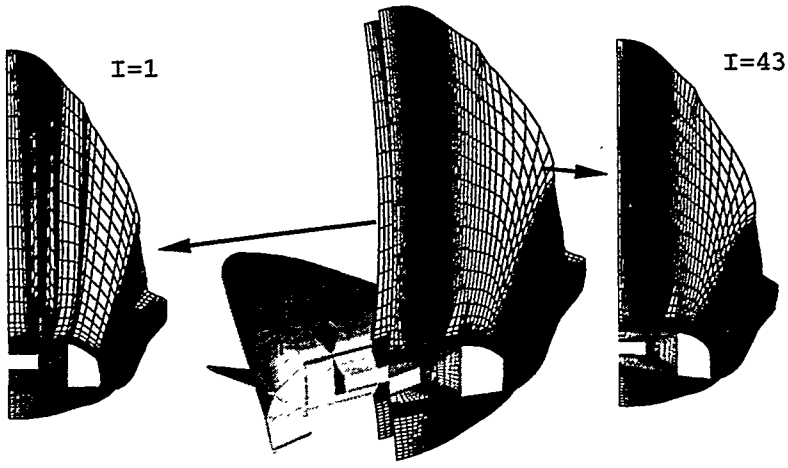
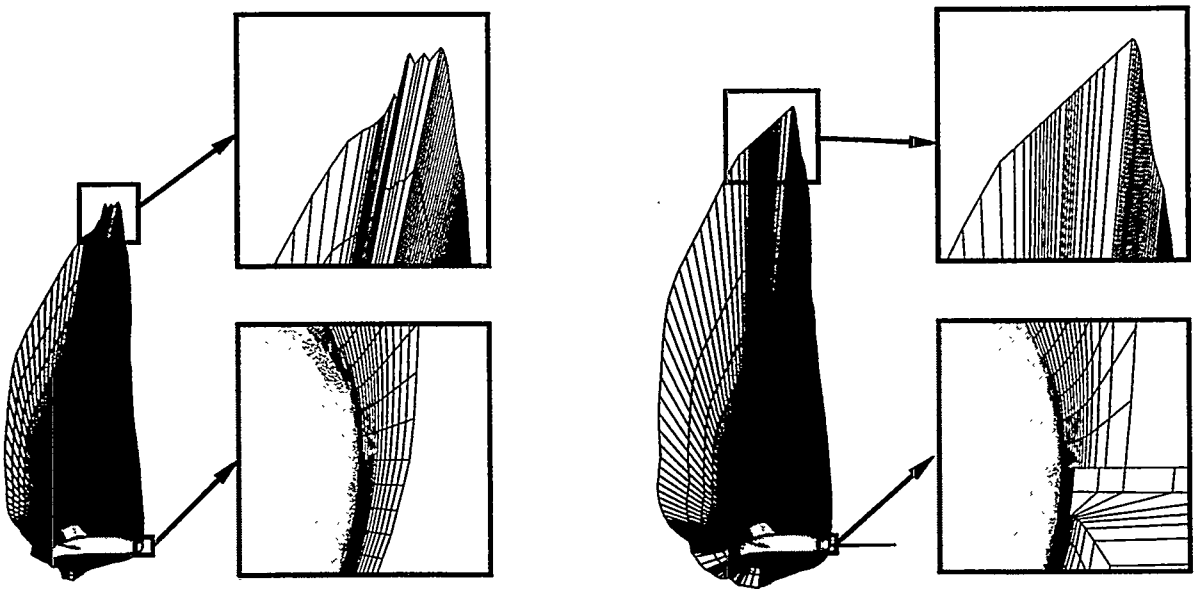


Figure 16: Representative planes of the X33 wake.



(a) Extrapolated

(b) Local coordinate expansion

Figure 17: Good and poor results from two grid expansion techniques.

6 Grid Expansion for Flow Capture

Another application of the grid generation methods of the VGM code is the expansion of a volume grid to enable flow field capture and to correct the negative volumes that can result. The VGM code also has the capability of merging several volume grids such that the best characteristics of each can be combined into the final volume grid. Using this grid merging technique, any number of methods can be used to expand a volume grid. For the case at hand two techniques are used, extrapolation along grid lines, and localized coordinate expansions with an exponential growth function[3]. The extrapolation along grid lines is accomplished with VGM, but produces the typical result of crossed grid lines, as shown in Fig. 17a. Although the extrapolation does not produce a usable grid everywhere, the forebody region of the volume grid is excellent. The second expansion, using a localized coordinate expansion, is shown in Fig. 17b. Again, the usable portion of this grid is the aft region because the forebody region has a spike due to near pole boundary problems which is common when using this technique.

As shown in Fig. 17a, a common side effect of applying extrapolation for grid expansion is the generation of negative volume cells. The severity of the negative cells is usually measured by the number of cells and the number of regions that contain the negative volume cells. Increases in these numbers indicate an increase in negative cell volume severity. Depending on the severity of the grid line curvature, there are several methods that can be used to eliminate these negative volume cells. These are in the order of increasing negative volume cell severity:

(1) TFI in two dimension can be used on the most severe surface and the volume regenerated with 3D-TFI.

(2) A spline curve, surface, or volume that smoothes over the section of negative volumes can be created accompanied by an application 2D-TFI on the faces and 3D-TFI on the volume to re-generate the volume grid.

(3) Redistributing along a coarse set of grid lines that are used as defining curves for the domain.

(4) Merging of multiple volume grids resulting from various domain expansion techniques.

For the current situation, the fourth technique is used because the number of negative volume cells comprises nearly 40% of the volume grid which occur in over 8 different regions of the domain. This technique merges the forebody region of the extrapolated grid to the localized grid point expanded grid using a small interface region. Through the interface region, the contributions of the forebody grid are reduced from 100% at the beginning of the interface to 0% at the end of the interface region. The localized grid expansion region is blended in an opposite manner by

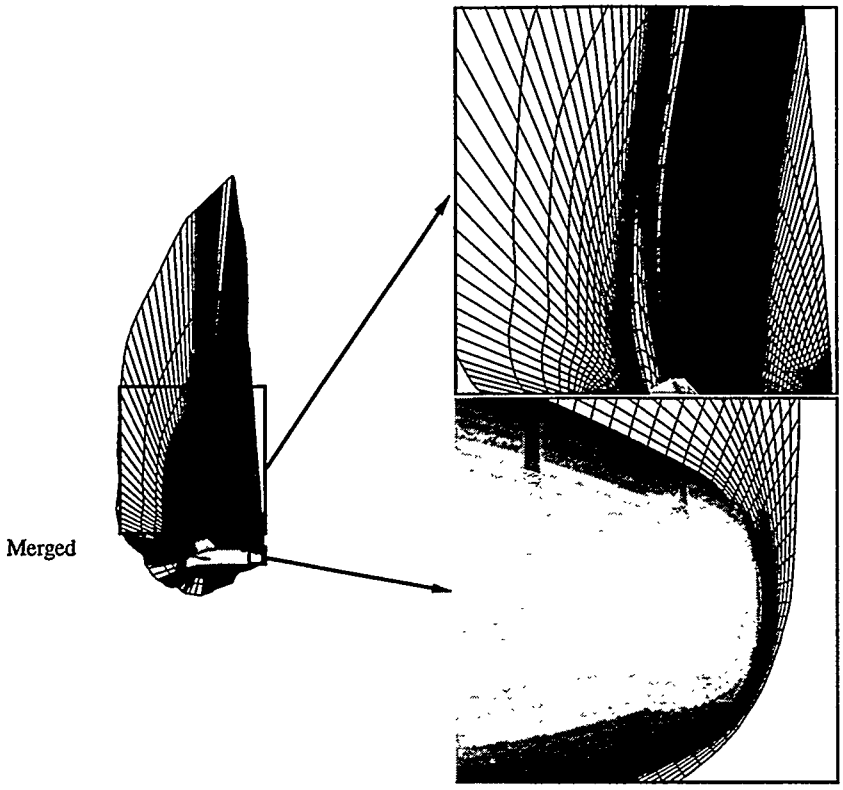


Figure 18: Correction of negative volumes through the merging of high fidelity grid regions.

increasing the contributions at the beginning of the interface region to the end of the region at the same rate that the extrapolated grid is reduced. The results of these manipulations are shown in Fig. 18.

Of the two volume expansion methods, the merging method is more robust as the expanded grid can be obtained through a variety of methods, and only those usable regions need be merged into a master volume grid. The transition scheme used in this grid merging process was a linear function. Numerous functions can be created from the existing distribution functions in VGM as well as the use of the interpolation schemes in VGM. For example, expanding the forebody region ahead of the canted fin root can be done by merging the extrapolated region on the nose to the localized grid expansion on the middle region and merging back to the original grid in the aft region. For these three volume grids, the blending function could be a combination of a sine in the nose region, a cosine in the middle region, and a combined blending function subtracted from unity for the aft region. The resulting merging function is shown in Fig. 19 with the original and final grids shown in Figs. 20 and 21, respectively. Notice that the poor regions of the extrapolated and local coordinate expansions are ignored infavor for the good expanded qualities of each volume grid. By utilizing only those portions of good grid fidelity, a single master volume grid can be easily produced. The zones that are merged have no negative cell volumes, which results in a master volume grid that has similar characteristic quality. Hence, another expanded volume grid can be produced though the merging of three volume grids.

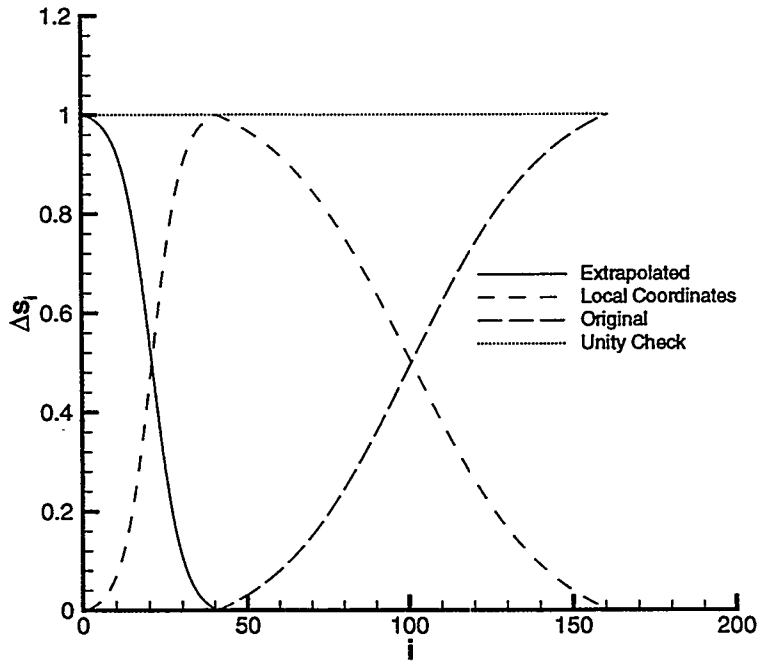


Figure 19: Three function blending used to merge three volume grids.

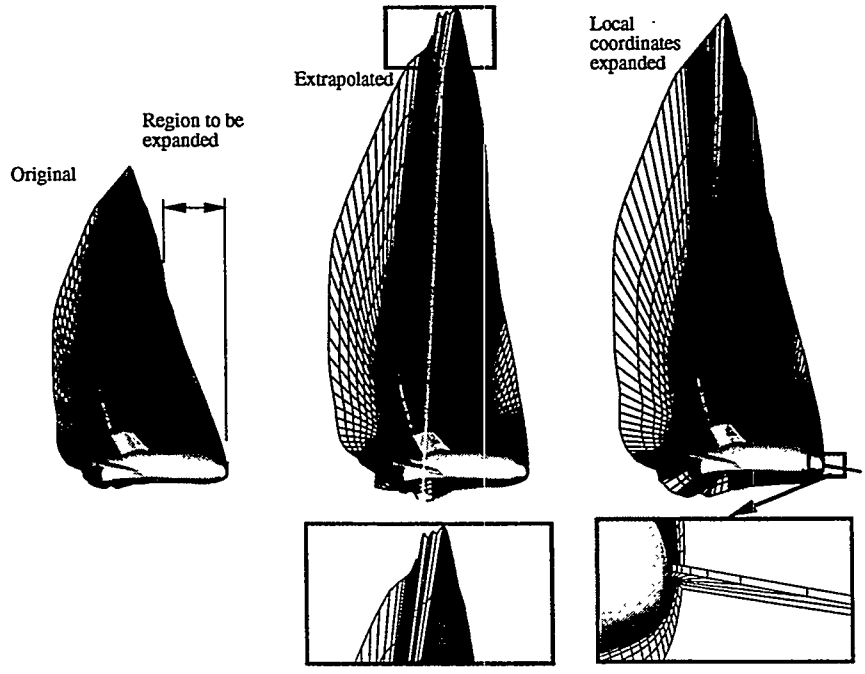


Figure 20: Original three zones to be merged to capture high quality of each.

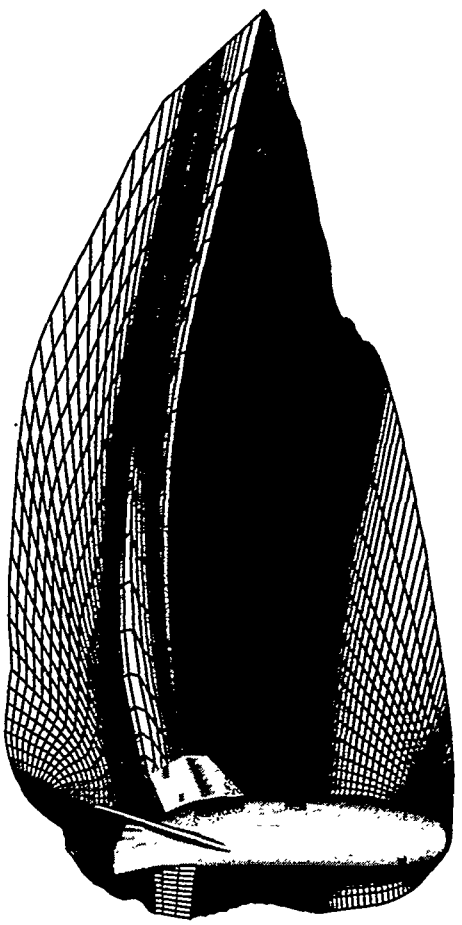


Figure 21: Final merged zones to expand a volume grid.

7 Concluding Remarks

It has been demonstrated in this article that the multiple zone, multiple variable isotropic adaption in three dimensions is a powerful procedure that improves the efficient use of available grid points to capture gradients that are important to the CFD simulation. Use of isotropic grid adaption retains most of the grid quality, while multiple variable dependency provides adequate resolution of the entire flow field. The VGM code enables the use of existing geometry and extracted grid data to produce appended wake domains. Using the scripting capability of this code, wake domain grid generation is simple and quickly repeatable, allowing minor modifications to the resulting discretizations, thus facilitating parametric analysis. Additionally, the scripting and data extraction capabilities of this code permits fast and efficient accommodation of new geometry. Development of a new volume grid can then be parametrically driven, providing a powerful mechanism to assess design changes. The expansion of volume grids through the grid merging technique, provides a hybrid approach for the use of straight line extrapolation and local coordinate scaling to ensure flow capture. The grid merging approach to grid expansion enables the use of any number of grid expansion techniques, by enabling the capture of high fidelity regions in the various expanded grids. Although this technique is not as robust as regenerating the flow domain, it is a fast and powerful alternative that can result in high fidelity discretizations. The techniques described herein are all algebraic operations which translates into reduced cost of grid alterations because of the inherent computational speed, and significantly reduces the cost associated with using structured volume grids for flow field analysis.

Acknowledgments

This work was performed under NASA Langley Research Center contract NAS1-96014. The author wishes to express his appreciation to Dr. Ramadas Prabhu and K. James Weilmuenster for their editorial comments, and William Kleb for his contributions to the typesetting of this document.

References

[1] C. B. Davies and E. Venkatapathy, "A Simplified Self-Adatpive Grid Code, SAGE," NASA TM–102198, October 1989.

[2] P. A. Gnoffo, "An Upwind–Biased Point–Implicit Relaxation Algorithm for Viscous, Compressible Perfect–Gas Flows," NASA TP–2953, February 1990.

[3] S. J. Alter and F. M. Cheatwood, "Volume Grid Expansion Techniques for Computational Fluid Dynamic Algorithms," AIAA Paper 96–0029, January 1996.

[4] S. J. Alter, *The Volume Grid Manipulator (VGM): A Grid Reusability Tool.* NASA CR–4772, April 1997.

[5] P. P. Walatka, P. G. Buning, L. Pierce, and P. A. Elson, "PLOT3D User's Manual," NASA TM–101067, March 1990.

[6] J. P. Steinbrenner, J. R. Chawner, and C. L. Fouts, "The GRIDGEN 3D Multiple Block Grid Generation System," Wright Research and Development Center Report WRDC–TR–90–3022, October 1989.

[7] I. Amtec Engineering, "Tecplot: version 7 User's Manual," Amtec Engineering publication V7.0/96, August 1996.

[8] R. W. Walters, D. C. Slack, P. Cinnella, M. Applebaum, and C. Frost, *GASP Version 3, The General Aerodynamic Simulation Program, Computational Flow Analysis Software for the Scientist and Engineer, User's Manual.* Blacksburg, VA: Aerosoft Incorporated, first ed., May 1996.

[9] V. N. Vatsa and B. W. Wedan, "Development of a Multigrid Code for 3–D Navier–Stokes Equations and Its Application to a Grid–Refinement Study," vol. 18, pp. 391–403, June 1990.

[10] R. A. Benson, *Development of a Time-Accurate Solution Algorithm Coupled to a Dynamic Solution-Adaptive Grid Algorithm with Applications to Generic Inlet/Diffuser Configurations.* PhD thesis, North Carolina State University, 1994.

[11] R. W. Noack and D. A. Anderson, "Solution Adaptive Grid Generation Using Parabolic Partial Differential Equations," AIAA paper 88–0315, January 1988.

[12] S. J. Alter and K. J. Weilmuenster, "Cell Volume Control at a Surface for Three–Dimensional Grid Generation Packages," in *Software Systems for Surface Modeling and Grid Generation* (R. E. Smith, ed.), vol. NASA CP–3143, pp. 273–298, 1992.

[13] M. Vinokur, "On One-Dimensional Stretching Functions for Fininte-Difference Calculations," NASA CR–3313, 1993.

[14] B. R. Hollis and J. N. Perkins, "High–Enthalpy Aerothermodynamics of a Mars Entry Vehicle, Part 2: Computational Results," *AIAA Journal of Spacecraft and Rockets*, vol. 34, pp. 457–463, July–August 1997.

[15] B. R. Hollis and J. N. Perkins, "Transition Effects on Heating in the Wake of a Blunt Body," AIAA Paper 97-2569, June 1997.

[16] S. J. Alter, *The Volume Grid Manipulator (VGM): A Grid Reusability Tool*, ch. 8, pp. 51–52. NASA CR–4772, April 1997.

[17] S. J. Alter, "Complex Volume Grid Generation Through the Use of Grid Reusability," AIAA Paper 97–1987, June 1997.

[18] S. J. Alter and K. J. Weilmuenster, "The Three–Dimensional Multi–block Advanced Grid Generation System (3DMAGGS)," NASA TM–108985, April 1993.

Quadrilateral Meshing

Quadrilateral Meshing with Directionality Control through the Packing of Square Cells

Kenji Shimada * **Jia-Huei Liao** †
Carnegie Mellon University

Takayuki Itoh ‡
IBM Research, Tokyo Research Laboratory

Abstract

This paper proposes a computational method for fully automated quadrilateral meshing. Unlike previous methods, this new scheme can create a quadrilateral mesh whose directionality is precisely controlled. Given as input: (1) a 2D geometric domain, (2) a desired node spacing distribution as a scalar function defined over the domain, and (3) a desired mesh directionality as a vector field defined over the domain, the proposed method first packs square cells closely in the domain. The centers of the squares are then connected by Delaunay triangulation, yielding a triangular mesh topology. The triangular mesh is further converted into a quad-dominant mesh or an all-quad mesh that satisfies the given mesh directionality. Since the closely packed square cells mimic a pattern of Voroni polygons corresponding to a well-shaped graded quadrilateral mesh, the proposed method generates a high quality mesh whose element sizes and mesh directionality conform well to the given input.

Keywords: quadrilateral meshing, unstructured grid, mesh directionality, Voronoi diagram, Delaunay triangulation

1 Introduction

Some FEM analyses prefer quadrilateral meshes over triangular meshes. Examples of such analyses include automobile crash simulation, sheet metal forming simulation, and fluid dynamics analysis. It is also known that 4-node quadrilateral elements perform better than 3-node triangular elements when used in FEM analyses of plain stress and strain[15].

Quadrilateral meshing is often a bottleneck in FEM, however, due to its severe requirements of element shape regularity, precise node spacing control, mesh directionality control, and adaptive remeshing capability. These requirements are also common to triangular meshing, with the exception of mesh directionality control, which is unique to quadrilateral meshing. Quadrilateral meshing usually has a desired "mesh flow direction" predicted by boundary geometries or the directionality of physical phenomena to be analyzed using FEM. For example, in fluid dynamics simulation a quadrilateral mesh should align along shock/boundary layers and stream lines. Similarly, in automobile crash simulation, a mesh should align along the direction of force transmission.

Assuming that grid size distribution is given as a scalar field and the directionality is given as a vector field defined over a domain to be meshed, we propose a computational method that creates a well-shaped, well-aligned, graded quadrilateral mesh. The proposed approach is an extension of the *bubble mesh* method that we previously proposed for triangular meshing [23, 21, 22, 29]. In bubble meshing, a well-shaped graded

*Kenji Shimada, Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
Tel:(412) 268 3614, Fax:(412) 268 3348, shimada@cmu.edu, http://ciel.me.cmu.edu/shimada/
†Jia-Huei Liao, Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
‡Takayuki Itoh, IBM Research, Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato, Kanagawa, 242, Japan.

triangular mesh is created by (1) packing an appropriate number of spherical cells, or bubbles, closely in a domain, while the sizes of the spheres are adjusted based on a specified node spacing function, and (2) connecting the bubbles' centers by constrained Delaunay triangulation to generate node connectivity. The novelty of the bubble mesh process is that the close packing of bubbles mimics a pattern of Voronoi polygons that yields well-shaped triangles.

In this paper, we extend the bubble mesh concept for quadrilateral meshing such that we pack square cells, instead of spherical cells, closely in a domain, mimicking ideal Voronoi polygons that yield a well-shaped quadrilateral mesh. Another major extension over the original bubble mesh is to allow the user to specify a desired mesh directionality by a vector field.

The remainder of the paper is organized as follows. After reviewing previous work we outline our basic approach to quadrilateral meshing. We then elaborate on the technical issues of: (1) how to find node locations suitable for quadrilateral meshing, and (2) how to connect the nodes to obtain a mesh topology that aligns along a specified mesh directionality.

2 Previous Work

There are several reviews available of mesh generation methods [27, 5, 9, 19]. Ho-Le, in his comprehensive survey paper [9], gives a classification based on the temporal order in which nodes and elements are created. The resultant classification is well-accepted and referred to by many other researchers. One problem, as Ho-Le acknowledged in the paper, is that some methods do not seem to fit into any class, while others could be put into two or more classes. In fact, as research in mesh generation has matured, most modern algorithms utilize and combine several sub-processes to improve the quality and efficiency of meshing.

In this section, therefore, we summarize and review some of the key sub-processes commonly used in existing quadrilateral meshing methods. These sub-processes include: (1) node placement and connection, (2) mesh template mapping, (3) element-level domain decomposition, (4) grid-based spatial subdivision, and (5) triangular to quadrilateral mesh conversion. One complete meshing scheme can be characterized by a combination of these sub-processes, performed sequentially or merged into a single process.

Common limitations among previously proposed approaches to quadrilateral meshing include: (1) little or no control over mesh directionality; (2) poor control over node spacing, and/or (3) no efficient adaptive remeshing capability.

2.1 Node placement and connection

In this process, a mesh is constructed in two stages: (1) node placement, and (2) node connection. Node placement and connection can serve as a complete meshing process. The process has become popular due to its conceptual simplicity and the availability of a robust mathematical algorithm for node connection, called *Delaunay triangulation*. When Delaunay triangulation is used for node connection the triangular mesh generated must be converted to a quadrilateral mesh by using a mesh conversion process described later under Triangular to Quadrilateral Mesh Conversion.

During node placement, an appropriate number of nodes needs to be inserted in a well distributed configuration. Several early methods use random node placement followed by validity checks[7, 3, 4, 16]. Lee proposed a CSG-based node placement method[12, 13] in which regular node distribution patterns prescribed for all CSG primitives are combined by Boolean set operations into a single set of nodes.

Although most approaches place all the nodes at one time and then connect them at once in another step, in Frey's and Ruppert's methods[6, 18] two stages of node placement and connection are applied in an iterative manner.

Shimada et al.'s *bubble mesh*[25] and Bossen and Heckbert's *pliant method*[2] use proximity-based forces to find node locations suitable for anisotropic meshing.

2.2 Mesh template mapping

When used for 2D meshing or surface meshing, the template mapping technique maps a prescribed simple mesh template such as a square grid into a given four-sided patch using a blending function. This mapping technique has been one of the most popular approaches in commercial software packages. One drawback

of this method, however, is that it is applicable only to topologically simple domains, and thus it is often necessary for users to subdivide the domain manually into a set of simple subdomains. If this manual subdivision is carefully done the mesh directionality can be controlled to some extent. The process, however, is highly labor intensive, and the mesh directionality cannot be controlled in a precise manner.

2.3 Element level domain decomposition

Element-level domain decomposition refers to the process of subdividing a domain to the element level either by: (1) iterative element extraction[1, 28, 17, 14]; or (2) recursive domain splitting to the element level. The former is more suitable for quadrilateral meshing, and the advancing front method, adopted in many modern commercial packages, is one example of such an algorithm. In Blacker and Stephenson's *paving*[1], meshing fronts that start from domain boundaries are advanced to the interior of the domain, generating quadrilateral mesh elements one by one. A mesh created by an advancing front type of method aligns well along boundaries, a desirable characteristic in most engineering analysis. Such a method, however, cannot control a mesh directionality inside the domain or generate a mesh with an arbitrary mesh directionality.

2.4 Grid-based spatial subdivision

Grid-based spatial subdivision methods superimpose a hierarchical grid, similar to a quadtree, onto the domain to be meshed. Such methods are typically followed by a two-step procedure: (1) classification of grid elements into three types, inside/outside/on-boundary; and (2) adjustment of on-boundary elements to make them consistent with the domain boundary. Yerry and Shephard's *modified octree* is a representative method in this category[30, 20]. A mesh created by a grid-based method typically has a strong directionality in the coordinate axis directions, and it is not possible to adjust mesh directionality over a domain.

2.5 Triangular to quadrilateral mesh conversion

It is well known that any triangular mesh can be converted into a quadrilateral mesh by adding a node to the center of each triangle and by dividing the triangle into three quadrilaterals. Although the idea is straightforward and the implementation is simple, this process introduces a significant topological irregularity into a mesh, and thus it is usually not practical.

More sophisticated ways to convert triangles into quadrilaterals are proposed by Heighway[8] and Jonston et al.[10].

Heighway presents a technique for combining two adjacent triangles into a quadrilateral. Isolated triangles remaining in the mesh are then combined by moving them toward each other until they become adjacent and can be combined.

Johnston proposes a three step procedure: (1) extract boundary information from mesh data, and apply Laplacian smoothing; (2) identify and prioritize corner- and boundary-elements, and perform element by element conversion by coupling elements, splitting a coupled element, and propagating the split to maintain the conformity; and (3) combine all isolated triangles into adjacent quadrilaterals, and divide the combined five-sided elements into three quadrilaterals by introducing nodes inside.

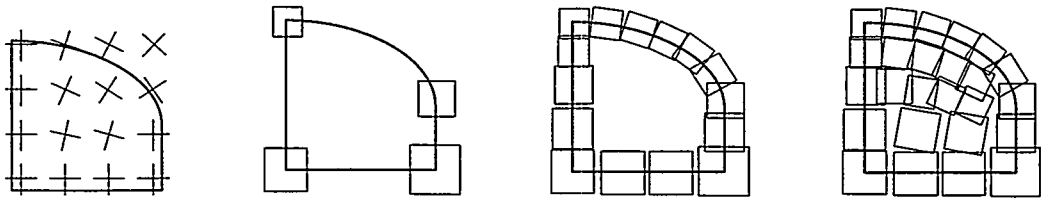
Shimada and Itoh propose a conversion method that uses three conversion templates: (1) from one triangle to three quadrilaterals; (2) from two triangles to four quadrilaterals; and (3) from four triangles to nine quadrilaterals[24]. The method first subdivides a triangular mesh into layers by offsetting boundary, similar to the advancing front method, and then applies conversion templates within each layer.

3 Outline of the Technical Approach

This section describes our basic approach to the following quadrilateral meshing problem.

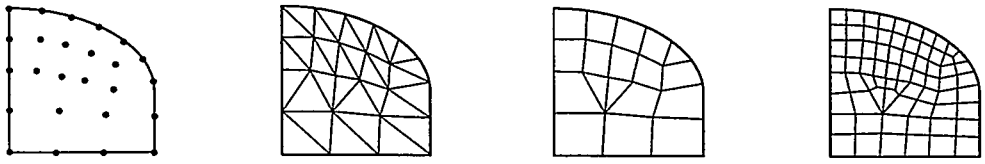
Given:

- a 2D geometric domain
- a desired node spacing distribution $d(\mathbf{x})$, given as a scalar field
- a desired mesh directionality $\mathbf{v}(\mathbf{x})$, given as a vector field



Step 1 Step 2 Step 3

(a) mesh directionality (b) vertex square cells (c) edge square cells (d) face square cells



Step 4 Step 5 Step 6 Step 7

(e) node locations (f) Delaunay triangulation (g) quad-dominant mesh (h) all-quad mesh

Figure 1: Quadrilateral meshing procedure

Generate:

- a well-shaped, graded quadrilateral mesh that is compatible with the given node spacing and mesh directionality

The proposed approach consists of seven steps, as illustrated in Figure 1:

Step 1: Place square cells on all vertices.

Step 2: Pack square cells on all edges.

Step 3: Pack square cells on the face.

Step 4: Place nodes at centers of square cells.

Step 5: Triangulate the domain by Delaunay triangulation.

Step 6: Selectively combine pairs of triangles to generate a quad-dominant mesh.

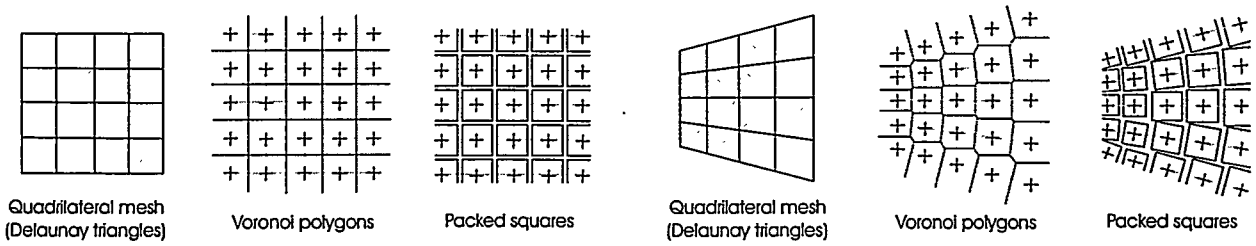
Step 7: Apply mesh conversion templates to obtain an all-quad mesh.

In Steps 1, 2, and 3 we find a node configuration suitable for quadrilateral meshing by closely packing square cells in a domain. The reason we pack squares is that the pattern of packed squares mimics a Voronoi diagram of a well-shaped quadrilateral mesh as shown in Figure 2. Note that the sizes of the cells are adjusted based on a given node spacing distribution $d(\mathbf{x})$ and that the directions of the squares are adjusted based on a given mesh directionality $\mathbf{v}(\mathbf{x})$.

There are two technical issues to be solved in packing square cells tightly in a domain: (1) what are the optimal locations of the squares? (2) how many squares should be packed to fill the domain?

To solve the first issue we use a physically-based model, similar to a particle system in computer graphics. A proximity-based force field is defined between two squares such that the force field exerts an attracting force or a repelling force, moving the cells so that they touch each other along their edges. Also assuming a point mass at the center of each square and the effect of viscous damping, we solve the equation of motion numerically to find a tightly packed configuration of cells.

The second issue of obtaining an appropriate number of squares in the domain is solved by checking the population density and then adaptively adding or removing squares during the numerical integration of the equation of motion, or dynamic simulation.



(a) uniform node spacing

(b) non-uniform node spacing

Figure 2: Close packing of square cells for quad meshing

Because square cells are placed in order of dimension (i.e. vertices, then edges, then faces) two fixed squares are already placed at the two endpoints when squares are packed on an edge; these two end squares are stable throughout the packing process, which prevent moving squares from escaping the range of the edge. Similarly, when squares are packed on the face, the boundary edges are already filled with fixed squares, preventing moving squares from escaping the domain. In this way we put higher priority on the cell placement of lower dimensional elements, i.e., vertex square cells over edge square cells, and edge square cells over face square cells. This strategy is sensible because lower order geometric elements are often more critical than higher order elements in FEM analyses.

Once square cells are packed so that they cover the entire domain without significant gaps and overlaps, their centers are connected by Delaunay triangulation (Steps 4 and 5), yielding a triangular mesh. Pairs of triangles are then selectively connected to create a quad-dominant mesh that aligns along the given mesh directionality (Step 6). When an all-quad mesh is required we further apply mesh conversion templates (Step 7). The edge lengths of the mesh elements in Step 7 are reduced by a factor of two compared with the mesh elements in Step 6.

The next two sections, (1) Close Packing of Square Cells and (2) Mesh Topology Generation, describe the essential elements of Steps 1 to 3 and Steps 5 to 7 respectively.

4 Close Packing of Square Cells

In this section we will first discuss how we can generate mesh directionality over the domain. We will then describe how proximity-based forces and potential fields are specified so that square cells repel or attract each other to yield a force-balancing configuration, or a closely packed configuration.

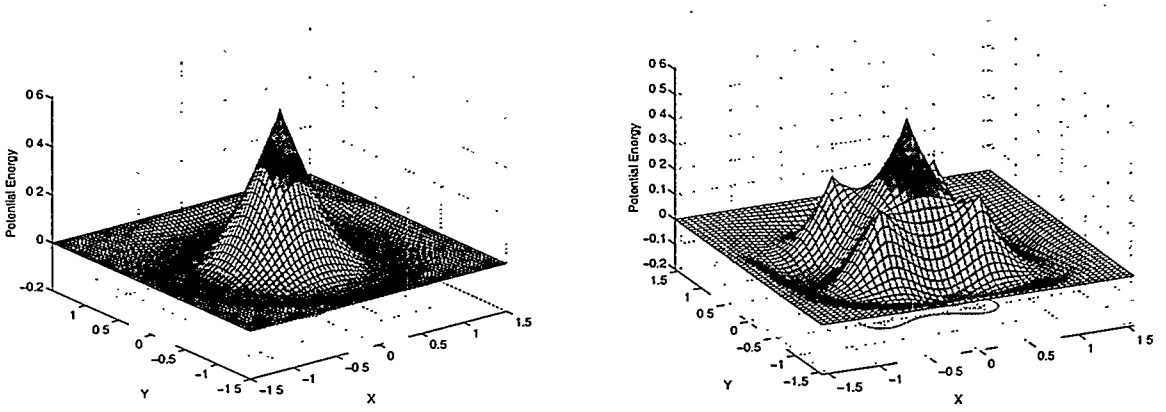
4.1 Mesh directionality

It is important that a desired mesh directionality be specified over the entire domain so that directions of packed square cells are adjusted accordingly. Unless a desired mesh directionality is automatically generated from a previous FEM result, the user typically gives only partial directions or no preference. In such a case it is important that the algorithm generates a complete mesh directionality over the entire domain.

To store a desired mesh directionality we define a background grid that covers the whole domain. The mesh directions are then explicitly stored at the grid nodes, and for an internal point of a grid cell a mesh directionality vector is calculated by linearly interpolating the directions at the four grid nodes.

If mesh directionality vectors are given at only some grid nodes we need to find the mesh directionality vectors at all the others so that the mesh directionality changes smoothly over the domain.

We solve this smooth interpolation problem by using relaxation, similar to Laplacian smoothing, widely used to improve mesh element shapes. As in Laplacian smoothing, which moves a mesh node iteratively to a location which represents the center of gravity of its adjacent node locations, the mesh direction vector at



(a) potential field in the original bubble mesh (b) new potential field for quadrilateral meshing

Figure 3: Potential fields

a grid node is iteratively modified to approach an average of the direction vectors at its four adjacent grid nodes.

4.2 Proximity-based potential fields and forces

In triangular meshing the ideal node configuration is a regular hexagonal arrangement. As proven in the original bubble mesh method [25, 21, 22], such an arrangement can be obtained by defining a force field similar to the van der Waals force, which exerts a repelling force when two molecules are located closer together than the stable distance and exerts an attracting force when two molecules are located farther apart than the stable distance.

Let the positions of adjacent nodes i and j be $\mathbf{x}_i$ and $\mathbf{x}_j$; the current distance between the two nodes $l(\mathbf{x}_i, \mathbf{x}_j)$; the target stable distance $l_0(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}\left(d(\mathbf{x}_i)) + d(\mathbf{x}_j)\right)$, which is a desired element size specified by the node spacing function $d(\mathbf{x})$; the ratio of the current distance and the target distance $w(\mathbf{x}_i, \mathbf{x}_j) = \frac{l(\mathbf{x}_i, \mathbf{x}_j)}{l_0(\mathbf{x}_i, \mathbf{x}_j)}$; and the corresponding linear spring constant at the target distance k_0. The force model used in the original bubble mesh is then written as

$$f(w) = \begin{cases} \frac{k_0}{l_0}\left(\frac{5}{4}w^3 - \frac{19}{8}w^2 + \frac{9}{8}\right), & 0 \le w \le 1.5 \\ 0, & 1.5 < w. \end{cases} \tag{1}$$

By integrating the above force field we obtain the following potential field around the center P of the potential field.

$$\Psi_P(w) = \begin{cases} -\frac{k_0}{l_0}\left(\frac{5}{16}w^4 - \frac{19}{24}w^3 + \frac{9}{8}w - \frac{153}{256}\right), & 0 \le w \le 1.5 \\ 0, & 1.5 < w, \end{cases} \tag{2}$$

Figure 3(a) shows this potential field function used in the original bubble mesh for triangular meshing.

This potential field applies either a repelling or attracting force between two nodes based on the following distance comparison. Assuming that two nodes are adjacent to each other, a repelling force is applied if l is smaller than l_0, or if $w < 1.0$. An attracting force is applied if l is larger than l_0, or if $1.0 < w \le 1.5$. No force is applied if two nodes are located exactly at the stable distance or if they are located much farther apart, the cases where $w = 1.0$ or $1.5 < w$. Note that the potential field shown in Figure 3(a) has circular stable positions–anywhere on the circle is equally stable.

In achieving a close packing of squares, however, the potential field shown in Figure 3(a) is not appropriate because it does not take into account mesh directionality, essential to quadrilateral meshing. Considering a mesh directionality there should be only four stable locations around a node, as shown in 3(b), and each of the stable locations corresponds to a situation where two square cells are placed side by side with their edges touching each other. In order to force squares to align this way, we need to add to the original potential

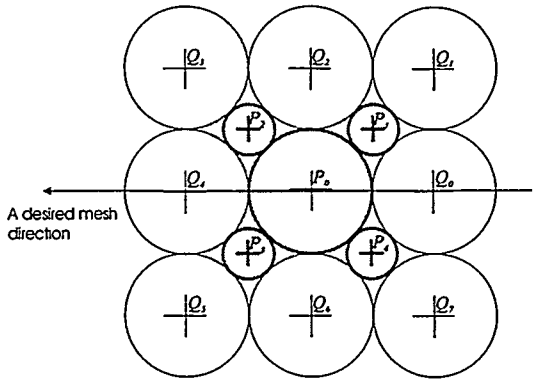


Figure 4: Stable positions in packing square cells

field four sub-potential fields Ψ_{P_1}, Ψ_{P_2}, Ψ_{P_3}, and Ψ_{P_4} at the four corners of a square P_1, P_2, P_3 and P_4 as shown in Figure 4.

If the desired element size is locally uniform the radii of the four sub-potential fields should be $(\sqrt{2}-1)r_0$, where r_0 is the radius of the central potential field Ψ_{P_0}. If graded element sizes are specified, however, the radii of the sub-potentials should be adjusted accordingly.

The potential field shown in Figure 3(b) is thus expressed as a weighted linear combination of the central potential field and the four sub-potential fields, i.e.,

$$\Psi = \Psi_{P_0} + (\sqrt{2}-1)(\Psi_{P_1} + \Psi_{P_2} + \Psi_{P_3} + \Psi_{P_4}). \tag{3}$$

With the above potential field, the primary stable positions of the squares surrounding square P_0 are Q_0, Q_2, Q_4 and Q_6 as shown in Figure 4. Once these primary stable positions are occupied by square cells, then Q_1, Q_3, Q_5 and Q_7 also become stable positions.

4.3 Force-balancing configuration of square cells

Given the proximity-based intercell force, we apply physically-based relaxation to find a close packing configuration of square cells. This is also a configuration that yields a static force balance.

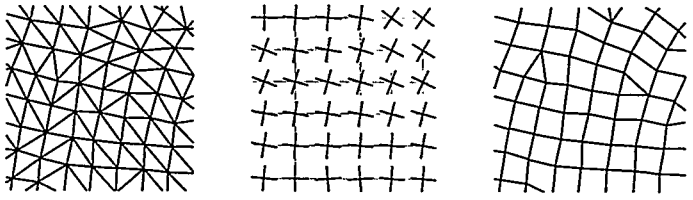
Due to the nonlinearity of the force and complex geometric constraints on square locations, the force balance equation becomes highly nonlinear, and thus it is difficult to solve the equation directly by a multi-dimensional root-finding technique such as the Newton-Raphson method.

Our alternative approach is to assume a point mass m at the center of each cell and the effect of viscous damping c, and to solve the following equation of motion[1] by using a standard numerical integration scheme such as the fourth-order Runge-Kutta method.

$$m\ddot{\mathbf{x}}_i(t) + c\dot{\mathbf{x}}_i(t) = \mathbf{f}_i(t), \qquad i = 1, \ldots, n. \tag{4}$$

In solving Equation (4) numerically, we adaptively adjust the number of square cells packed in the domain. This is important because we do not know beforehand an appropriate number of squares that is necessary and sufficient to fill the region. We generate an initial configuration by using octree subdivision, and although this process gives a reasonably good guess of the number of squares it is still not optimal. We therefore implemented a procedure to check a local population density and to add more squares in sparse areas and delete squares in over-packed areas.

[1]The first order equation can also be used [2]. In either case, the essential point is that after a certain number of iterations the system reaches a virtual equilibrium, where both the velocity term $\dot{x}$ and the acceleration term $\ddot{x}$ approach zero, leaving a static force balance.



(a) triangular mesh (b) mesh directionality (c) quad mesh

Figure 5: Converting a triangular mesh into a quad-dominant mesh

Note that the dynamic simulation and the adaptive node population control described above make efficient adaptive remeshing possible because we do not need to rebuild a mesh from scratch when the domain geometry, node spacing, and/or mesh directionality is slightly modified.

5 Mesh Topology Generation

Once a force-balancing configuration of squares is obtained, the squares' centers must be connected to form a complete quadrilateral mesh. In connecting nodes, *Delaunay triangulation* is first applied to create a triangular mesh, and the triangular mesh is then converted into a quad-dominant mesh by selectively merging two adjacent triangular elements into a quadrilateral element in such a way that the resultant mesh aligns along the specified mesh directionality (see Figure 5). In the final step the quad-dominant mesh is converted into an all-quad mesh by applying two mesh conversion templates: (1) splitting a quad element into four quad elements, and (2) splitting a triangular element into three quad elements.

In converting a triangular mesh to a quad-dominant mesh we use the following three steps so that the resultant mesh aligns along the specified mesh directions. This procedure is based on the practice of removing the shared edge between two adjoining triangles in order to form a quadrilateral element.

1. For the ith non-boundary edge of a triangular element, calculate a score Λ_i that measures how well the resultant quadrilateral element aligns along the specified mesh directions if the edge is removed to form a quadrilateral.

2. Make a priority queue of all the non-boundary edges by sorting the scores assigned to the edges.

3. Delete edges one by one from the top of the priority queue—one edge deletion creates one quadrilateral element.

The quality score Λ_i of a possible quadrilateral element is calculated by comparing the directions of the four side edges of the resultant quadrilateral element with specified mesh direction vectors at the centers of the four edges. For the jth side edge of the quadrilateral element, we take the absolute value of the inner product λ_{ij} of: (1) the unit vector $\mathbf{u}_{ij}$ of the side edge; and (2) the mesh direction vector $\mathbf{v}_{ij}$ at the center of the edge or the unit vector orthogonal to the mesh direction. λ_{ij} is thus expressed

$$\lambda_{ij} = \begin{cases} |\mathbf{u}_{ij} \cdot \mathbf{v}_{ij}|, & |\mathbf{u}_{ij} \cdot \mathbf{v}_{ij}| \geq \frac{1}{\sqrt{2}} \\ \sqrt{1 - (\mathbf{u}_{ij} \cdot \mathbf{v}_{ij})^2}, & |\mathbf{u}_{ij} \cdot \mathbf{v}_{ij}| < \frac{1}{\sqrt{2}} \end{cases} \tag{5}$$

where $j = 1, 2, 3, 4$, the subscript i represents the index of a quadrilateral element, and the subscript j the index of the side edge of the quadrilateral element. Note that the the value of λ_{ij} is bounded between $\frac{1}{\sqrt{2}}$ and 1.

Using the λ defined above we can calculate the score Λ_i as follows, and it measures how well the ith quadrilateral element aligns along the given mesh direction vector $\mathbf{v}(\mathbf{x})$

$$\Lambda_i = \frac{1}{4}\sum_{j=1}^{4}\lambda_{ij} \tag{6}$$

The value of Λ_i is bounded between $\frac{1}{\sqrt{2}}$ and 1, and as Λ_i approaches 1 the ith quadrilateral element aligns more accurately along the mesh direction vector field.

6 Results and Discussions

The proposed quadrilateral meshing algorithm has been implemented in C and C++ on Unix workstations (IBM RS6000 and SGI O2) and Windows PCs.

In this section we measure the quality of generated quadrilateral meshes using two types of mesh irregularity measures, *topological irregularity* and *geometric irregularity*.

For topological irregularity, we use the following measure [24]:

$$\varepsilon_t = \frac{1}{n}\sum_{i=0}^{n}|\delta_i - D|, \tag{7}$$

where δ_i represents the *degree*, or the number of neighboring nodes, and n represents the total number of nodes in the mesh. $D = 4$ if the ith node is an internal node; $D = 3$ if the ith node is a boundary node. As the mesh becomes topologically similar to a structured grid this topological irregularity approaches 0, but vanishes only when the mesh is perfectly structured, a rare situation. Otherwise, it has a positive value that measures how much the mesh topologically differs from a perfectly regular structured grid.

For geometric irregularity we define the measure, ε_g, that is the ratio of the radius of the *minimum inscribed circle* [2] to the radius of the *maximum circumcircle* [3]. Geometric irregularity ε_g is thus calculated as

$$\varepsilon_g = \frac{1}{m}\sum_{i=0}^{m} g_i, \tag{8}$$

where $g_i = \left(\frac{1}{\sqrt{2}} - \frac{r_i}{R_i}\right)$, m is the number of quadrilaterals, r_i the minimum inscribed circle radius of the ith quadrilateral, and R_i the maximum circumcircle radius of the ith quadrilateral. Since the ratio r_i/R_i takes its maximum value $\frac{1}{\sqrt{2}}$ for a perfect square element, an ideal element, the smaller the value of ε_g, the more geometrically regular the quadrilateral mesh.

Five meshing results are shown in Figures 6, 7, 8, 9, and 10, and some statistics are shown in Table 1 and Figure 11 for the first four meshes. Table 1 summarizes the mesh statistics including: (1) the numbers of mesh nodes and elements; (2) CPU times for the initial meshing and CPU times for 100 iterations of dynamic simulation; and (3) mesh irregularity measure. All the CPU times are measured on a SGI O2 workstation with a R5000/180MHz CPU.

In generating Mesh 1 and Mesh 2 shown in Figures 6 and 7 respectively the vector fields that represent desired mesh directions are automatically generated from the boundary geometry. The final quadrilateral meshes are thus aligned along the boundary directions. The node spacing functions are uniform so that the domain is packed with squares of a uniform size, yielding uniform quadrilateral meshes.

In Mesh 3 shown in Figure 8 a non-uniform node spacing function is specified to generate a graded quadrilateral mesh. Note that the sizes of the packed square cells in Figure 8(c) are adjusted based on the node spacing function shown in Figure 8(b), yielding the well-shaped, graded quadrilateral mesh shown in Figure 8(f).

Mesh 4 and Mesh 5 shown in Figures 9 and 10 respectively are meshes of the same geometric domain. The two meshes are created, however, using different mesh direction vector fields. In Mesh 4 the mesh directions are specified so that they align along the domain boundary, and in Mesh 5 the mesh directions are uniform. Note that both meshes are well aligned along the specified mesh directions.

[2]The minimum inscribed circle is the smallest circle tangent to at least three edges of a quadrilateral element.

[3]The maximum circumcircle is the largest circle that goes through at least three vertices of a quadrilateral element.

Table 1: Mesh statistics.

Mesh	*Number of elements in all-quad mesh*	*Number of nodes, quad, and tri in quad-dominant mesh*	*CPU time initial mesh*	*CPU time 100 iterations**	*Mesh irregularity after convergence*	
Mesh 1	743	222, 167, 25	0.787 sec.	4.366 sec.	$\varepsilon_t = 0.18468$	$\varepsilon_g = 0.09933$
Mesh 2	1748	488, 401, 48	1.259 sec.	11.631 sec.	$\varepsilon_t = 0.12705$	$\varepsilon_g = 0.08428$
Mesh 3	617	166, 134, 27	0.660 sec.	2.480 sec.	$\varepsilon_t = 0.19880$	$\varepsilon_g = 0.08841$
Mesh 4	804	239, 180, 28	0.794 sec.	4.432 sec.	$\varepsilon_t = 0.13389$	$\varepsilon_g = 0.09867$

* Approximately 50 to 100 iterations are sufficient to generate a reasonably good mesh.

7 Conclusion

We have presented a new physically-based method for well-shaped, graded quadrilateral meshing of a 2D region. Our central idea was to pack squares closely in a domain to mimic a pattern of Voronoi polygons corresponding to a well-shaped, graded quadrilateral mesh. To obtain a close packing of squares, we proposed a physically-based approach using a proximity-based potential field.

The most powerful feature of this new approach is that we can specify arbitrary mesh directionality as a vector field defined over a domain as well as arbitrary node spacing as a scalar field. The mesh directionality can be either: (1) manually specified by the user; (2) automatically generated from domain boundary directions; or (3) automatically generated from a previous analysis result.

One advantage of our physically-based packing of square cells is that the quadrilateral elements generated are so well-shaped that no further smoothing or *topological cleanup* [26, 11] is necessary. Most previous approaches require smoothing or topological cleanup to improve the mesh quality, and these operations often destroy the node spacing or mesh directionality in the original mesh.

Another advantage of using dynamic simulation is that it makes adaptive remeshing efficient. Adaptive remeshing is necessary in some FEM analyses in which the domain boundary, node spacing, and/or mesh directionality change over time. Fluid dynamics simulations with moving boundaries and large deformation structural analyses fall into this category. In these analyses it is possible that a mesh becomes too distorted over time to yield a valid computational result, and the mesh has to be updated. Our method can handle this remeshing efficiently because it updates the mesh easily by running a few iterations of dynamic simulation without constructing the new mesh from scratch.

A potential limitation of the proposed method is its relatively expensive computational cost compared to some of the purely geometric approaches. The method, therefore, can best be utilized in applications that benefit from regular element shapes, well-controlled element sizes, and well-controlled mesh directionality. Such applications include FEM analysis of thermal/fluid dynamics simulation, automobile crash simulation, and sheet metal forming simulation.

Finally, like the original bubble mesh method for triangular and tetrahedral meshing, the proposed method can be naturally extended to quadrilateral meshing of a parametric surface and hexahedral meshing of a solid by packing cubical cells instead of square cells.

References

[1] T.D. Blacker and M.B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *Intl. J. Numer. Meth. Eng.*, 32:811–847, 1991.

[2] Frank J. Bossen and Paul S. Heckbert. A pliant method for anisotropic mesh generation. In *Proc. of 5th Intl. Meshing Roundtable*, pages 63–74, 1996.

[3] J.C. Cavendish. Automatic triangulation of arbitrary planar domains for the finite element method. *Intl. J. Numer. Meth. Eng.*, 8:679–696, 1974.

[4] J.C. Cavendish, D.A. Field, and W.H. Frey. An approach to automatic three-dimensional finite element mesh generation. *Intl. J. Numer. Meth. Eng.*, 21:329–347, 1985.

[5] M.S. Shephard et al. Trends in automatic three-dimensional mesh generation. *Computers and Structures*, 30(1/2):421–429, 1988.

[6] W.H. Frey. Selective refinement: A new strategy for automatic node placement in graded triangular meshes. *Intl. J. Numer. Meth. Eng.*, 24:2183–2200, 1987.

[7] J. Fukuda and J. Suhara. Automatic mesh generation for finite element analysis. In J.J. Oden, editor, *Advances in Computational Methods in Structural Mechanics and Design*, Huntsville, Alabama, U.S.A., 1972. UAH Press.

[8] E.A. Heighway. A mesh generator for automatically subdividing irregular polygons into quadrilaterals. *IEEE Transactions on Magnetics*, Mag-19, 1983.

[9] K. Ho-Le. Finite element mesh generation method: A review and classification. *Computer-Aided Design*, 20(1):27–38, 1988.

[10] B.P. Johnston, J.M. Sullivan Jr., and A. Kwasnik. Automatic conversion of triangular finite element meshes to quadrilateral elements. *Intl. J. Numer. Meth. Eng.*, 31, 1991.

[11] Paul Kinney. Clean up: Improving quadrilateral finite element meshes. *Proc. of 6th Intl. Meshing Roundtable*, pages 449–461, 1997.

[12] Y.T. Lee. *Automatic Finite Element Mesh Generation Based on Constructive Solid Geometry.* PhD thesis, University of Leeds, Leeds, England, 1983.

[13] Y.T. Lee. Automatic finite-element mesh generation. *ACM Transactions on Graphics*, 3(4):287–311, 1984.

[14] Randy R. Lober, Timothy J. Tautges, and Rich A. Cairncross. The parallelization of an advancing-front, all-quadrilateral meshing algorithm for adaptive analysis. *Proc. of 4th Intl. Meshing Roundtable*, pages 59–70, 1995.

[15] Anish Malanthara and Walter Gerstle. Comparative study of unstructured meshes made of triangles and quadrilaterals. *Proc. of 6th Intl. Meshing Roundtable*, pages 437–447, 1997.

[16] A.O. Mascardini, B.A. Lewis, and M. Cross. Agthom - automatic generation of triangular and higher order meshes. *Intl. J. Numer. Meth. Eng.*, 19:1331–1353, 1983.

[17] Matthew Rees. Combining quadrilateral and triangular meshing using the advancing front approach. *Proc. of 6th Intl. Meshing Roundtable*, pages 337–348, 1997.

[18] J. Ruppert. *Results on Triangulation and High Quality Mesh Generation.* PhD thesis, Univeristy of California at Berkeley, CA, U.S.A., 1992.

[19] N. Sapidis and R. Perucchio. Advanced techniques for automatic finite element meshing from solid models. *Computer-Aided Design*, 8(4):248–253, 1989.

[20] M.S. Shephard and M.K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *Intl. J. Numer. Meth. Eng.*, 32:709–749, 1991.

[21] Kenji Shimada. *Physically-Based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, U.S.A., 1993.

[22] Kenji Shimada and David C. Gossard. Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing. In *Third Symp. on Solid Modeling and Appls.*, pages 409–419, 1995.

[23] Kenji Shimada and David C. Gossard. Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis. *Computer Aided Geometric Design*, 15/3:199–222, 1998.

[24] Kenji Shimada and Takayuki Itoh. Automated conversion of 2d triangular meshes into quadrilateral meshes. In *Proc. of International Conference on Computational Engineering Science*, 1995.

[25] Kenji Shimada, Atsushi Yamada, and Takayuki Itoh. Anisotropic triangulation of parametric surfaces via close packing of ellipsoids. *Intl. J. on Computational Geometry and Applications*, 1997. submitted.

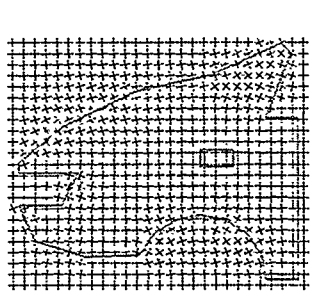
[26] Mathew L. Staten and Scott A. Canann. Post refinement element shape improvement for quadrilateral meshes. *Trends in Unstructured Mesh Generation, ASME*, 220:9–16, 1997.

[27] W.C. Thacker. A brief review of techniques for generating irregular computational grids. *Intl. J. Numer. Meth. Eng.*, 15:1335–1341, 1980.

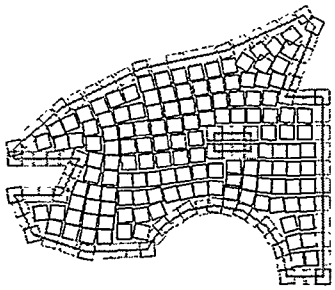
[28] David R. White and Paul Kinney. Redesign of the paving algorithm: Robustness enhancements through element by element meshing. *Proc. of 6th Intl. Meshing Roundtable*, pages 323–335, 1997.

[29] Atsushi Yamada, Kenji Shimada, and Takayuki Itoh. Energy-minimizing approach to meshing curved wire-frame models. In *Proc. of 5th Intl. Meshing Roundtable*, pages 179–191, 1996.

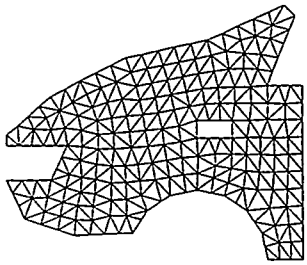
[30] M.A. Yerry and M.S. Shephard. A modified-quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3:39–46, 1983.



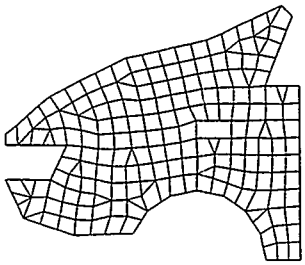
(a) mesh directionality



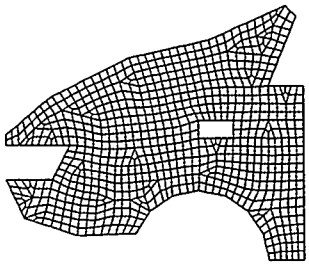
(b) packed square cells



(c) triangular mesh

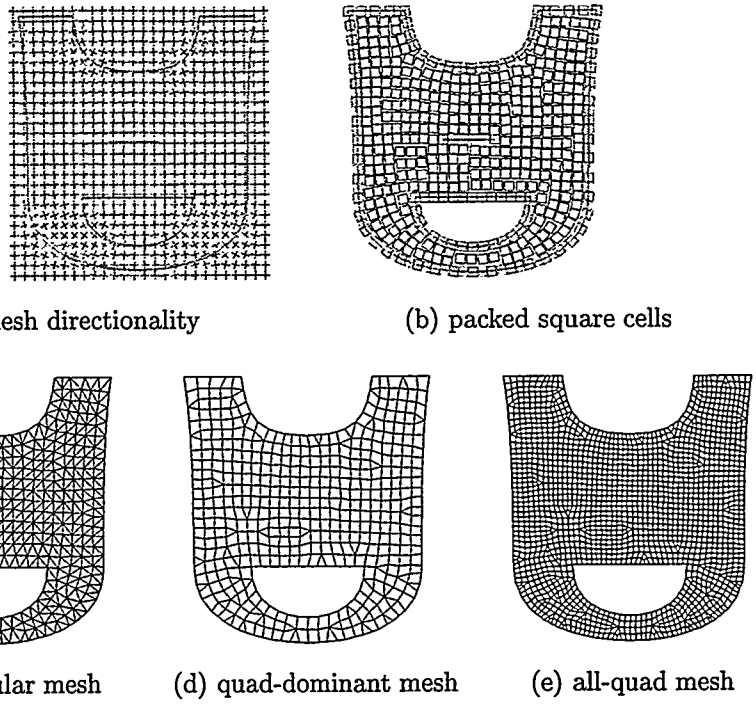


(d) quad-dominant mesh



(e) all-quad mesh

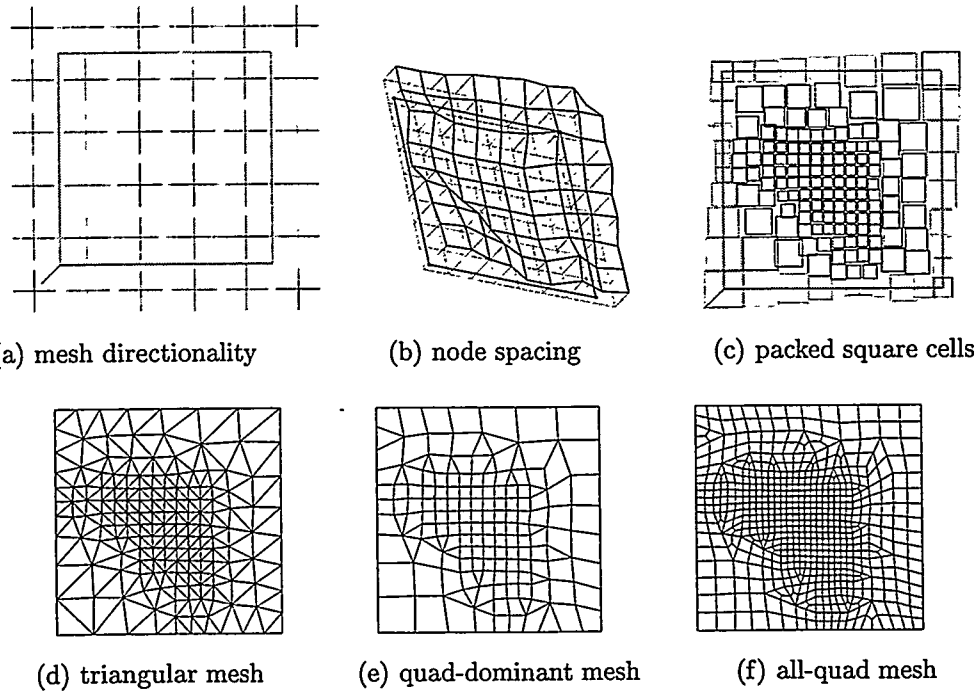
Figure 6: Mesh 1: uniform size, mesh directionality aligned along boundary



(a) mesh directionality (b) packed square cells

(c) triangular mesh (d) quad-dominant mesh (e) all-quad mesh

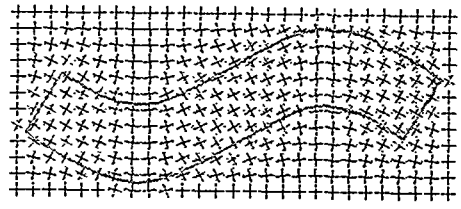
Figure 7: Mesh 2: uniform size, mesh directionality aligned along boundary



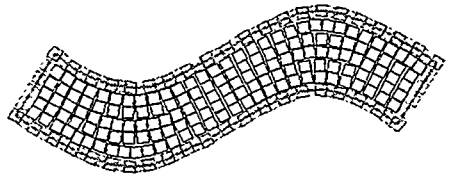
(a) mesh directionality (b) node spacing (c) packed square cells

(d) triangular mesh (e) quad-dominant mesh (f) all-quad mesh

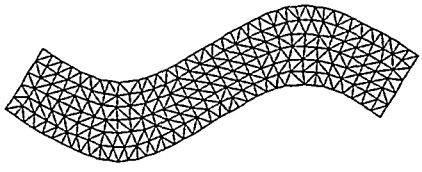
Figure 8: Mesh 3: graded size, uniform mesh directionality



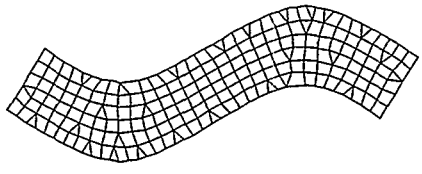
(a) mesh directionality



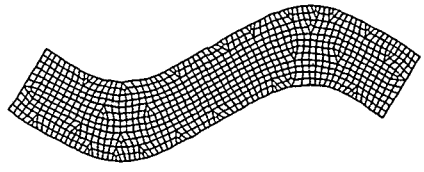
(b) packed square cells



(c) triangular mesh

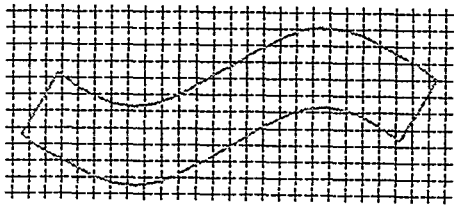


(d) quad-dominant mesh

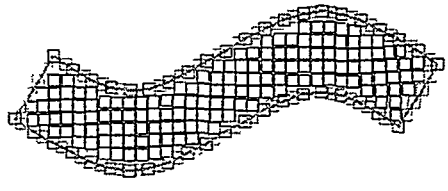


(e) all-quad mesh

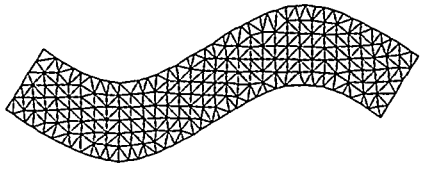
Figure 9: Mesh 4: uniform size, mesh directionality aligned along boundary



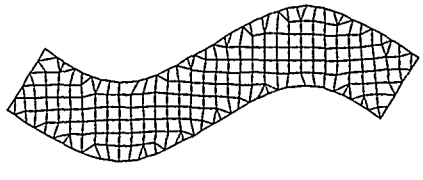
(a) mesh directionality



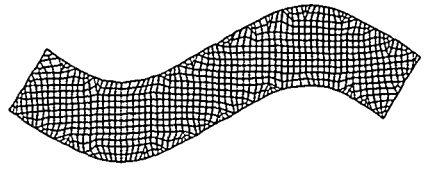
(b) packed square cells



(c) triangular mesh

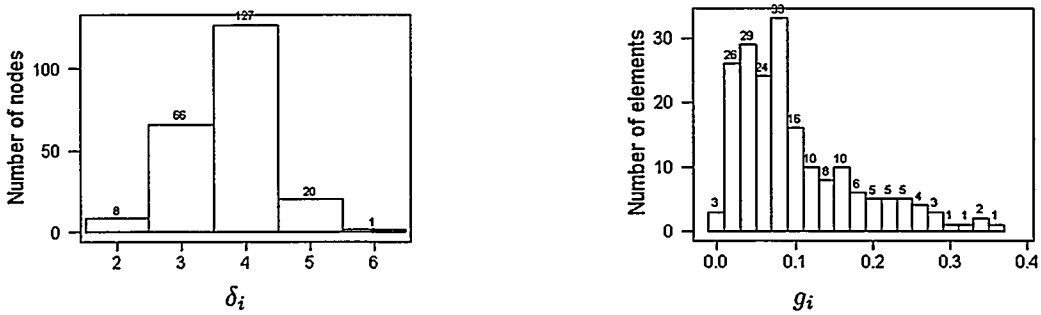


(d) quad-dominant mesh

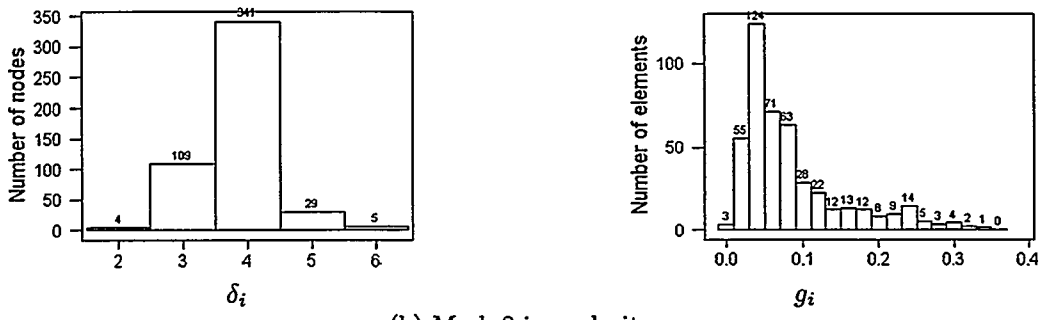


(e) all-quad mesh

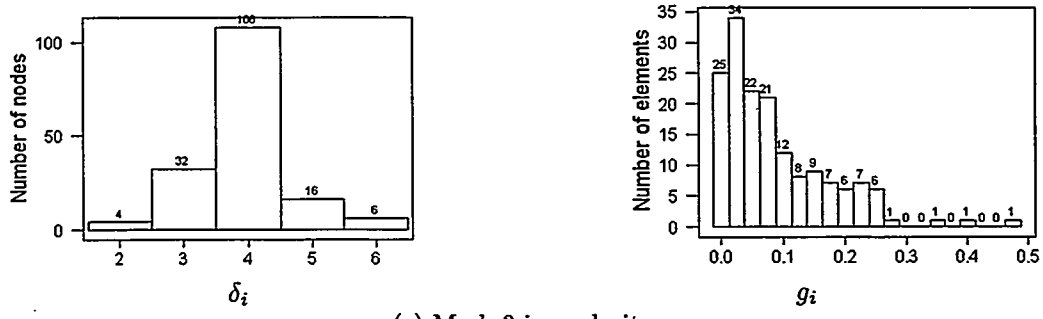
Figure 10: Mesh 5: uniform size, uniform mesh directionality



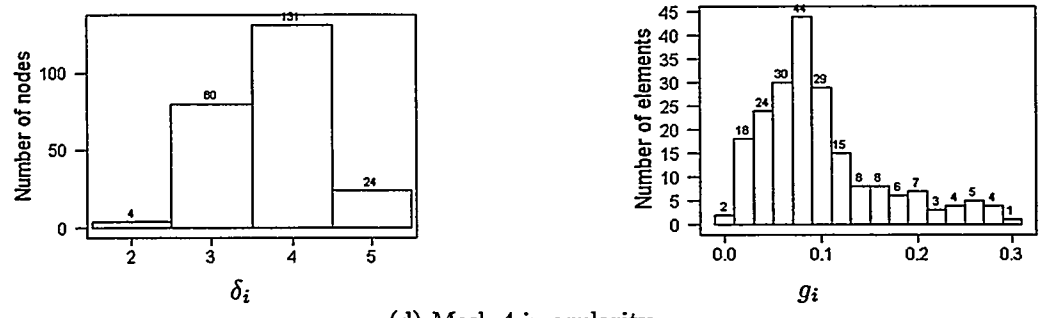
(a) Mesh 1 irregularity



(b) Mesh 2 irregularity



(c) Mesh 3 irregularity



(d) Mesh 4 irregularity

Figure 11: Topological irregularity and geometric irregularity

Automated Conversion of 2D Triangular Mesh into Quadrilateral Mesh with Directionality Control

Takayuki ITOH * Kenji SHIMADA ** Keisuke INOUE *
Atsushi YAMADA * Tomotake FURUHATA ***

* IBM Research, Tokyo Research Laboratory
** Carnegie Mellon University, Mechanical Engineering
*** IBM Japan, AP Solution Development

* { itot, inoue, ayamada } @trl.ibm.co.jp ** shimada@cmu.edu *** furuhata@yamato.ibm.co.jp

Abstract: *This paper proposes an automated quadrilateral meshing technique based on the conversion of triangular meshes. Given a triangular mesh and a vector field representing the desired directionality of quadrilateral elements, our new approach generates a well-shaped and well-aligned quadrilateral mesh. In the approach, the values of three scalar functions are first calculated for each pair of adjacent triangular elements. These functions evaluate the following conditions: (1) directionality control, (2) optimization of the shapes of elements, and (3) reduction of the number of isolated triangular elements. Pairs of triangular elements are then converted into quadrilateral elements in order of the sum of the values of the three functions.*

Keywords: quadrilateral mesh, square packing, bubble mesh, directionality.

1 Introduction

In some types of FEM analysis, such as sheet-metal forming simulation and automobile crash simulation, quadrilateral meshes are preferable to triangular meshes because they allow more accurate and efficient simulation. However, conventional quadrilateral meshing techniques do not always satisfy the requirements of such simulations. We think that quadrilateral meshing techniques should satisfy the following requirements:

Requirement 1: Fully automated operation

For example, mapping methods [9], which are implemented in many commercial meshing programs, require manual operations by users to decompose given geometric domains into triangular or quadrilateral subdomains. Manual operations should be omitted from the meshing procedures, because such operations sometimes require lengthy design processes.

Requirement 2: Adaptive control of the size of elements

For efficiency of simulations, it is desirable that the size of elements should be freely controlled, and that fine elements should be generated only in regions that have sensitive solutions.

Requirement 3: Directionality control of quadrilateral elements

Many simulations require quadrilateral elements to be aligned along the boundary of given domains or physical phenomena. It is therefore desirable to generate quadrilateral elements aligned along arbitrary vector fields encompassing the geometric domains given by users.

Requirement 4: Well-shaped quadrilateral elements

Generally, it is desirable that all quadrilateral elements in a mesh should be almost square or rectangular. Distorted elements often give poor solutions in the abovementioned simulations.

One of the most popular quadrilateral meshing approaches is the conversion of triangular meshes [2] [3] [4] [5] [6]. Here, Requirements 1 and 2 are satisfied by the use of such conversion techniques, since there are triangulation methods that can automatically generate size-controlled elements in complicated geometric domains. However, we do not think that the conversion techniques always satisfy the other two requirements which concern the directionality and shape of elements. We think that the following Condition A needs to be satisfied in order to meet Requirement 3. Similarly, we think that the following Conditions B and C need to be satisfied in order to meet Requirement 4:

(Condition A) Alignment of elements with the given directions

It is desirable that the given direction and the four edges of a quadrilateral element be almost parallel or perpendicular.

(Condition B) Improvement of geometric irregularity

It is desirable that the angles of the four vertices of a quadrilateral element be approximately right angles.

(Condition C) Reduction of the number of isolated triangular elements

It is desirable that the largest possible number of adjacent triangular elements be coupled and converted into quadrilateral elements.

In this paper, we propose a new conversion method that meets all of the above three conditions. This method first calculates the values of three scalar functions that evaluate the abovementioned three conditions. Pairs of triangular elements are then converted into quadrilateral elements in order of the sum of the values of the three functions.

2 Related Work

2.1 Node Generation Methods Using Particle Models

Many triangulation approaches first generate a set of nodes inside a given domain, and then connect them to form triangular elements. We previously proposed node generation methods that use particle models.

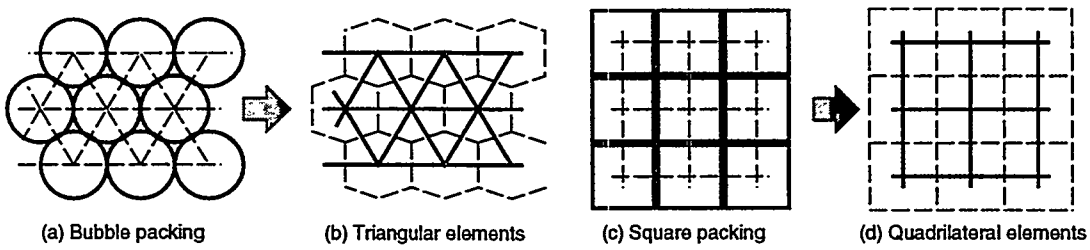


Figure 1: Voronoi diagram and generation of elements.

Shimada proposed the bubble mesh method [7], which tightly packs spherical objects that exert repulsive or attractive forces on each other. The packed spherical objects, called bubbles, form a pattern like the hexagonal pattern of a Voronoi diagram, and therefore well-shaped triangular elements can be generated by connecting the centers of these bubbles (see Figures 1 (a) and 1 (b)).

The method properly distributes nodes in complicated geometric domains, and the density of nodes can be adaptively controlled according to a scalar function representing the size of the bubbles. These characteristics allow the method to satisfy Requirements 1 and 2 if the elements are properly generated.

The method generates triangular meshes by connecting the centers of bubbles, using the Delaunay triangulation method. The triangular meshes are generated by coupling most of the triangular elements [6]. However, well-shaped

quadrilateral elements are not always generated by merging regular triangular elements. Moreover, such conversion of regular triangular meshes often causes topological irregularities.

Shimada, Liao, and Itoh proposed the square packing method [8], which tightly packs square objects instead of bubbles. The packed square objects form a pattern like square pattern of a Voronoi diagram, and well-shaped quadrilateral elements can be generated by connecting the centers of these squares (see Figures 1 (c) and 1 (d)).

The method properly distributes nodes in complicated domains, and the density of nodes can be adaptively controlled, as in the bubble mesh method. Moreover, the square packing method aligns nodes with a vector field given by a user. Figure 2 (a) shows an example of a vector field, while Figure 2 (b) shows an example of squares packed by the method. The figures demonstrate that the method distributes nodes along the given vector field, indicating that the square packing method can satisfy Requirement 3 if the quadrilateral elements are then properly generated.

The method generates triangular meshes by connecting the centers of squares, using the Delaunay triangulation method. Note that triangular elements generated by the square packing method are not regular but almost right-angled isosceles triangles (see Figure 2 (c)). When two such adjacent triangular elements that share their hypotenuses are coupled, they can be converted into a well-shaped quadrilateral element.

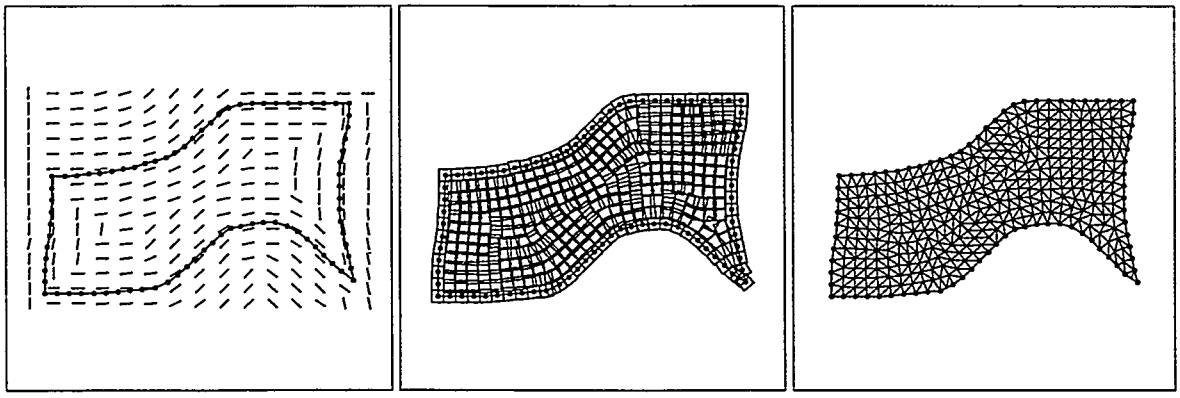


Figure 2: Example of (a) a vector field, (b) packed squares, and (c) a triangular mesh.

2.2 Quadrilateral Elements Generation

This section introduces some typical conventional quadrilateral mesh generation methods.

Mapped mesh

This method first manually divides a given region into a group of triangular or quadrilateral subdomains, and then divides the subdomains into small elements [9]. The method is not fully automated, and it is difficult to freely control the distribution of the size of elements. Therefore, it does not satisfy Requirements 1 and 2.

Grid-mapping

This method first maps a lattice grid inside a given region, and then connects nodes on the ends of the grid to nodes on the boundary of a given region [9]. Poor elements are often generated around the boundary, and it is difficult to freely control the distribution of the element size. Therefore, the method does not satisfy Requirements 2 and 4.

Advancing front

This method recursively generates elements along the boundary of an empty region, until elements occupy the entire region [10]. In typical implementations of the method, nodes are generated simultaneously with elements; however, a set of nodes previously generated by using one of the abovementioned particle-based methods [7] [8] can be used.

Conversion of Triangular Elements into Quadrilateral Elements

This method first pairs together adjacent triangular elements in some particular order, and converts them into quadrilateral elements by eliminating their shared edges (see Figure 3 (a)). When the process is complete, most of the triangular elements have been converted into quadrilateral elements, and finally a quad-dominant mesh that includes a small number of triangular elements is generated. If an all-quadrilateral mesh is necessary, the templates described in [6] are useful. They convert a triangular or quadrilateral element into half-size quadrilateral elements (see Figure 3 (b)).

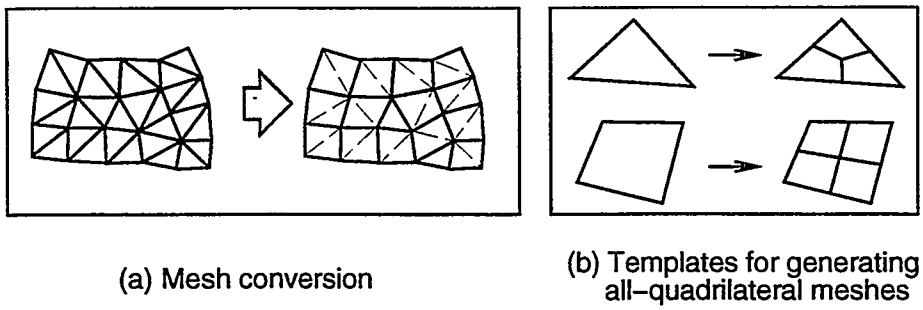


Figure 3: Conversion into quadrilateral mesh.

In the above conversion methods, the quality of quadrilateral meshes strongly depends on the order in which pairs of adjacent triangular elements are converted. The algorithms of the conventional methods can be categorized into the following three types:

1. Optimization of geometric irregularities. In this approach, the scalar function representing the shape of quadrangles generated from all possible pairs of adjacent triangular elements is first calculated, and the unprocessed triangle pairs are then converted in order of the value of this function [2] [3]. The approach satisfies Condition B, but does not meet the other two conditions.
2. Minimization of the number of isolated triangular elements. In this approach, the number of unprocessed adjacent triangular elements N_t for each triangular element are counted, and triangle pairs for which the value of N_t of at least one triangular element is 1 are then converted [4] [5]. The approach satisfies Condition C, but does not meet the other two conditions.
3. Belt-shaped decomposition of triangular meshes. In this approach, groups of triangular elements are first created according to the topological distance of each element from arbitrary marked nodes, and triangular elements in each group are then coupled [6]. The approach attempts to satisfy Conditions A and C, but it sometimes generates poor meshes when the grouping process does not work properly.

3 New Conversion Method with Directionality Control

In this section, we propose a new method for generating quadrilateral meshes by converting triangular meshes. Our implementation first applies node-generation methods such as the square packing method [8], and then connects them

by the Delaunay triangulation algorithm to form a triangular mesh. The new method then evaluates the quality of all the possible quadrangles that can be generated by merging pairs of adjacent triangular elements, and converts the pairs in order of the evaluation results.

Such a quadrilateral element generation procedure can be also realized by the advancing front methods. However, we think that the new conversion method has the following advantages:

1. Delaunay triangulation and mesh conversion are easier to implement, since the advancing front method has complicated procedures for managing the topology of empty regions.
2. The order of quadrilateral element generation is nearly optimized in the new conversion method, while the advancing front method always generates quadrilateral elements starting from the boundary of a given region.

3.1 Scalar functions for three conditions

The new mesh conversion method uses scalar functions $F_A(P_{i,j})$, $F_B(P_{i,j})$, and $F_C(P_{i,j})$, to evaluate the above-mentioned three conditions. Here, $P_{i,j}$ denotes a pair of adjacent triangular elements T_i and T_j that share an edge of a triangle. The method first calculates the values of these functions for all possible pairs of adjacent triangular elements. The pairs are then converted in order of the sum of the values of the three scalar functions.

The method uses a vector field to represent the desired mesh directionality, as in the square packing method [8].

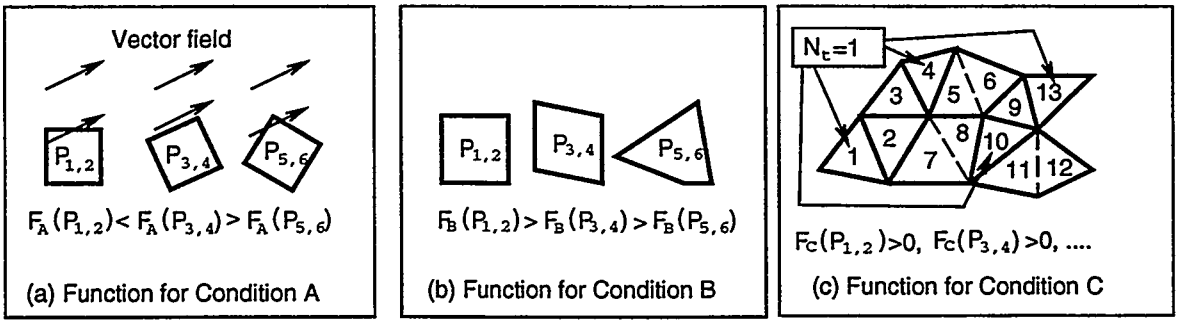


Figure 4: Scalar functions for three conditions.

Scalar function for generating elements aligned along the vector field

The function $F_A(P_{i,j})$ is used to evaluate Condition A. We implemented it as the following equation:

$$F_A(P_{i,j}) = \Sigma_{i=1}^{4} max\{(E_i \cdot V), (E_i \cdot V_o)\}, \qquad V_o = N \times V \tag{1}$$

Here, E_i denotes the unit vector of an edge of the quadrilateral elements generated from $P_{i,j}$, while V denotes the unit alignment vector according to the given vector field at the center of the quadrilateral element, N denotes the unit normal vector of the quadrilateral element, and V_o denotes the outer product of N and V. The function returns a higher value, when the edges and the vector are parallel or perpendicular (see Figure 4 (a)). Many quadrilateral elements aligned along the vector field are generated by converting pairs of triangular elements in order of the value of F_A.

Scalar function for generating well-shaped quadrilateral elements

The function $F_B(P_{i,j})$ is used to evaluate Condition B. We implemented it as the following equation:

$$F_B(P_{i,j}) = 4.0 - \Sigma_{i=1}^{4}(E_i \cdot E_{(i+1)\%4}) \quad (2)$$

Here, E_i denotes the unit vector of an edge of the quadrilateral elements generated from $P_{i,j}$. It returns a higher value when the angles of the vertices of the quadrilateral element generated from $P_{i,j}$ are almost right-angled (see Figure 4 (b)). Many well-shaped quadrilateral elements are generated by converting pairs of triangular elements in order of the value of F_B.

Scalar function for minimizing the number of isolated triangular elements

The function $F_C(P_{i,j})$ is used for Condition C. It returns a positive value only when the only unprocessed adjacent triangular element of T_i is T_j, or the only unprocessed adjacent triangular element of T_j is T_i; otherwise, it returns zero. When $P_{i,j}$ is not converted into a quadrilateral element but the value of F_C is positive, it means that an isolated triangular element is to be generated. The number of isolated triangular elements is reduced by converting pairs of triangular elements in order of the value of F_C. We implemented it as a constant value.

Figure 4 (c) shows examples of the values of this function. In the mesh, triangular elements T_5 and T_6 have already been coupled and converted. Similarly, T_7 and T_8, and T_{11} and T_{12} have been coupled and converted. T_1, T_4, T_9, and T_{13} have only one unprocessed adjacent triangular element, and therefore the values of $F_C(P_{1,2})$, $F_C(P_{3,4})$, $F_C(P_{9,10})$, and $F_C(P_{9,13})$ become positive.

3.2 Procedure for converting triangular elements

The procedure for converting triangular elements into quadrilateral elements in the new method is as follows. Here a, b, and c are coefficients representing the importance of Conditions A, B, and C.

1. Calculate the value of $V_1 = aF_A + bF_B$ for each pair of adjacent triangular elements. Pairs that generate well-shaped and well-aligned quadrangles have higher values.
2. Sort the triangle pairs according to the value of V_1, and register them in a list L_1.
3. Calculate the value of F_C for each pair of adjacent triangular elements. If F_C is positive, register the pair in a list L_2, and calculate the value of $V_2 = aF_A + bF_B + cF_C$. The latter value is higher than that of V_1, since the pairing should place high priority on not causing isolated triangular elements.
4. Extract from L_1 the triangle pair that has the highest value V_{1max}. Similarly, extract from L_2 the triangle pair that has the highest value V_{2max}. Compare the values V_{1max} and V_{2max}, and select the triangle pair P that has the highest value.
5. If at least one of two triangular elements of P, T_i and T_j, has been marked,
   - (a) Eliminate P from L_1 or L_2.
   - (b) Go to 4.
6. If both T_i and T_j are unmarked,
   - (a) Convert P into a quadrilateral element.
   - (b) Eliminate P from L_1 or L_2.
   - (c) Mark T_i and T_j.
   - (d) Extract unmarked triangular elements T_n from adjacent triangular elements of T_i and T_j.
   - (e) Calculate the values of F_C for triangle pairs that include T_n, since the number of unmarked adjacent triangular elements of T_n may become 1 at that time.

(f) If there are any triangle pairs for which the value of F_C is positive, register them in L_2, and calculate the value of V_2.

(g) Go to 4.

7. Repeat 4, 5, and 6 until both L_1 and L_2 become empty.

4 Results

The new conversion method was implemented and executed on an IBM PowerStation RS/6000 Model 42T (AIX 4.1.4). Some results are given in this section.

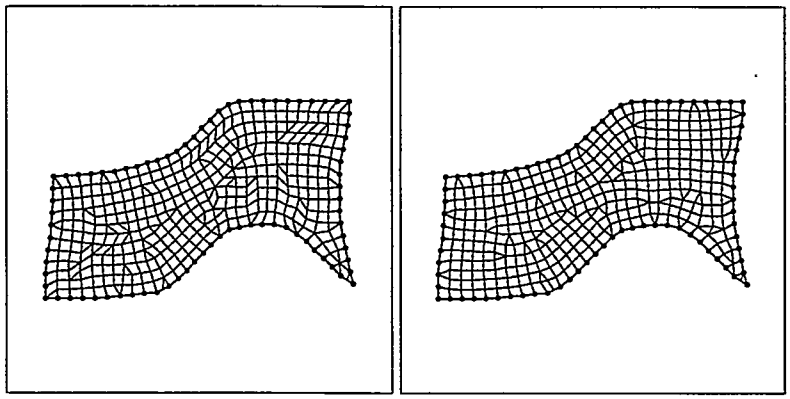


Figure 5: Quad-dominant meshes generated by conventional and new methods.

Table 1: Comparison between conventional and new methods.

	Computation time (sec.)	Geometric irregularity	Number of triangles
Conventional method	0.09	3.106304	47
New method	0.14	2.698476	43

Figure 5 shows the meshes generated by our conventional method [6] and the new method. In the new method, the values of the coefficients are $a = 0.5$, $b = 1.0$ and $c = 0.3$. The meshes were obtained by converting the same triangular mesh generated by the square packing method with a vector field, as shown in Figure 2. Table 1 shows the computation times of the two methods and the quality of the generated meshes. These results show that the conventional method does not always generate well-shaped and well-aligned meshes; however, this problem is dramatically reduced by the new method.

Here, the values of geometric irregularity are calculated as the ratio of the circumscribed and inscribed circles of elements. Generally, well-shaped meshes have lower values for the geometric irregularity.

Figures 6 and 7 show the packed squares, triangular meshes, and quadrilateral meshes generated by the new method, when the values of the coefficients are $a = 0.5$, $b = 1.0$, and $c = 0.3$. The packed squares were generated by using a scalar field representing the distribution of the sizes of elements. The results show that the method generates well-shaped and well-aligned meshes in which the sizes of elements are adaptively controlled.

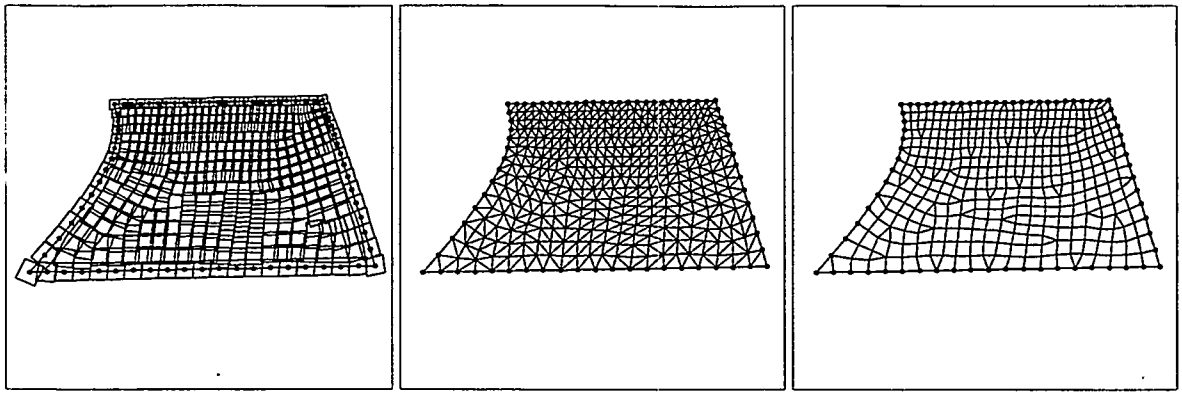


Figure 6: Quad-dominant meshes with adaptively controlled sizes of elements (1).

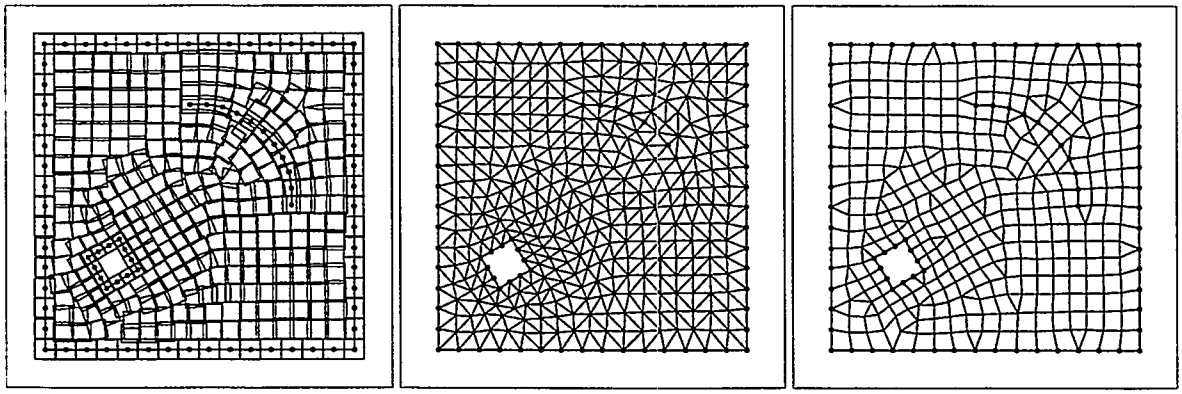


Figure 7: Quad-dominant meshes with adaptively controlled sizes of elements (2).

Figure 8 shows the meshes generated by setting the value of the coefficient a to 0.0, 0.15, and 0.5, and the values of the other coefficients to $b = 1.0$ and $c = 0.3$. Figure 9 shows the meshes generated by setting the value of the coefficient c to 0.0, 0.55, and 1.0, and the values of the other coefficients to $a = 0.5$ and $b = 1.0$. The meshes were obtained by converting the same triangular mesh generated by the square packing method with a vector field, as shown in Figure 2. Table 2 shows the quality of the quadrilateral meshes. These results show that the values of the coefficients should be carefully adjusted according to users' requirements.

Finally, we observed how changing the number of nodes inside the same region affected the computation time for the new method. Table 3 shows the computation time for the Delaunay triangulation and mesh conversion processes. These results show that the computation time is more than proportional to the number of nodes. Actually, it can be regarded as $O(n \log n)$, where n denotes the number of nodes, since the mesh conversion method includes the sorting process. If the time needs to be reduced, it is useful to classify pairs of triangular elements according to the evaluation, instead of using the sorting process. The order of element conversion is slightly changed by using the classification process; however, in many cases the results are acceptable.

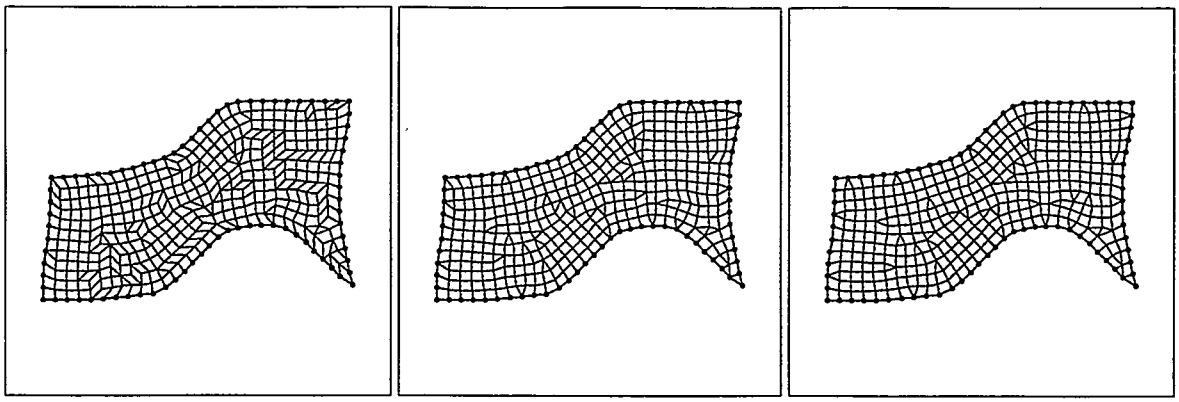


Figure 8: Quad-dominant meshes obtained by setting the value of the coefficient a to 0.0, 0.3, and 1.0, b to 1.0, and c to 0.3.

Table 2: Comparison by changing the values of coefficients.

a	b	c	Geometric irregularity	Number of triangles
0.0	1.0	0.3	3.901622	31
0.3	1.0	0.3	2.760574	41
1.0	1.0	0.3	2.697353	45
0.5	1.0	0.0	2.697353	45
0.5	1.0	0.55	2.906827	35
0.5	1.0	1.0	3.111585	31

5 Conclusion

We have proposed a new method for converting triangular meshes into quadrilateral meshes. The method uses three scalar functions for evaluating directionality, the shape of quadrilateral elements, and the reduction in the number of isolated triangular elements. In this way, it generates well-aligned and well-shaped quad-dominant meshes with small numbers of triangular elements.

References

[1] Ho-Le K., *Finite Element Mesh Generation Method: a Review and Classification,* Computer Aided Design, Vol. 20, No. 1, pp. 27-38, 1988.

[2] Lo S. H., *Generating Quadrilateral Elements on Plane and over Curved Surfaces,* Computer and Structures, Vol. 31, No. 3, pp. 421-426, 1989.

[3] Borouchaki H., Frey P. J., and George P. L., *Unstructured Triangle-Quadrilateral Mesh Generation. Application to Surface Meshing,* Proceedings of 5th International Meshing Roundtable, pp. 229-242, 1996.

[4] Heighway E. A., *A Mesh Generator for-Automatically Subdividing Irregular Polygons into Quadrilaterals,* IEEE Transactions on Magnetics, Mag-19, pp. 2535-2538, 1983.

[5] Johnston B. P., Sullivan J. M., and Kwasnik A., *Automatic Conversion of Triangular Finite Element Meshes to Quadrilateral Elements,* International Journal for Numerical Methods in Engineering, Vol. 31, pp. 67-84, 1991.

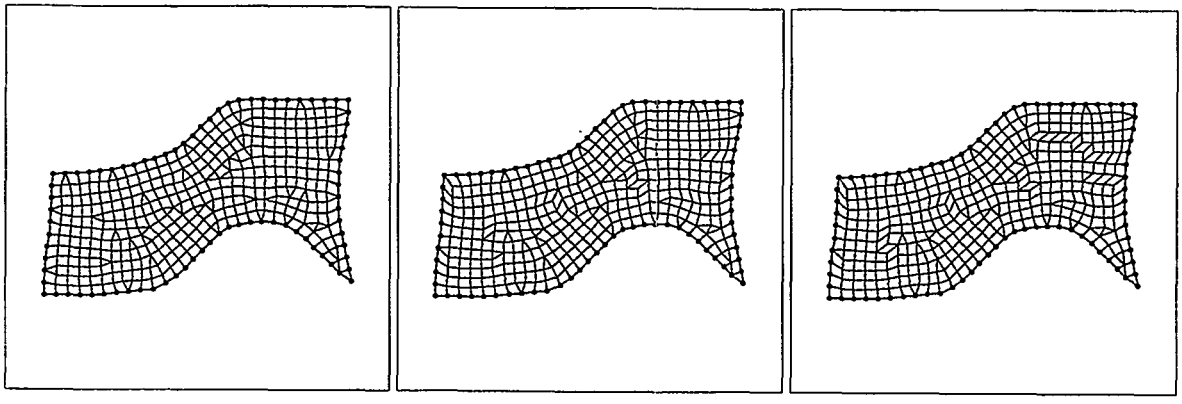


Figure 9: Quad-dominant meshes obtained by setting the value of the coefficient c to 0.0, 0.55, and 1.0, a to 0.5, and b to 1.0.

Table 3: Computation time.

Number of nodes	64	196	357	1228	2003
Delaunay triangulation (sec.)	0.02	0.06	0.12	0.59	1.87
mesh conversion (sec.)	0.01	0.06	0.20	1.11	4.31

[6] Shimada K., and Itoh T., *Automated Conversion of 2D Triangular Mesh into Quadrilateral Mesh,* Proceedings of International Conference on Computational Engineering Science '95, pp.350-355, 1995.

[7] Shimada K., *Physically-Based Mesh Generation:Automated Triangulation of Surfaces and Volumes via Bubble Packing,* Dissertation, Massachusetts Institute of Technology, 1993.

[8] Shimada K., Liao J., and Itoh T., *Quadrilateral Mesh Generation via Close Packing of Square Cells,* Proceedings of 7th International Meshing Roundtable.

[9] Ho-Le K., *Finite Element Mesh Generation Method: a Review and Classification,* Computer Aided Design, Vol. 20, No. 1, pp. 27-38, 1988.

[10] Blacker T. D., *Paving: A New Approach to Automated Quadrilateral Mesh Generation,* International Journal for Numerical Methods in Engineering, Vol. 32, pp. 811-847, 1991.

A Global Minimization-Based, Automatic Quadrilateral Meshing Algorithm

Paul Wolfenbarger (pwolfen@arc.unm.edu)[1]
Joseph Jung (jjung@sandia.gov) [2]
Clark R. Dohrmann (crdohrm@sandia.gov)[2]
Walter R. Witkowski (wrwitko@sandia.gov) [2]
Malcolm J. Panthaki(panthaki@arc.unm.edu)[3]
Walter H. Gerstle (gerstle@unm.edu)[1]

Abstract: *A novel method is presented for automatically generating quadrilateral meshes on arbitrary two-dimensional domains. Global minimization of a potential function governs mesh formation and characteristics. Comprised of several terms, the potential function distributes the elements throughout the domain and aligns the edges of the elements to form valid connectivities. If there are any remaining unlinked element edges, the local connectivity is examined and a "hole elimination" algorithm is applied that successively finds alternative connectivities. Unlinked edges, representing holes in the mesh, are moved to either coalesce, or to a boundary. The components of the potential, the minimization procedure, and the connectivity refinement algorithm are presented. The method shows promise for extension to automatic three-dimensional hexahedral meshing.*

Initial conditions required to ensure mesh closure include an even number of elements on the boundary and a closed boundary. The desired mesh characteristics are programmed into the algorithm. A Poisson's solution scheme is utilized to generate a better initial placement, density, size and orientation of elements, leading to faster and more robust mesh closure. A number of example geometries have been meshed.

Keywords: Automatic quadrilateral meshing, global minimization-based meshing, mesh, computational mechanics.

1. Introduction and Motivation

The use of finite element codes is accelerating in industry, science, and academia [10]. Driven by cheaper computing, shorter design cycles, and costlier laboratory and destructive testing, this acceleration is taxing the ability of analysts to keep pace. In an attempt to shorten the design cycle, and to allow the analyst to focus on the results, many codes are turning to automated generation of meshes. In fact many modern designs require meshes which if generated by hand would take longer than the entire allotted design cycle [19, 13]. Evolutionary problems (where the model changes as the analysis progresses) and adaptive mesh refinement procedures also require robust, reliable and repeatable automatic mesh generation algorithms.

Many problems plague current methods for generating unstructured meshes. The most often used meshes are triangular/tetrahedral and quadrilateral/hexahedral meshes. Triangular/tetrahedral meshing is well understood, but many analysts dislike triangular/tetrahedral meshes. This dislike is due to the fact that triangular/tetrahedral meshes are inherently stiffer than quadrilateral/hexahedral meshes. This forces the use of more, or higher order, elements to produce the same level of accuracy [1, 12]. No reliable, widely

[1]Department of Civil Engineering, University of New Mexico (UNM), Albuquerque, NM 87131
[2]Sandia National Laboratories, Albuquerque, NM 87185 (The portion of this work performed at Sandia National Laboratories was supported by the United States Department of Energy under Contract Number DE-AC04-94AL85000.)
[3]Independent Engineering Software Consultant and Senior Research Engineer at the Albuquerque High Performance Computing Center, Albuquerque, NM 87131

accepted method of automatically generating good quality, unstructured, quadrilateral or hexahedral meshes has been adopted. The current state of the art in the field of quadrilateral meshing is the Paving algorithm which has some problems closing local discontinuities such as two elements needing to attach to five elements or two elements overlapping in space. These discontinuities occur when multiple advancing fronts of elements converge at "unacceptable" angles and densities [20]. Several other reliable but less accepted or less automatic algorithms are available [2].

In an attempt to find a new approach to these problems, Sandia National Labs and the University of New Mexico have developed a new algorithm. This algorithm is based on using a 2D particle model to govern the movement of each element and then forming links between particles to create a completed mesh. By creating a generalized functional that relates each quadrilateral element directly to the geometry and to the other elements, the global problem is discretized without a loss of generality. Using methods borrowed from physics, chemistry, and other scientific computational methods, a "mostly connected" mesh is created. Topologically-based connectivity improvement techniques are then utilized to close the remainder of the mesh. While mesh refinement is sometimes useful to alleviate the problem of acute angles, no smoothing is required as the global minimization algorithm uses a technique very similar to existing mesh smoothing algorithms (known as Relaxation Smoothing [8]) to smooth the mesh during its creation.

During the course of developing this method, it was necessary to create an infrastructure which would allow the algorithm to be easily studied and debugged, and allow flexibility in the data structure of the mesh at various stages of the mesh creation. To this end, the meshing tool was implemented in the CoMeT (Computational Mechanics Toolkit) framework which is described in detail by Panthaki et. al. [15]. (Additional information on the CoMeT framework can be obtained at *http://www.arc.unm.edu/CoMeT*). A new meshing subsystem was added to CoMeT to facilitate development of this and other mesh generation schemes. CoMeT provides software utilities that allow the algorithm developer to graphically view the state of the mesh during execution. CoMeT also provides an interpreted, extensible language that is used to modify, test and debug the meshing algorithm. Changes can be made to the flow of the algorithm, often without the need to modify any of the underlying C++ code. These changes allow the algorithm developer to interactively modify and test various algorithmic parameters including the actual flow of the algorithm and proved useful during the developmental stages of this meshing tool. The graphical debugging tools and animation tools, of course, provided insight into the runtime behavior of the meshing algorithm.

While still a prototype tool, this meshing algorithm is a considerable step toward meeting the goal of robust, fully-automatic all quadrilateral mesh generation over arbitrary two-dimensional domains. The essence of the algorithm is promising for extension to fully-automatic three-dimensional hexahedral meshing. Furthermore, the mesh subsystem developed in CoMeT will make future research on advanced meshing algorithms much simpler. So far, all the research has been targeted towards achieving robust closure of meshes (resulting in a topologically-valid quadrilateral mesh) using any arbitrary, user-specified number of elements over any arbitrary two-dimensional domain. Little emphasis has been given to either the final quality of the resulting mesh or the computational efficiency of the algorithm. The next phase of the research will address these issues as their resolution is critical to the eventual adoption of this algorithm into analysts' tool box of meshing tools.

2. Global Minimization-Based Quadrilateral Meshing Algorithm

A functional is used to drive a dynamic simulation in a manner similar to computational chemistry and molecular biology simulations [4, 11, 6]. Use of a functional to represent desirable relationships between elements, the geometry and other elements makes the method extensible to 3D or to generalized surface meshing. Only the functional needs to be changed to modify the bulk of the program's behavior. An implicit goal of the method is to minimize the user input required to create a topologically-valid, high quality mesh that captures the geometry of the domain as well as it can, using the criteria specified by the user.

2.1. Overview of the Algorithm

In creating a meshing tool that provides a global approach to all the various stages of mesh generation, all data are distributed at the lowest possible object level. All particles have the same level of knowledge about their surrounding conditions and act as independently as possible. A number of methods were borrowed from particle physics, and molecular dynamics to control element movement and linking, with the final mesh being completed using one of several methods developed for coalescing "holes", (see Figure 1), or removing them at the mesh boundary.

After establishing the 2D geometry, particles are placed using one of several user-prescribed methods as described below. At present the main constraint on particle placement is that a layer of elements must be attached to the domain boundary. Also the system is designed to allow the particle size to be defined according to user criteria. The initial size for a particle is maintained throughout the algorithm. The functional is then applied to control the movement of the particles in several phases.

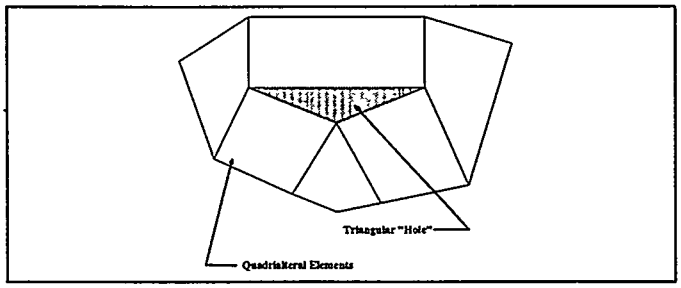


Figure 1. Detail of a Mesh Which Shows a Triangular Hole Between Quadrilateral Elements.

- Initially a boundary mesh of particles is attached to each boundary of the domain, and interior particles are placed as directed by the user (see Section 2.2).
- The second phase of the simulation distributes the particles throughout the domain to be meshed, using simple particle-to-particle and particle-to-boundary repulsive forces as described in Section 2.3.2. The simulation then runs until an equilibrium (within a user-specified tolerance) position is reached. In cases where the initial input is known to have good gradation(e.g., Poisson placement [9]), this step may be omitted.
- For the third phase, each particle finds the best match it can for each arm end (see Section 2.3.3) and creates a link. In a similar fashion to the way atoms form covalent bonds to form a molecule, each arm then has an attractive force based on the distance to its bond partner (see Section 2.3.4). Each link also creates a force to align each particle as well as possible with the other particles to which the particle is linked (see Section 2.3.4). When the original input is not well graded, this phase is continued until the system is deemed to be in an approximate state of force equilibrium.
- In the final phase, the algorithm locates the unlinked particle arms and determines the best closure states of the "holes" they belong to (see Section 2.4). These holes are then coalesced or moved to the boundary to close the mesh.

The result is a closed, topologically valid quadrilateral mesh containing approximately the distribution of elements specified by the user. Note that the final phase of the algorithm may result in the addition or removal of a few elements on the boundary - future improvements will work to eliminate this aspect of the algorithm as it interferes with the need to maintain mesh compatibility across shared domain boundaries.

2.2. Initial Particle Placement

As with any nonlinear problem, the closer the initial guess is to the final solution, the shorter the time to convergence. In this research there are also multiple solutions and it has been found that better initial conditions also favor better finished meshes.

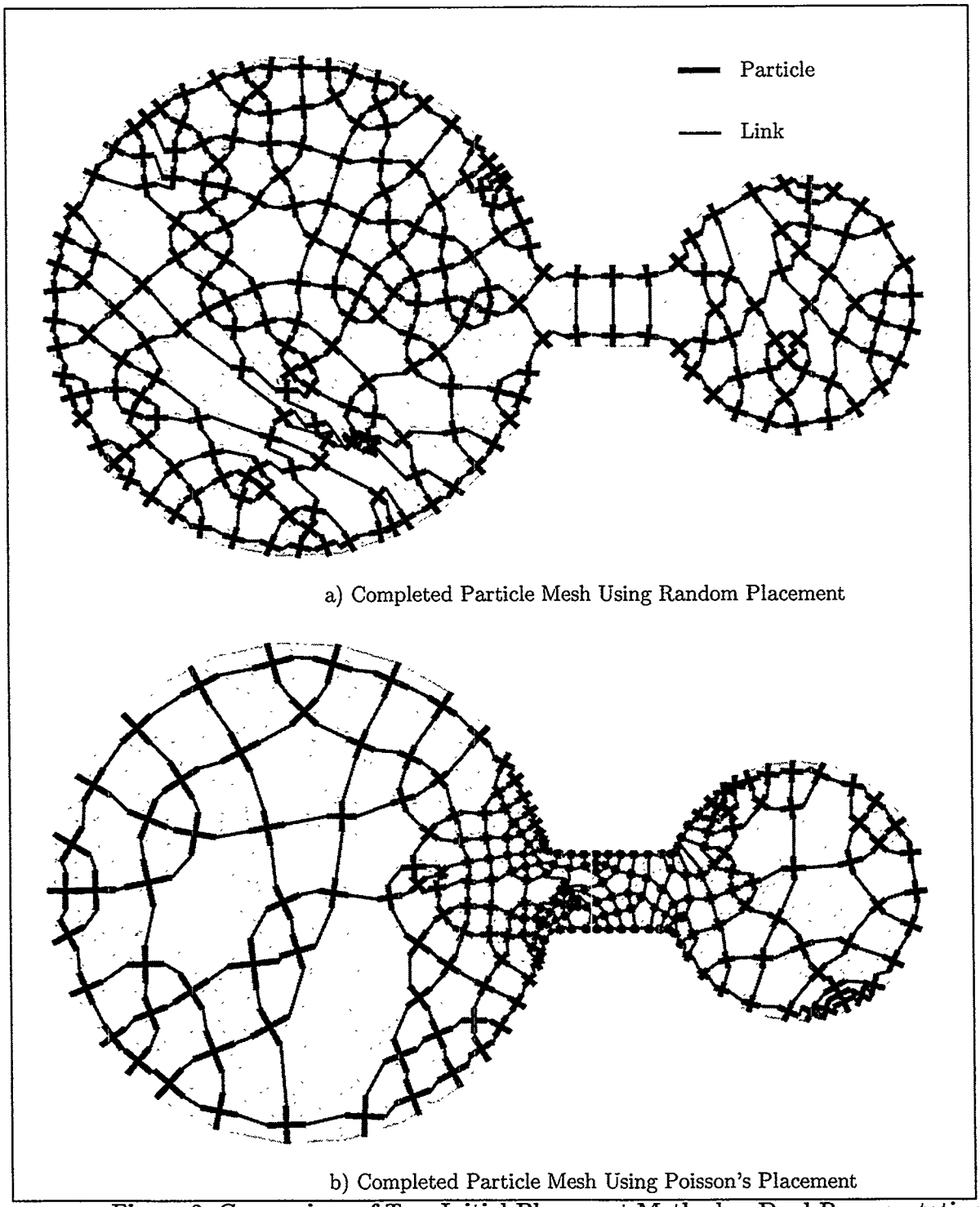
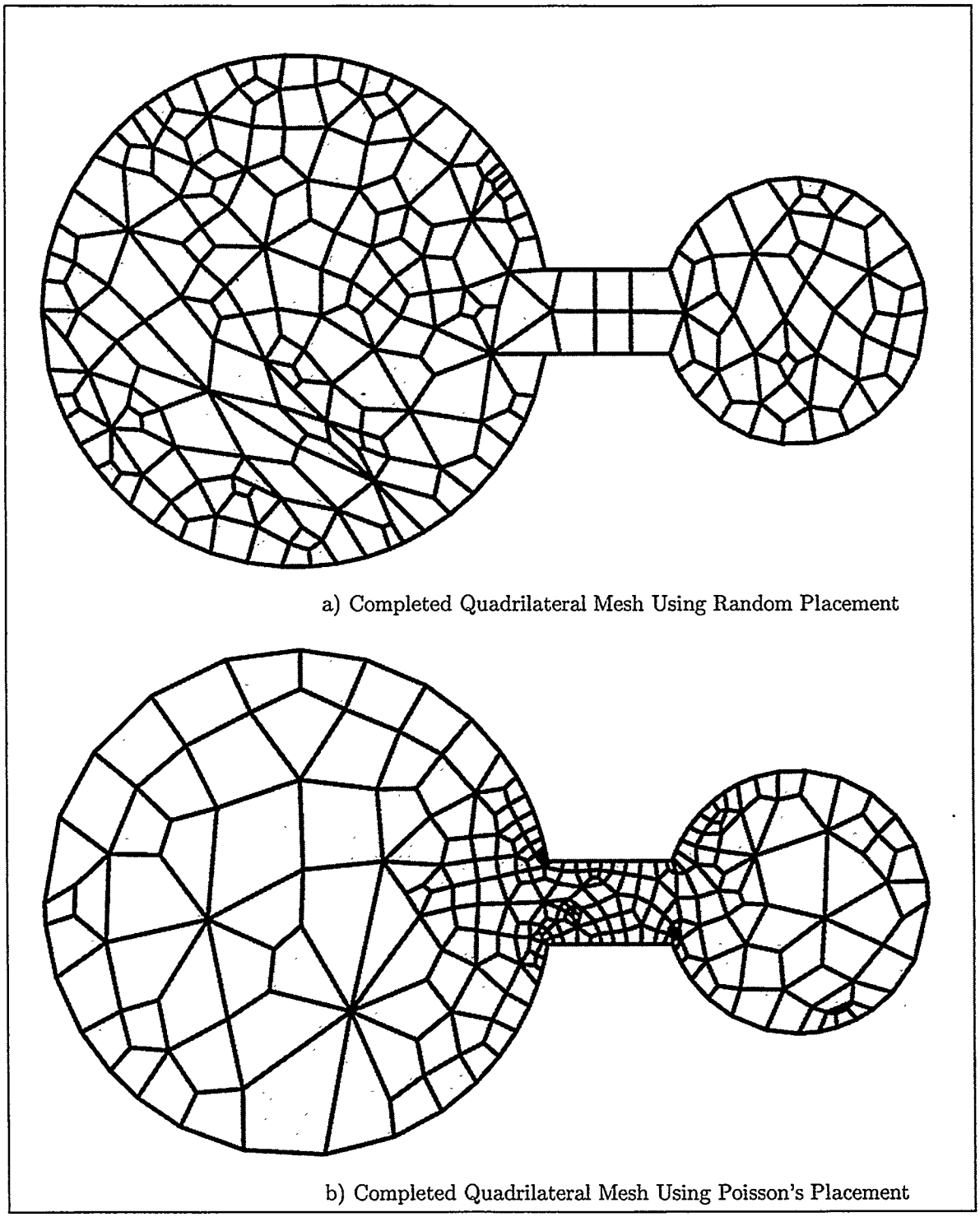


Figure 2. Comparison of Two Initial Placement Methods - Dual Representation



a) Completed Quadrilateral Mesh Using Random Placement

b) Completed Quadrilateral Mesh Using Poisson's Placement

Figure 3. Comparison of Two Initial Placement Methods - Quadrilateral Mesh Representation

As an example, two meshes of a dog-bone shape are shown in Figure 2. The top series was created using an initially random placement. This method simply places a predefined number of particles evenly spaced on the boundary and then scatters the remainder randomly throughout the interior. In this case 235 particles were used to mesh the figure with 81 of them on the boundary. The bottom figure is a comparable mesh created using a placement technique called Poisson's placement method based on a local feature size, which is calculated using a boundary integral solution [9]. The Poisson's placement method created a mesh which was smoother around the corners and had a much better gradation of element size. Also, due to better initial conditions (distribution, size and orientation of the particles), the Poisson's mesh took about a third of the time to close when compared to the time it took to close the mesh using the random placement method. In both figures there are clumps of particles which represent areas that did not run to complete equilibrium or where a hole was driven to the boundary. Large holes caused by coalescing holes which do not complement each other well such as the one shown in Figure 2(b) also occur. Figure 3 shows the resulting meshes for the same figures. Inspection of these meshes shows that the large open areas produce the worst type of elements with acute included angles and a high number of elements sharing a single node. Efforts are underway to eliminate these two related problems.

Another placement method used consists of placing all of the interior particles in a box smaller than the area to be meshed. Other methods have been considered including partial results of other meshing tools in the same manner as the Poisson placement method.

At present, the only constraint on initial particle placement is that a layer of particles must first be attached to the boundary. This helps to constrain interior particles from leaving the domain during the minimization phase. The functional is then applied to control the movement of the interior particles in the domain, with the goal of distributing the particles appropriately. Any method can be used for initial particle placement over the domain so long as this constraint is met.

2.3. Functional Definition

The functional, Φ, is a set of relations between the particles and their environment which govern how the particles behave. To define the functional the particle is first defined and then the parts of the functional $\Phi = ParticleRepulsion + ArmAttraction + ArmAlignment + Damping$ are examined. Each part of the functional is described in a separate section. The complete functional is

$$
\begin{aligned}
\Phi \quad = \sum_{i=1}^{n} \{ & \\
& \sum_{x=1}^{n} C_R \left(\frac{l_I}{|\overrightarrow{C_i C_x}|} \right)^2 + \sum_{X=1}^{m} C_R \left(\frac{l_I}{|\overrightarrow{C_i B_X}|} \right)^2 + && (\textit{Particle Repulsion} - \textit{see Section } 2.3.2.) \\
& C_A |\overrightarrow{A_{ik} A_{jl}}|^2 + && (\textit{Arm Attraction} - \textit{see Section } 2.3.4.) \\
& C_{Rot} (\overrightarrow{A_{ik} A_{jl}} \cdot \overrightarrow{C_{ik} A_{I(k-1)}})^2 \; \} && (\textit{Arm Alignment} - \textit{see Section } 2.3.5.)
\end{aligned}
\tag{1}
$$

and viscous Damping is defined as

$$
Damping \quad = \quad \sum_{i=1}^{n} \{ C_{D_{trans}} V_i + C_{D_{rot}} \omega_i \} \qquad (\textit{Damping} - \textit{see Section } 2.3.6.)
\tag{2}
$$

This functional represents the differential equation of motion for the particle which the program must solve. The entire algorithm can be thought of as simply a way to minimize the functional as a scalar representation

of the state of the particles. Note that the form of the functional is a fairly simple one which was chosen for proof of concept and development. For instance, the repulsive potential is set as a basic inverse power relation but a Lennard-Jones potential could instead be used in its place [6, 3]. The fact that the algorithm's driving differential equation is simple to change and is loosely coupled to the system insures mesh closure, making the algorithm provably robust. In subsequent sections, we will examine how the particle moves in response to the functional and what this means in terms of reproducibility.

2.3.1. Particle Description

Figure 4 shows the basic particle used in the simulation as a dual representation of a quadrilateral element. A particle consists of a location ($C_i(x,y)$), size (l_i), and orientation (θ) along with dynamic information used in the simulation.

For each particle we define five positions. C_k for the center and four A_{kn} for the ends of each particle arm. This allows us to define each arm as a vector which will be useful in the functional. Since the rigid particle is potentially rotating, we do not actually store all of these positions but instead store position C_k and angle θ defining the particle's rotation from the horizontal.

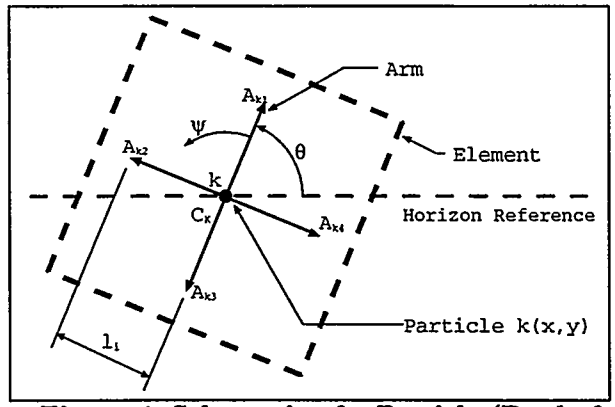


Figure 4. Schematic of a Particle (Dual of Quadrilateral Element)

ψ is the relative angle from arm number one to the arm for which we need information. Note that the particle arms are fixed at a 90° relationship to each other, so that no storage of this information is required. As links can be formed at any angle, it was found that allowing a degree of freedom in ψ added complexity and instability to the solution.

The size of a particle is defined as the distance $|\overrightarrow{C_k A_{kn}}|$ where all arms will have the same length. This information is enough to allow the rapid determination of all the required vectors at a given point in time but does not require computing all of them every time C_k, θ, or the size changes.

The dual nature of our particle with the eventual element which will replace it is also shown in Figure 4. Although we have drawn a regular quadrilateral with no distortion, there is no such constraint on the finished mesh (see Figure 5). This can result in some elements which are severely distorted. In order to limit this as much as possible, the tool tries to maintain nodes that are shared by only four elements. While not a panacea, this does help to keep the number of vertices with small included angles to a minimum. Mesh quality could also be improved by adding another term to the functional, using a simulated annealing [6] approach to get a better initial set of links and improved link finding methods. In fact, almost any quality measure can be incorporated into the algorithm at an early stage if it can be well defined in terms of particle-to-particle and particle-to-boundary interactions.

2.3.2. Repulsive Potential

Each particle sees potential from all of the other particles and boundaries in line of sight (LOS). In Figure 6 particle "I" (C_i) has a direct line of sight to particles "h" (C_h), "j" (C_j), "k" (C_k), and "l" (C_l). The line from the center of particle "I" to particles "o" (C_o), and "p" (C_p) are blocked by intersecting with boundary segment "A" (B_A). For each of the directly seen particles, particle "I" will have a functional contribution of $C_R \left(\frac{l_I}{|\overrightarrow{C_i C_x}|} \right)^2$, where C_R is the repulsive constant defined by the user, $|\overrightarrow{C_i C_x}|$ is the distance between

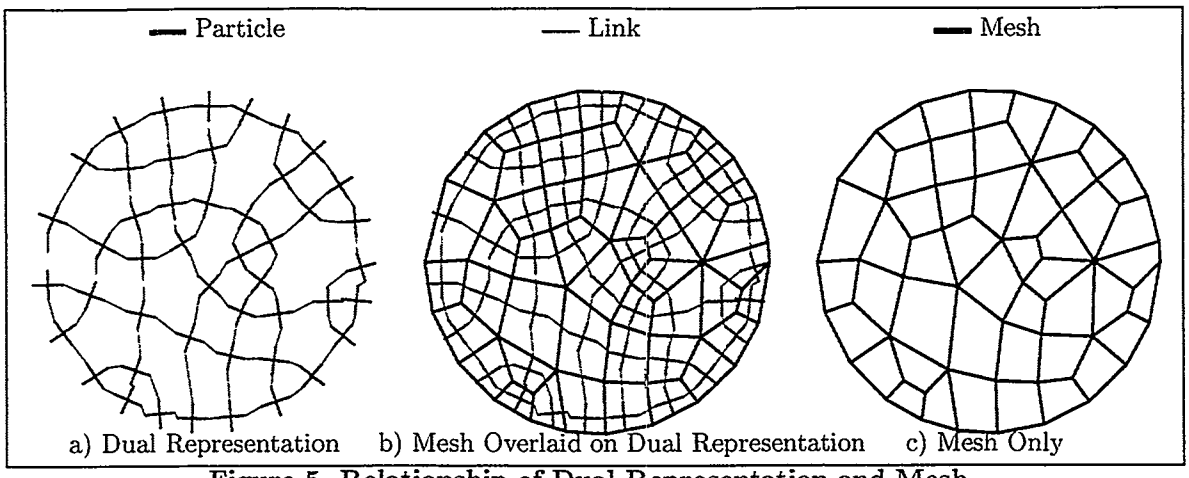


Figure 5. Relationship of Dual Representation and Mesh

centers of particles, l_I is the nominal size of the particle "I". Since we want to keep track of the forces rather than the functional, we actually use the first derivative with respect to position of all the functional formulas in the program. This requires less computation because it removes the exponential term. A good deal of work was done to balance the repulsive coefficients (C_R) with the attractive (C_A and C_{Rot}) terms in order to have a stable simulation. The simulation is very stable after the links are formed and the forces are mostly in balance, but in the initial stages a bad combination of particle placements and constants can easily send a particle completely out of the domain. The system does not contain any functional contribution designed specifically to handle this. The problem of particles leaving the simulation domain is dealt with by adapting the time step to a small enough size to allow the particle to remain inside the domain while the functional acts to counteract the excess velocity.

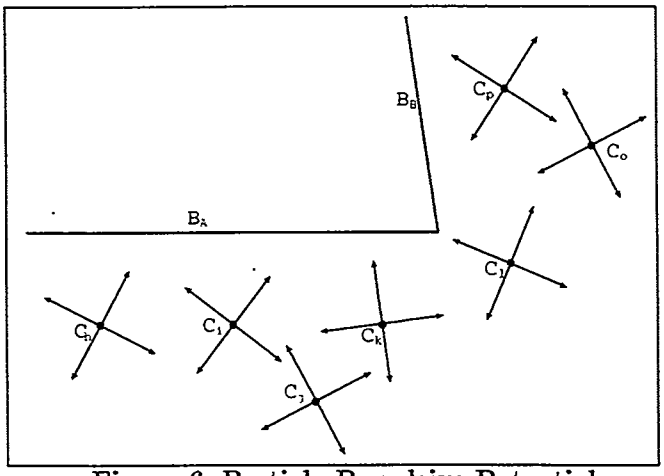


Figure 6. Particle Repulsive Potential

Particle "I" will also see a contribution from boundary segment "A" since a line perpendicular to it will not cross any other boundary segments. On the other hand, no contribution is seen from boundary segment "B" (B_B), since a perpendicular line to this segment does not intersect the segment itself. Again, the functional contribution is of the form, $C_R \left(\frac{l_I}{|\overrightarrow{C_i B_X}|} \right)^2$, where all variables are the same as discussed before except that the distance $|\overrightarrow{C_i B_X}|$ is defined as the shortest (perpendicular) distance to the boundary segment "X".

Note that at each time step each particle must check $n-1$ other particles, either to determine the functional contribution or to determine that no functional contribution is required due to the LOS rules. If well implemented, this can be brought from $O(n^2)$ to $O(\frac{n(n-1)}{2})$. This small improvement still leaves determining the repulsive potential as the main performance bottleneck. Several possible performance enhancements are under discussion, none of which have yet been implemented.

2.3.3. Link Creation

Each link between two particle arms represents a shared edge between two quadrilateral elements in the 2D mesh. If an element edge lies on the domain boundary, that edge is "linked" to the geometric boundary rather than to another particle's arm.

Requiring that each arm be linked to one and only one other arm or boundary ensures that upon completion a finished mesh exists with no ambiguous connections. To find these connections a simple method was developed which finds the closest arm end in the direction that the arm points to within a predefined angle (see figure 7). The system can also set the maximum distance at which a link will be made in terms of a multiple of the particle size. Too small a distance will result in no links at all being formed, and too large a distance will result in extra computation time. Links cannot be formed if the two ends are not within LOS as described in Section 2.3.2.

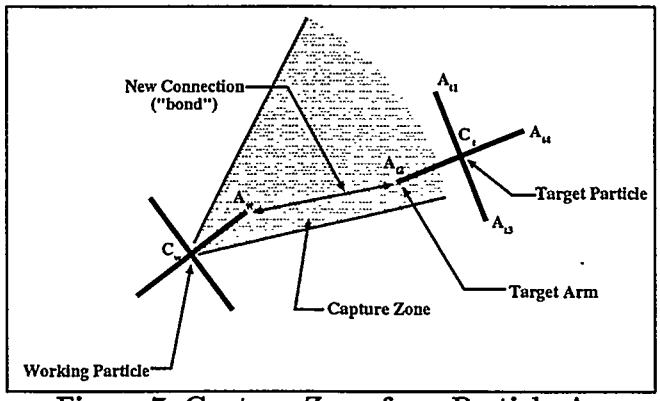


Figure 7. Capture Zone for a Particle Arm

In contrast to a molecular dynamics simulation in which all molecules feel alignment forces from all other molecules, [4] each particle arm in the simulation feels alignment forces from only the one particle arm to which it is linked. This method is used in simulated annealing where the randomness of the molecules movement is slowly reduced leaving all of the molecules closely aligned with its neighboring molecules [3]. Some thought has been given to using a generalized method until the particle links are found in order to have better aligned particles, resulting in more suitable links being made.

For the 2D case the existing method functions quite well. The 3D version may require a more sophisticated solution. Several refinements have been suggested to improve the way that links are selected between multiple candidates in the search area or "Capture Zone." The current implementation takes the best dot product match between particle arms on the working and target particles, $[min(\overrightarrow{C_w A_w} \cdot \overrightarrow{C_t A_{tk}})]$ with the target particle simply being the closest particle in the capture zone. As an alternative the best dot product match of all of the particle arms in the capture zone might be used. Another alternative is to assign each target arm a weight based on a fuzzy evaluation of the arm's distance, deviation from the arm angle, and dot product match [16].

2.3.4. Attractive Potential

Each particle has up to four links (one per arm). These links can be attached either to other particle's arms or to a boundary as discussed in Section 2.3.3. A particle I with arm k attached to arm j of particle l will see a functional contribution of $C_A|\overrightarrow{A_{ik}A_{jl}}|^2$ (see Figure 8). Since the force must act through the center of the particle, an additional rotational contribution to the functional is present in the alignment potential as discussed in Section 2.3.5. It is not possible to make a link to a particle or boundary that would be out of line of sight so the LOS rules specified in Section 2.3.2 are implicitly accounted for.

2.3.5. Alignment Potential

In most cases, any two linked arms will not be colinear. The attractive force, therefore, would not pass through the center of the particle. This results in a torque and an additional force component to rotate the

particles into alignment to match this offset. This moment is represented by a functional contribution in the form of $C_{Rot}(\overrightarrow{A_{ik}A_{jl}} \cdot \overrightarrow{C_{ik}A_{I(k-1)}})^2$ for particle I arm k attached to particle j arm l.

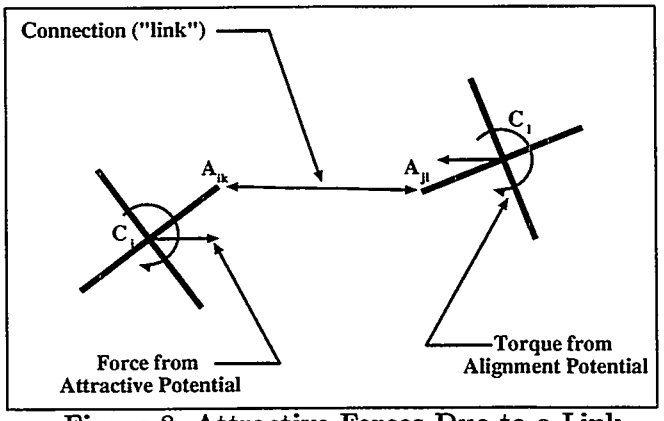


Figure 8. Attractive Forces Due to a Link

The arms are fixed at a perpendicular angle, so using $A_{I(k-1)}$ gives us only the component of the vector which is perpendicular to the line through the center of the particle, creating a pure torque to apply to the particle. Again these forces are only applied along existing links (see Section 2.3.3).

Figure 8 shows two particles which share a link and the resultant force and moment which must be applied to the centers of the particles.

As each particle has four links and each link is shared with exactly one other particle, we have $4n/2$ calculations to perform in finding both the attractive and alignment potentials. This is small by comparison to the work required for the repulsive potential.

2.3.6. Viscous Damping

Finally each particle has a damping force based on its translational velocity (V_i) and its rotational velocity (ω_i). This force is in direct opposition to the direction of movement. A particle I would see a functional contribution of $C_{D_{trans}}V_i + C_{D_{rot}}\omega_i$. This reduces the amount of free vibration in the system and helps to insure that the system will reach an approximate equilibrium state.

2.3.7. Particle Movement Based on the Potential

After collecting the forces and torques due to each of the parts of the potential the algorithm determines a new location, velocity and acceleration for each particle in both translational and rotational frames. A fairly standard first-order method known as the Euler method is used for forward time integration [6]. Translational and rotational velocity tolerances for the simulation allow the algorithm to determine when it has reached equilibrium. (These trigger points are set to control the amount of expansion or other activity.) The use of finite, non-zero values for these tolerances provide the algorithm with reasonable convergence criteria, allowing it to ignore undamped high frequency vibrations that could prevent convergence of the system.

By moving particles in this manner, the algorithm is reproducible. This means that if a problem occurs, it can be exactly reproduced and corrected. It also allows the system to be stopped and restarted with no loss of generality as long as the complete state of both particles and the mesh controller are maintained.

2.4. Connectivity Refinement

After the link finding phase and when the system has reached a new approximate equilibrium, either the mesh will be complete, or more likely, there will be unlinked arm ends. These unlinked arms represent "holes" in the mesh. Holes are dealt with as ordered loops of particles surrounding one or more open arm ends (see Figure 9). These loops are manipulated to reduce the number of loops and/or the number of open arm ends in each loop. At the end of this iterative process, a closed mesh results.

2.4.1. Determination of Holes (Loops)

Several types of loops exist. A "closed" loop is a loop with no unlinked arm ends. A loop with only a single unlinked arm is a "singleArm" loop. A loop with two unlinked arms on a single particle (of the loop) is a "doubleArm" loop. Similarly, a "tripleArm" loop has three unlinked arm ends on a single particle bounding that loop. Any loop that contains unlinked arms on two or more particles is defined as a "compound" loop. Figure 9 illustrates each of these types of loops. This figure also illustrates the value of the debugging tool used in the program. All particles which are not completely linked are rendered in a separate user-specified color and the state of the simulation can be rendered as often as needed during its execution. The current list of unclosed loops in the mesh is kept up-to-date for execution efficiency.

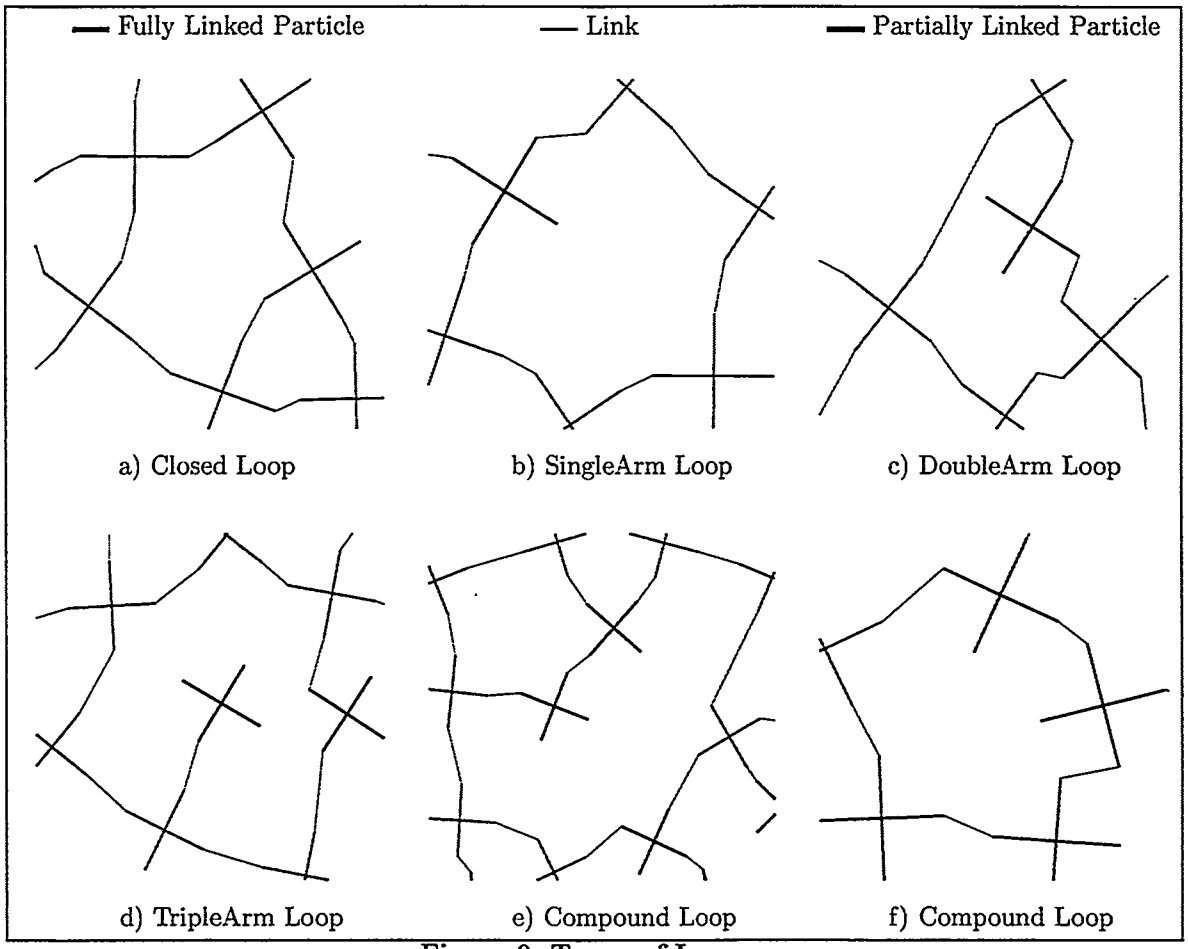


Figure 9. Types of Loops

2.4.2. Movement of Holes to Close the Mesh

After finding all the "unclosed" loops the algorithm begins the process of dealing with them. The goal is to remove them all, resulting in a closed, topologically valid mesh of quadrilateral elements. The loops are moved through the mesh to force them to either coalesce with other loops or to "squeeze" them out of the domain at the boundary.

Loops are often in a state which can be directly reduced in complexity. This is referred to as "reducing a loop to its minimum state". Figure 10 shows a loop reduction process - the loop is not closed but its complexity

is reduced. The dashed line indicates the new link which is added to change the loop from a compound loop to a double arm loop - this is considered to be a reduction in complexity. The system is restrained to create only valid connectivities in that no crossed links are allowed, no two particles may share more than one link, and no particle may have links to itself. By restraining the possible links to the adjacent free arms in the loop we insure that the loop is never split into two loops. This reduction favors links that will create a node shared by four elements over one shared by three or five elements, which, in turn, are favored over those which would be shared by six or more elements. This helps to improve the quality of the final mesh.

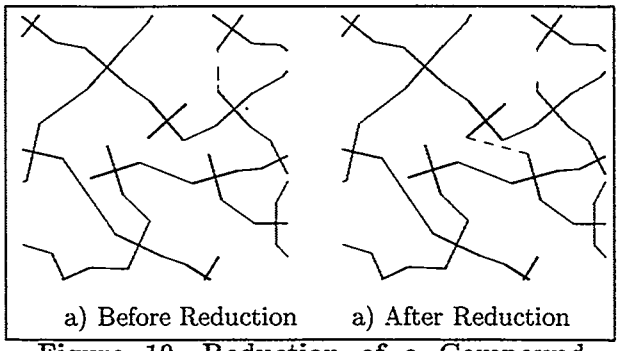


Figure 10. Reduction of a Compound Loop to a DoubleArm Loop

When two loops coalesce, they will either form a single loop of different complexity, or if they are complementary, they will result in a closed loop. For instance, two single-arm loops coalescing will either form a two arm compound loop or they will close completely as shown in Figure 11. In all cases, the overall complexity of the problem is always reduced during the process of loop coalescence.

Holes moved to the boundary will either add or remove a particle to eliminate the open arm ends in that loop, as shown in Figure 12. This is undesirable not only because it can interfere with the requirement of nodal continuity across shared boundaries with meshes on both sides but also because the total number of particles on the boundary is a user input value which the system tries to maintain as closely as possible. Hence, the preference is to coalesce loops rather than move them to a boundary. On the other hand, the ability to add or remove at least one particle from the boundary is critical in proving the mesh can always be closed [14]. In the future, the need for particle addition/removal will be reduced to only the level required to keep the algorithm provably robust.

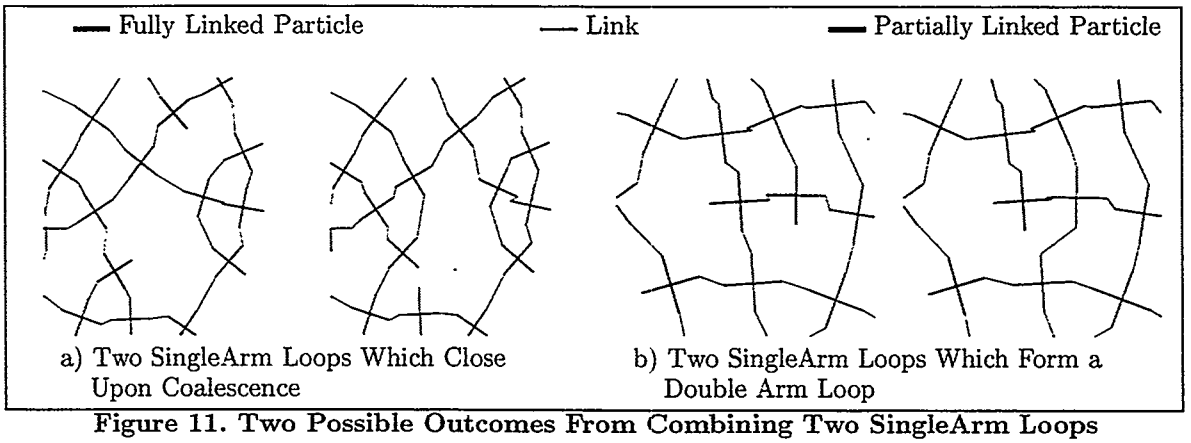


Figure 11. Two Possible Outcomes From Combining Two SingleArm Loops

In order to select a pair of loops to coalesce, priority is given to the following items in order.

1. Loops which share a common link are favored over those at a distance.
2. Loops which share a common particle are favored over those at a distance.
3. Loops of opposite type are preferred (loops with free arms on particles on the boundary prefer loops with all free arms on interior particles.)
4. Closer loops are preferred to more distant ones (in terms of distance between geometric centers.)

Figure 13.1 shows a pair of loops which share a link. The shared link is opened and then the reduction routines discussed previously act to close the loop. The dashed link will be removed creating a single larger

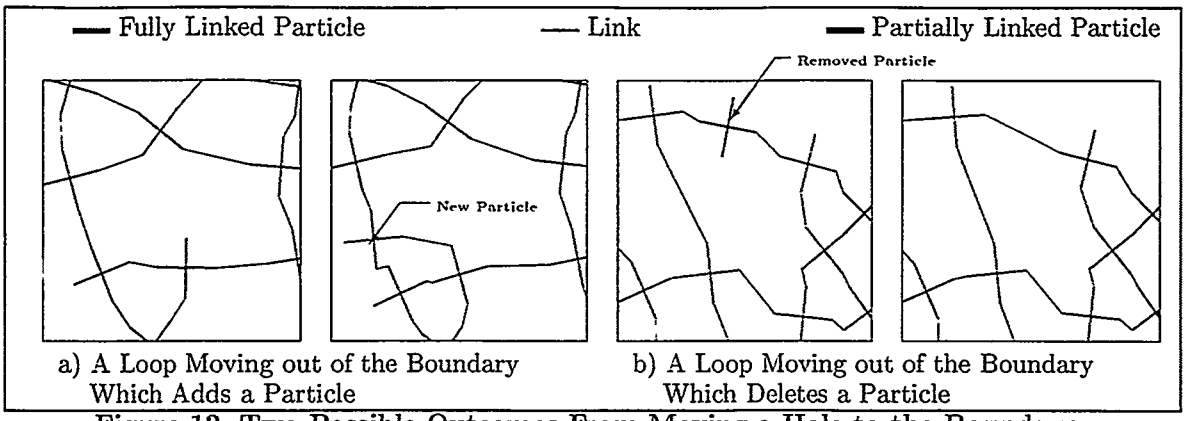


Figure 12. Two Possible Outcomes From Moving a Hole to the Boundary

loop. The new loop is then reduced giving the result shown. The system again runs to an approximate equilibrium before finding another pair of loops to combine.

Figure 13.2 shows two loops which share a common particle. The algorithm first opens a link in one loop which will force it to share a link with the other loop. The link on the shared particle which is most closely aligned to the vector between the centers of the two loops is selected for removal. The dashed link has been selected for removal. After removing the selected link the two loops are now in the shared link condition and are handled accordingly.

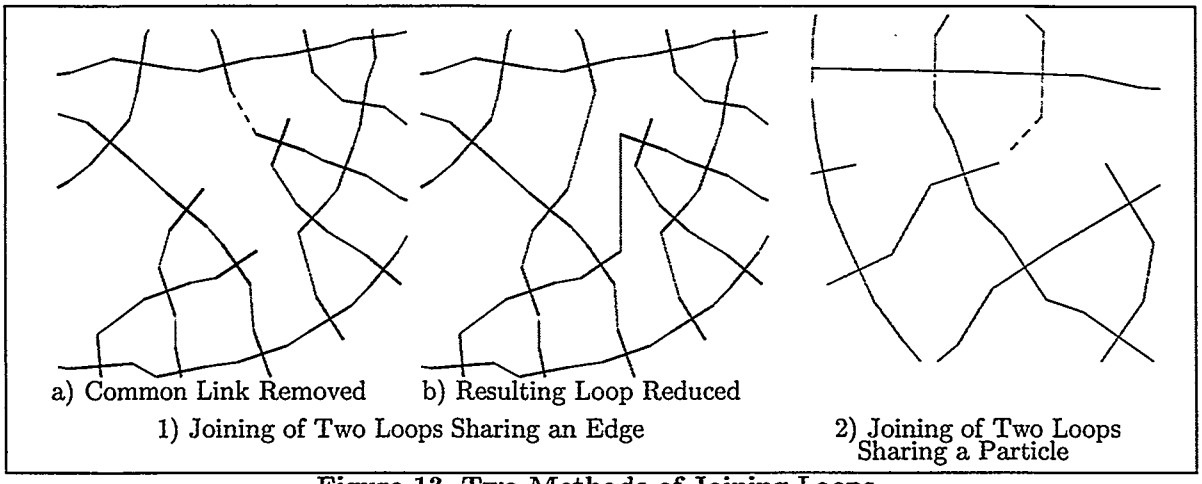


Figure 13. Two Methods of Joining Loops

In the case of a pair of loops that do not share a common particle, one loop is told to "open" itself to the other loop. This is done by finding the link which is most closely in the direction of the target loop and opening that link. These steps are repeated until the two loops share a link, and then merging is handled as described above (see Figure 14.) It is important in this case that the loop which is opening itself does not come to share a link with a loop other than the one it is merging with. The loop which is opening also must not intersect a boundary of the domain. To avoid these problems, the closest pair of loops is always selected, and a test is performed to insure that there are no boundaries between them.

3. Algorithm Analysis

All work that has been done on the algorithm to date has been focused on attaining reliable mesh closure. No work has been done on improving the computational efficiency of the algorithm.

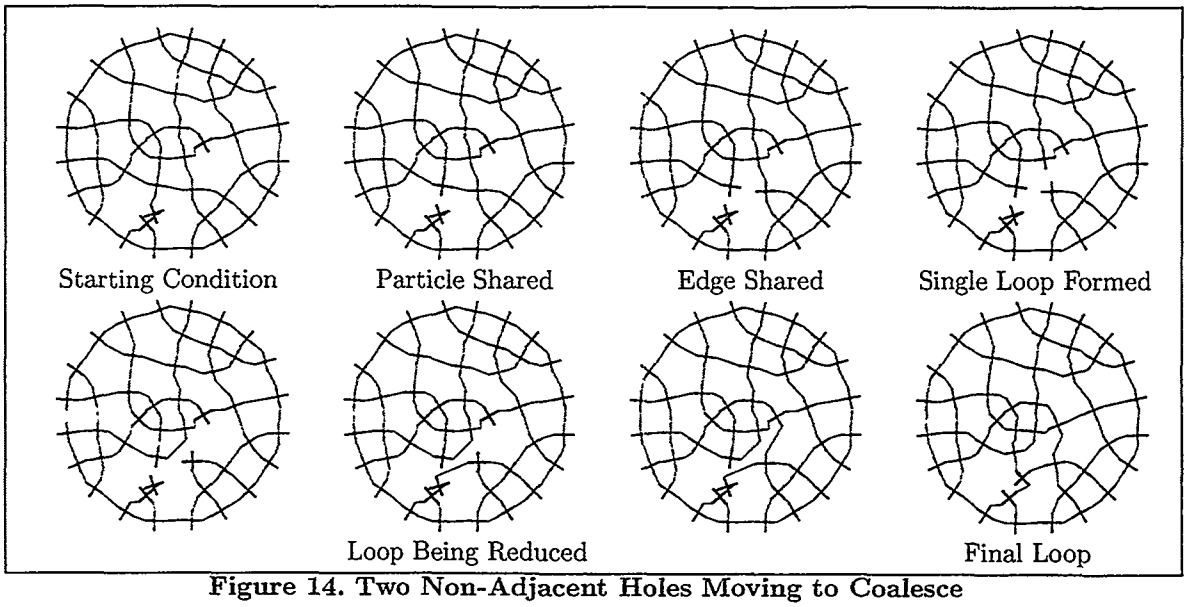


Figure 14. Two Non-Adjacent Holes Moving to Coalesce

Finding the distance between particles to calculate the repulsive potential dominates the computation. If we assume that we have n particles and m boundary segments, then each particle must check $(n + m - 1)$ other items every time repulsive forces are found. This gives a total of $n(n + m - 1)$ operations to find the repulsive forces. If we take advantage of the fact that the force contribution on particle I due to particle j (F_{ij}) is equal to the opposite of the force on particle j due to particle I $(F_{ij} = -F_{ji})$, we can cut this in half. If we also assume that $m << n$ (which is the case for the class of meshes used in our testing) this gives $\frac{n(n-1)}{2}$ operations per time step.

Since this functional contribution diminishes quickly as the inter-particle distance increases (approximately as the inverse square of the distance), one method of optimizing the simulation would be to maintain lists of the particles which significantly affect the amount of force on a given particle. At each time step, only the items on these proximity lists would be used to find functional contributions for a given particle. The lists would be rebuilt from the global data only when the simulation had moved the particles enough to change the proximity lists. A number of other potential performance enhancements to the algorithm are under consideration and the authors are confident that the meshing times will eventually become acceptable.

4. Results and Conclusions

The vast majority of the effort so far has been directed toward achieving closed, topologically-valid meshes on a consistent basis rather than toward mesh quality or algorithm performance.

Several test domains were used to determine the robustness of the code in closing meshes and handling various topologies. The test domains were a circle, a crescent figure, a cam, a dog-bone figure to test how the system handles ligaments and an offset washer to test a multiply-connected topology. Closure was achieved in better than 99 percent of the figures tested in the latest version of the code. All failures, so far, were traced to a single bug in a vendor's geometric modeling code rather than to the meshing algorithm.

The circular and crescent figures are the easiest to test and run in a relatively short time period. Most of the robustness testing was done using these two figures. The circular figure was used with a set number of particles (45), and the seed for the random number generator used to determine the initial location of the particles was varied. A script was used to run and save the results of 125 simulations in a way that allowed reproduction of the results at any time (see Figure 15).

Only one of the simulations failed to form a closed mesh. The problem was traced to a bug in the vendor code used for geometric computations. Work is currently underway with the vendor to solve this problem. Several of the meshes have areas of low element density (represented by a "large" number of elements sharing a single node). This results in unacceptable, acute element angles at these locations. Techniques for eliminating these types of problems before they occur are currently under development. The authors believe that such mesh quality issues can be addressed by modifying the functional appropriately.

The crescent figure was tested with a fixed number of boundary particles and a varying number of interior particles (see Figure 18). All fifteen of the series that were tested resulted in a topologically-valid, closed mesh. Larger numbers of interior particles produced a more evenly graded mesh. This brings up the question of how to balance the interior and exterior numbers of particles to produce a properly graded mesh. No solution to this has yet been found.

The cam and bone figures were used to verify various input methods and ligament behavior. Five versions of each were run using two with random initial placement and three using a method based on a Poisson's solution developed by Dr. Walter Gerstle [9]. This allows the size of the elements to be varied as a function of the geometry, reducing the total number of particles needed to produce a reasonable mesh. The completed meshes are shown in Figures 16 and 2.

No difference in the percentage of simulations closed has been noted due to the initial placement methods selected.

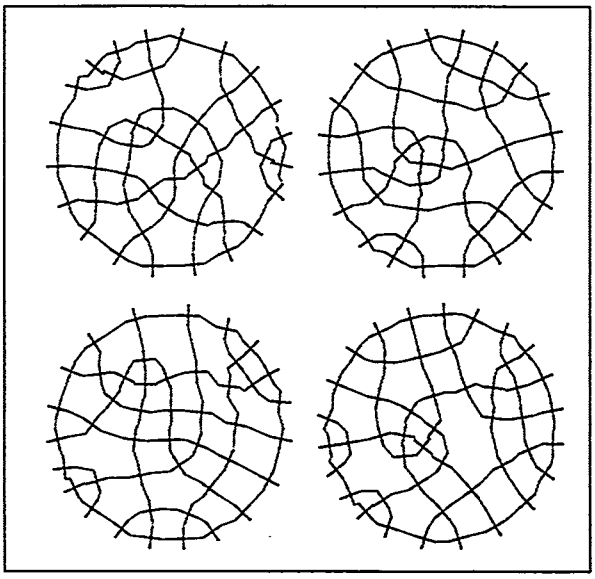


Figure 15. Closed Meshes on Circular Figure (Dual Representation)

The offset washer (see Figure 17) shows that the system can handle multiply connected topologies. Five versions were run with various numbers of particles and random number seeds. Three of the five closed correctly. The problem with the two was traced to the same vendor bug discussed in the results for the circular figure.

Although a formal proof of the algorithm's ability to close an arbitrary figure is beyond the scope of this paper, the algorithm appears to be a variant of the same class of problems that include the Hamiltonian Path and several shortest path problems [5, 7]. Although proof that the "best" solution of a Hamiltonian Path Problem is accomplished is NP hard, proof that a solution can be reached has been found [5]. This means that we can expect the algorithm to produce closed meshes on a routine basis. The most important remaining question is how to measure the quality of those meshes and improve them when required to meet the user's needs. To achieve the primary goal of the algorithm, this, too, must be done fully automatically.

Continuation of this work, planned or underway, includes the following:

- Methods to prevent the formation of and reduce the number/severity of acute angles in the 2D mesh.
- Extension of the functional definition and connectivity refinement method to 3D.
- Use of the connectivity refinement method to increase the robustness of other meshing algorithms.
- Use of a combinatorial approach to find better links and minimize mesh quality problems *a priori*.

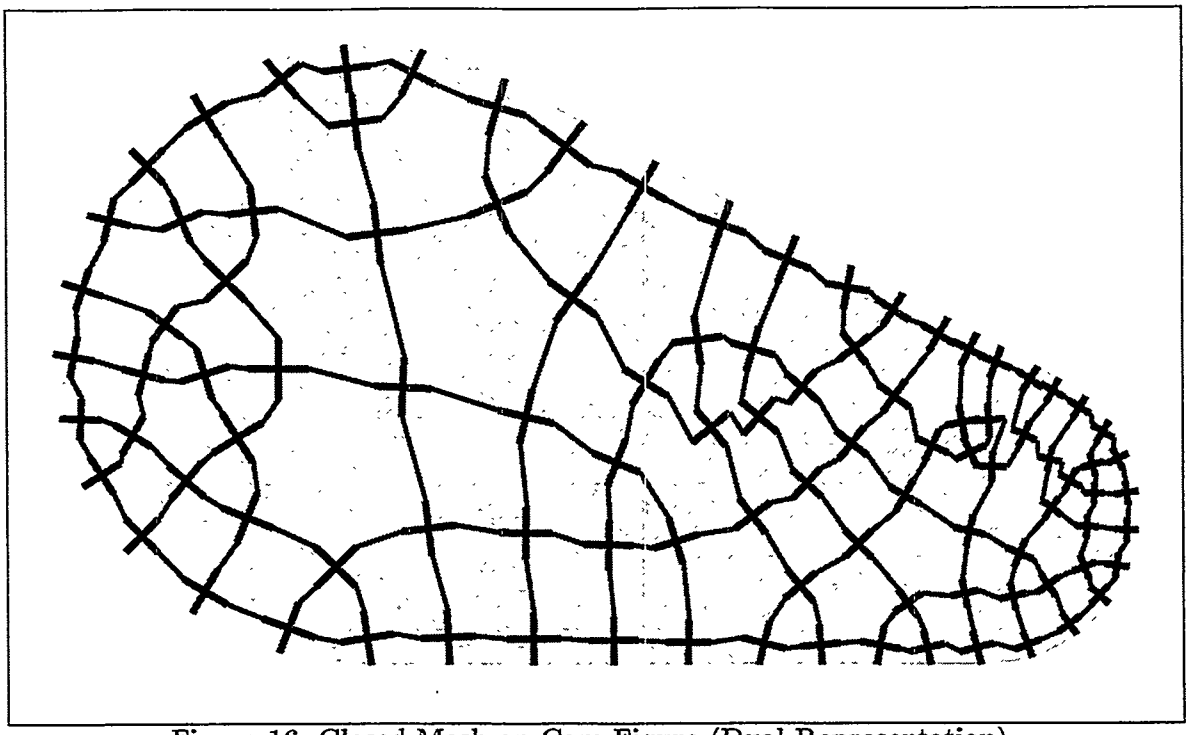


Figure 16. Closed Mesh on Cam Figure (Dual Representation)

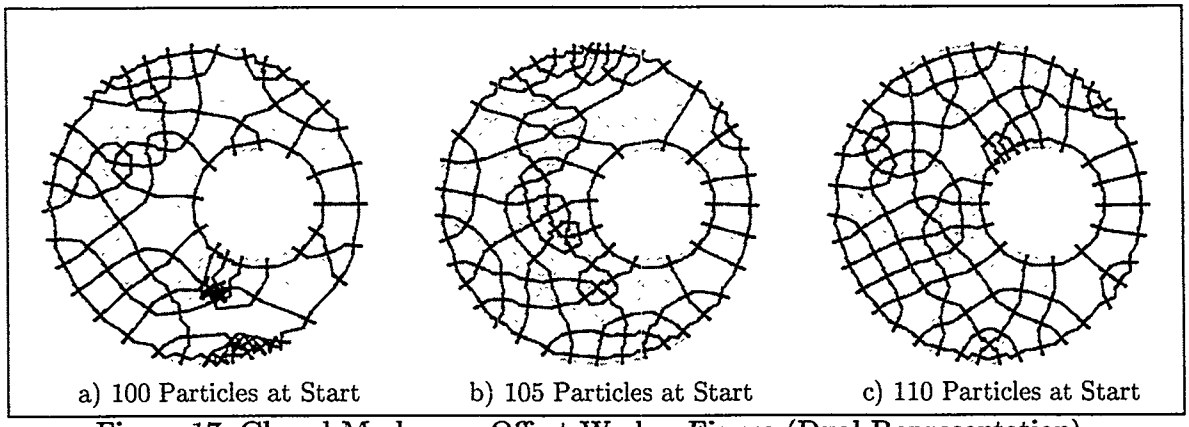


Figure 17. Closed Meshes on Offset Washer Figure (Dual Representation)

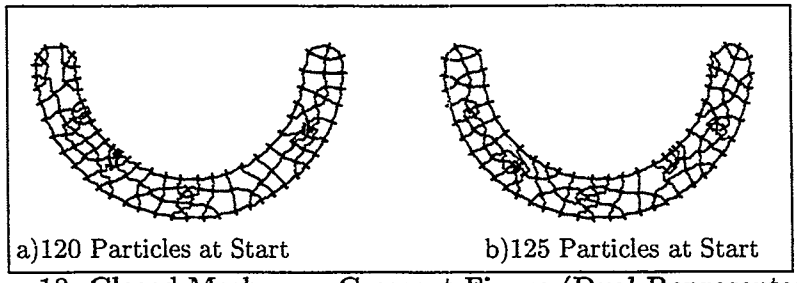


Figure 18. Closed Meshes on Crescent Figure (Dual Representation)

References

[1] Steven E. Benzley, Ernest Perry, Karl Merkley, Brett Clark, and Greg Sjaardema. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *Proceedings, 4th International Meshing Roundtable* [17], pages 179–191.

[2] Marshall Bern and David Epstein. *Computing in Euclidean Geometry*, chapter Mesh Generation and Optimal Triangulation, pages 23–90. World Scientific, 1992.

[3] Frank Bossen. *Anisotropic Mesh Generation with Particles.* PhD thesis, CMU, May 1996.

[4] Bernard R. Brooks, Robert E. Bruccoleri, Barry D. Olafson, David J. States, and S. Swminathan. Charmm: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[5] Frank Carrano. *Data Abstraction and Problem Solving with C++.* The Benjamin/Cummings Publishing Co. Inc., Redwood City CA, first edition, 1995.

[6] Lloyd D. Fosdick, Elizabeth R. Jessup, Carolyn J. C. Scheuble, and Gitta Domik. *An Introduction to High-Performance and Scientific Computing.* MIT Press Cambridge Massachuesetts, first edition, 1996.

[7] Ian Foster. *Designing and Building Parallel Programs.* Addison-Wesley, Reading Mass., 1994.

[8] Lori A. Freitag, Mark Jones, and Paul Plassmann. An effecient parallel algorithm for mesh smoothing. In *Proceedings, 4th International Meshing Roundtable* [17], pages 47–58.

[9] Walter H. Gerstle, Joseph Jung, Walt Witkowski, and Malcolm Panthaki. Use of boundary integral method to determine local characteristic feature size of geometric domain. In *Proceedings, 6th International Meshing Roundtable '97* [18], pages 157–167.

[10] M. Halpern. Industrial requirements and practices in finite element meshing: A survey of trends. In *Proceedings, 6th International Meshing Roundtable '97* [18], pages 399–411.

[11] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vechhi. Optimization by simulated annealing. *Science*, 220(4589):671–680, 1983.

[12] Anish R. Malanthara. Object-oriented design in finite element research. Master's thesis, University of New Mexico, August 1997.

[13] Darryl J. Melander and Steven E. Benzley. Generation of multi-million element meshes for solid model-based geometries: The dicer method. In *Trends in Unstructured Mesh Generation.*, 1997.

[14] Scott Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatable hexahedral mesh of the enclosed volume. Computational Mechanics and Visualization Dept., Sandia National Laboratories, Albuquerque, NM 87185, 1996.

[15] Malcolm J. Panthaki, Walter H. Gerstle, and Raikantha Sahu. Comet: A framework for team-based, multi-physics computational mechanics. In *Proceedings of the DoD User's Conference, June 1997, Houston, Texas.*, 1997. http://www.arc.unm.edu/CoMeT.

[16] Timothy J. Ross. *Fuzzy Logic with Engineering Applications.* McGraw-Hill Inc, NY, first edition, 1995.

[17] Sandia National Laboratories. *Proceedings, 4th International Meshing Roundtable*, 1995.

[18] Sandia National Laboratories. *Proceedings, 6th International Meshing Roundtable '97*, 1997.

[19] Harley Wattrick and Don Dewhirst. Automotive industry needs for versatile meshing tools. In *Proceedings, 6th International Meshing Roundtable '97* [18], pages 413–423.

[20] David R. White and Paul Kinney. Redesign of the paving algorithm: Robustness enhancements through element by element meshing. In *Proceedings, 6th International Meshing Roundtable '97* [18], pages 323–335.

Tetrahedral Meshing

Boundary Layer Meshing for Viscous Flows in Complex Domains

Rao V. Garimella and Mark S. Shephard
Scientific Computation Research Center
Rensselaer Polytechnic Institute, Troy NY 12180.
E-mail:garimell@scorec.rpi.edu, shephard@scorec.rpi.edu

Abstract.

High reynolds number flow simulations exhibit strong gradients normal to walls and across shear layers requiring much finer resolution of the solution in some directions compared to others. To keep mesh sizes manageable for such problems, meshes with highly anisotropic elements are necessary. In this paper, a method for generating boundary layer meshes for arbitrarily complex non-manifold geometric domains is discussed. A popular strategy for generation of boundary layers meshes is generalized to deal with non-manifold model topology as well as to generate better shaped elements. Results on complex configurations are presented to demonstrate the capabilities of the mesh generator.

1 Introduction

High Reynolds numbers flow simulations exhibit strong gradients near walls and in free shear layers requiring fine resolution of the solution normal to the boundary layer and across the free shear layers. Refining a finite element mesh isotropically to capture strong gradients in some directions results in excessive refinement along the other directions. In the interest of keeping the mesh size manageable anisotropic elements are needed in the critical regions.

The requirements on a mesh generator capable of producing such meshes are severe. Some of the characteristics of anisotropic meshes capable of capturing the solution in high Reynolds number flows are (i) elements with aspect ratios of 1000 to 100000 or more, (ii) elements of high quality, created by careful choice of node positions to control dihedral angles, and (iii) smooth gradation into the isotropic mesh.

A popular strategy for generating boundary layer meshes is the Advancing Layers Method (also called advancing normals method) [1, 2, 3, 4, 5]. Other techniques for generating anisotropic meshes can be found in [6, 7, 8, 9, 10, 11, 12, 13, 14]. The advancing layers method starts from a triangulation of the surfaces from which the boundary layer mesh is grown. From each surface node, nodes are generated along a single direction. These nodes form the basis for constructing layers of prisms on top of each surface triangle. The prisms are used directly as elements, or are subdivided into three tetrahedra. Care is taken to prevent crossover of the normal directions along which nodes are placed to avoid formation of invalid elements by smoothing of the directions or deletion of elements. Also, interference of the boundary layer mesh on different surfaces has been accounted for and eliminated by element deletion in some implementations. The restriction of growing a single set of nodes from each surface node constrains the advancing layers method to a subset of non-manifold models and also limits the quality of elements that can be created.

A generalization of the advancing layers is presented in this paper. This generalized advancing layers method has several improvements over the standard advancing layers techniques allowing the generation of quality boundary layer meshes for non-manifold geometric models of arbitrary complexity. Section 2 provides motivation for, and an outline of the Generalized Advancing Layers method. Section 3 discusses issues associated with point placement for boundary layer meshing of arbitrarily complex non-manifold geometric domains. Section 4 describes techniques to ensure that the boundary layer elements generated will be valid, while the process of boundary layer element creation is presented in Section 5. Section 6 discusses the method used to guarantee that the boundary layer mesh is not self intersecting. The paper concludes with results and discussion in Section 7.

2 Generalized Advancing Layers Overview

The Generalized Advancing Layers Method uses the surface mesh as the basic structure on which to grow the anisotropic boundary layer mesh. However, unlike other methods, the current approach allows for multiple sets of

nodes to emanate from each surface mesh vertex thereby facilitating the creation of valid meshes for non-manifold models and good quality element construction near boundaries with sharp corners or high curvature.

The presence of multiple sets of nodes originating from a single surface node allows the prisms on the mesh faces to be much better shaped naturally leading to good tetrahedral element quality. The gaps created between the prisms are abstracted as more general polyhedral shapes called *blends* which are tetrahedronized. Filling the gaps between the prisms is an important part of the algorithm since failure to do so will expose the highly anisotropic faces to the isotropic mesher. High aspect ratio faces may also get exposed if the number of layers between adjacent prisms is different. To hide these faces from the volume mesher, tetrahedral elements called *transition elements* are created on top of the prisms with fewer layers than any adjacent prism.

Curves along which boundary layer nodes must be placed are chosen for all surface mesh vertices (*growth curves*). The number of growth curves at mesh vertex is dependent on the topology and geometry of the mesh at the vertex. In non-manifold models, model faces may have regions on one or both sides of the face. Also, any number of model faces may be connected to a single model edge. Due to the generality of the non-manifold topology, two or more growth curves may be necessary at nodes at non-manifold boundaries to ensure that mesh edges will not penetrate model faces that locally divide the space into two halves [15].

Growth curves are first chosen at mesh vertices on model vertices. Mesh entities of growth curves lying on model edges are incorporated into the model edge discretization. Next growth curves are chosen for mesh vertices classified on model edges. Growth curves on model faces are smoothed, shrunk or pruned to avoid crossover and self-intersection. Boundary layer mesh vertices and edges are created for these growth curves and adjacent growth curves are joined to form abstracts *quadrilaterals* and *triangles*. These boundary polygonal constructs are triangulated and the triangles incorporated into the model face discretization. Growth curves are then constructed at mesh vertices classified on model faces and are smoothed, shrunk and pruned to ensure creation of valid elements. Mesh vertices and edges are created along these growth curves. Entities from adjacent growth curves connected up to form *triangular prisms* built on surface triangles and the prisms are tetrahedronized. If there is a difference in the number of layers between adjacent prisms, *transition elements* are created atop the prisms with fewer layers. As a final step in the creation of anisotropic elements, the gaps created between prisms due to the presence of multiple growth curves are abstracted as more general polyhedral shapes (*blends*) and tetrahedronized.

Once the anisotropic mesh is created, it is checked for any self intersections. Self intersections are fixed first by shrinking the layers locally and then by deletion of elements, if necessary. This completes the boundary layer mesh which is then handed over to isotropic mesher for completing the meshing process. The steps in the process are illustrated in Figure 1.

3 Finding Growth Curves

Point placement in the boundary layer mesh occurs along growth curves while respecting user mesh size specification for the boundary layers. Growth curves may be boundary, interior or both. The definition of growth curves allows them to start off with the nodes on the boundary of the model, then to separate from the model surface and grow into the interior of the model, and then reattach to the surface again. Reattachment of growth curves is not yet permitted in the implementation. Since the elements in the anisotropic mesh are created from triangular prisms and other blend polyhedra the quality of elements resulting is heavily influenced by the deviation of the growth curves from the normal direction to the base triangle. Therefore, nodes of growth curves growing from mesh vertices classified on model edges and vertices are allowed to lie on the boundary if the normal direction of the growth curve is close enough to the boundary. Figure 2 illustrates the types of growth curves.

Since multiple growth curves can go into a single region from any mesh vertex on the surface, each growth curve is associated with mesh face uses (mesh faces and their sides) using nodes of this growth curve to form elements. This information facilitates the connection of nodes of different growth curves in an unambiguous manner. Nodes of growth curves from mesh vertices of a mesh edge use are connected to form an abstraction called a boundary layer quadrilateral (Figure 3a) if they reference a common mesh face use. Nodes of growth curves from vertices of a mesh face are connected to form a boundary layer prism (Figure 3b) if they reference a common mesh face use. Nodes from multiple growth curves of a mesh vertex use are connected to form a boundary layer triangle (Figure 3c). The connectivity between nodes of a multiple boundary layer quads at mesh mesh edge uses and multiple growth curves at mesh vertex uses is more general. These constructs will be connected using more general procedures to create blend meshes (Figure 3d,e). Note that boundary layer quads, triangles, prisms and blends are abstract constructs that never actually exist in the mesh but are useful for algorithmic procedures and their discussion. In reality, triangles and tetrahedra of the individual layers are created directly in the mesh generator.

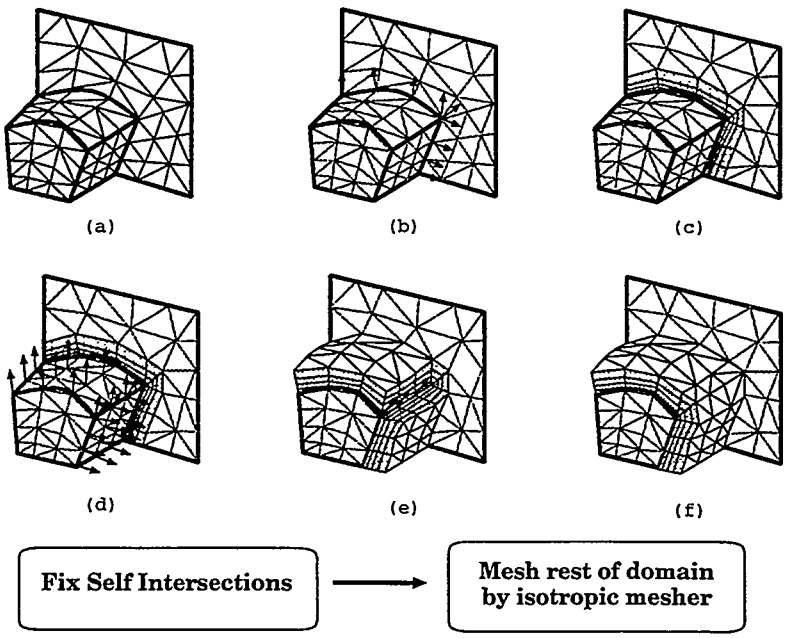


Figure 1: Steps in creation of boundary layer mesh (a) Surface mesh (b) Growth curves on model vertices and model edges (c) Boundary retriangulation (d) Growth curves on model faces (e) Prism creation (f) Blend creation.

3.1 Calculating the number of growth curves at each vertex

The number of growth curves at any mesh vertex with respect to a model face on which a boundary layer mesh is to be grown depends on the local mesh topology and geometry. The topological requirement for multiple growth curves at a mesh vertex with respect to a single face always arises for non-manifold faces with region to be meshed on both sides. At these boundaries at least two growth curves are necessity for a valid mesh. Figure 4 gives an example of such a situation. More complex interactions occur at model edges and vertices where, locally, the space may be subdivided into subspaces or manifolds, such that points from two different manifolds cannot be connected to each other without penetrating a model face [16, 17, 18, 19]. At such places, the minimum number of growth directions required depends on the number of manifolds.

Multiple growth directions may also be necessary at some mesh vertices due to the geometry of the model faces. Multiple growth directions are needed if the normals of the mesh faces vary so much that it is not possible to find valid common nodes that yields positive volume elements for the wedges of the connected mesh faces. Multiple growth directions are also desirable in cases where the normal variations are large enough to result in poorly shaped elements with large dihedral angles. Therefore, the procedures to find the growth curves first find the minimum number of growth curves required by the topology and increase this number further as dictated by the geometry.

3.2 Node placement and classification

The determination of growth curve starts with a procedure which assumes a single classification for all the nodes of the growth curve. If this causes problems with geometric or topological validity that cannot be resolved then a more general procedure is applied in which the nodes of a growth curve can have different classifications. In both cases the starting point is to place the nodes of the growth curve on the lowest order model entity possible.The basis for growing a boundary layer growth curve on a model face or edge is to use the average normal of the given mesh face uses of that direction and then find appropriate locations on the model entity close to the initial positions of the nodes. The specific checks to be satisfied for node placement and classification required to respect topological validity of the mesh, topological compatibility of the mesh with the model and estimated geometric validity of mesh are given in [15]. If creating a boundary growth curve violates any of these requirements, the growth curve is grown into the interior. Growth curves from the mesh vertices on a model face are always straight lines classified in the interior of

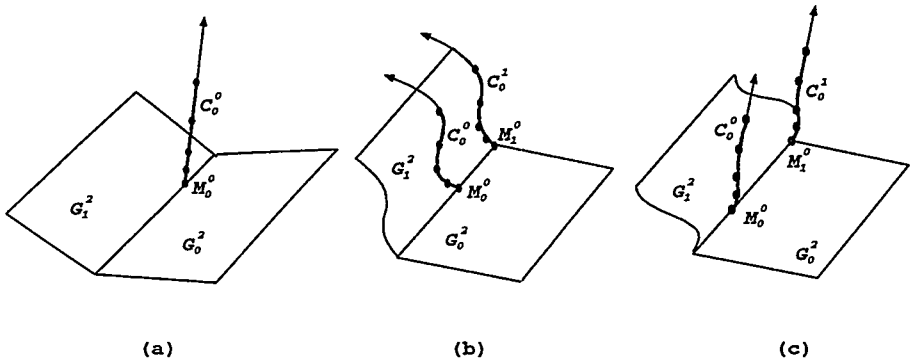


Figure 2: Types of growth curves (a) Interior Growth curve (b) Boundary growth curves (c) Partly boundary and partly interior growth curves

the model. Node spacing for growth curves may be *geometric*, *exponential* or *adaptive* [15]. In the adaptive method the growth curve at any location is terminated as soon as the layer thickness is approximately equal to the isotropic mesh size in the neighborhood.

4 Ensuring Element Validity

While growth curves are created with consideration for topological validity and topological compatibility, only preliminary consideration is given to geometric validity of elements during the point placement phase. This is because geometric invalidity arises mostly due to interactions between entities of neighboring growth curves connected to form tetrahedral elements. For this reason, it is only viable to check for geometric validity after creation of all growth curves. Invalidity of boundary layer elements occurs either due to crossover of growth curves or more than a 90° deviation of the growth curve from the mesh face normal.

4.1 Element Validity Checks

The individual triangles of the boundary layer quad are checked in real space for near zero area since negative area does not have any meaning for triangles on a general surface in 3-space. Then adjacent triangles are checked to see if the dihedral angle along their common edge is greater than an assumed tolerance α (taken to be 90°). This is to measure if the discretization of the surface is excessively distorted. Finally, the lateral edges of the boundary layer quad are checked for intersection by projection onto a common plane to verify that the growth curves are not crossed over. The validity of boundary layer prisms is done by checking the validity of all its component tetrahedra.

4.2 Correcting Invalid Elements

Correction of crossover considers three different methods, Smoothing, Shrinking and Pruning in that order. In the smoothing step, a weighted Laplacian smoothing procedure is applied to growth curves to eliminate crossover. Smoothing of boundary and interior growth curves is done separately on a model edge-by-edge and model face-by-face basis respectively for two reasons. Firstly, all boundary quads must be incorporated into the surface mesh before growth directions are determined at nodes on the model face. Secondly, boundary growth directions may be general curves on model surfaces preventing a direct application of the Laplacian smoothing technique. The shrinking procedure is based on the principle that crossover often occurs because the thickness of the boundary layer is high relative to the local curvature of the model face or the acuteness of the angle between model/mesh faces. Therefore, the shrinking process locally reduces the thickness of the boundary layers if it will make the affected elements valid. Shrinking of growth curves is followed by recursive adjustment of neighboring growth curve heights to ensure a smooth gradation of boundary layer thickness.

In some severe or degenerate case, neither smoothing nor shrinking can fix the invalid elements. In such a case the

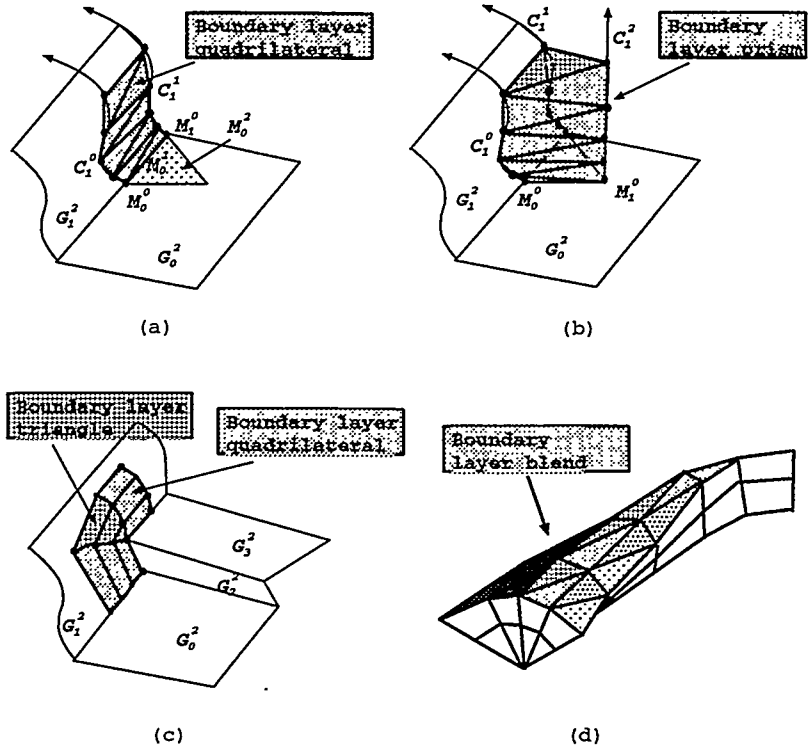


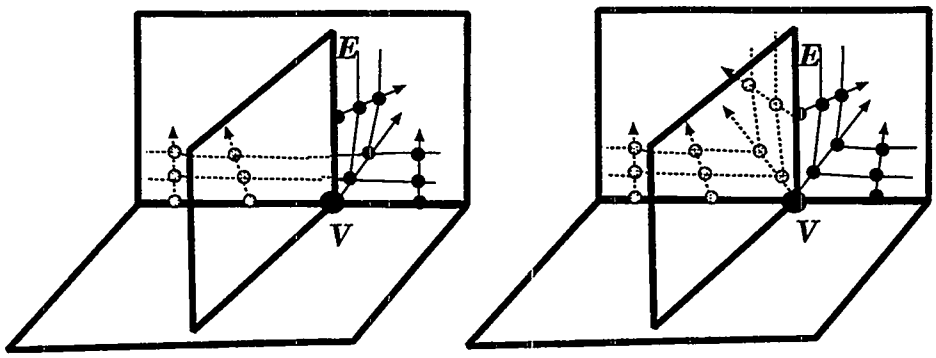
Figure 3: Abstract polygonal and polyhedral constructs in the boundary layer mesh.

growth curves of affected elements are pruned, i.e., some of their nodes are deleted, so that only valid elements are remaining. Pruning of growth curves is also accompanied by recursive pruning of adjacent growth curves to avoid sudden variations in the node distribution. When a simple prism is pruned, some of its high aspect ratio faces get exposed to the volume mesher which is not designed to properly deal with them. Therefore, transition elements are constructed on the top of prisms having a level difference with adjacent prisms. Figure 5 illustrates the three methods for fixing growth curve crossover in a 2D boundary layer mesh.

5 Boundary Layer Element Creation

Of the different constructs used to create anisotropic elements, the boundary layer quads and prism contribute the most elements, being grown on surface mesh edges and mesh faces respectively. The other constructs only serve to shield the isotropic mesher from the stretched mesh faces on the sides of prisms. The presence of multiple growth curves will yield adjacent prisms which are separated by gaps. The walls of these gaps are formed by the highly stretched faces of the anisotropic mesh which must be shielded from the isotropic mesh generator. These gaps are filled by boundary layer blends (Figure 6). In principle, boundary layer blends may or may not have fixed number of elements along the edges. Variable blend constructs typically occur at model edges where the dihedral angle between the connected mesh faces is varying along the edge. While boundary layer blends at mesh edges are easy to mesh using templates, boundary layer blends at mesh vertices are harder since an arbitrarily number of prisms and blends may be incident on the mesh vertex.

The last boundary layer construct used is the transition tetrahedron. When the growth curves of a mesh face forming a prism have different number of nodes, a step is formed in the boundary layer. To avoid leaving the highly stretched faces exposed, tetrahedra are created transitioning from the growth curves with larger number of nodes to those with smaller number of nodes (See Figure 7). Transition tetrahedra can span more than one layer (A similar concept has also been introduced by Connell and Braaten[1] for dealing with one layer difference).



(a) Single growth curve along edge E *(b) Multiple growth curves along edge E*

Figure 4: Need for multiple growth curves at non-manifold boundaries. In (a) a single set of growth curves originate from nodes on model vertex V and model edge E. Edges connecting the shaded and solid nodes then intersect model edge E. In (b) Multiple growth curves from nodes on vertex V and edge E result in valid connections of the nodes.

5.1 Model Edge Retriangulation

The insertion of boundary layer mesh edges and vertices classified on model edges is carried out through local mesh modification operators [20]. Given an edge to be inserted into the discretization of a model edge, existing mesh edges overlapping the edge to be inserted are identified by examining the one-dimensional parametric space of the model edge. The end vertices of the edge to be inserted are introduced into the model edge discretization by an edge split if coincident vertices do not already exist in the mesh. Then all edges overlapping the edge to be inserted are deleted.

5.2 Triangulation of Boundary Layer Quads and Blend Triangles

Triangulation of boundary layer quads is done by connecting nodes of adjacent growth curves *not* originating from the same mesh vertex. In converting these boundary layer quads to triangles the choice of the diagonal is dictated by the validity of the connected prisms. For reasons explained in the section describing prism tetrahedronization (Section 5.4 below), the diagonal is made by connecting lower node of the growth curve at the mesh vertex with a lower identifying number (vertex id) to the next higher node of the growth curve at the other vertex. Creation of boundary layer blend triangles is similar to the creation of boundary layer quads except that they establish connections between nodes of two growth curves originating from the same mesh vertex.

5.3 Model Face Retriangulation

Mesh entities classified on model faces are incorporated into the surface mesh by using local mesh modifications [20]. This method is chosen to avoid the use of the parametric space provided by the geometric modeler which can sometimes be highly distorted. This appears to be a particularly severe problem for surfaces formed by concatenation of triangular facets. The model face retriangulation procedure is done for each model face that the growth curves of a model edge affect. Given a model edge and a model face on which growth curves lie, the following steps are carried out to create and incorporate the boundary layer mesh into the surface mesh triangulation:

1. Boundary layer quads classified on the the model face are created as described above.
2. Each boundary layer mesh entity that forms the outer boundary of the set of boundary layer mesh faces classified on the model face is incorporated into the surface mesh by the edge recovery procedure briefly described below and discussed in full detail in [20]. Once all the necessary edges have been recovered, the outer boundary of the set of faces to be inserted into the mesh exactly matches the outer boundary of a set of faces in the underlying surface mesh.
3. Mesh faces of the existing surface triangulation overlapping the boundary layer mesh faces are deleted and the boundary layer faces incorporated into the mesh instead.

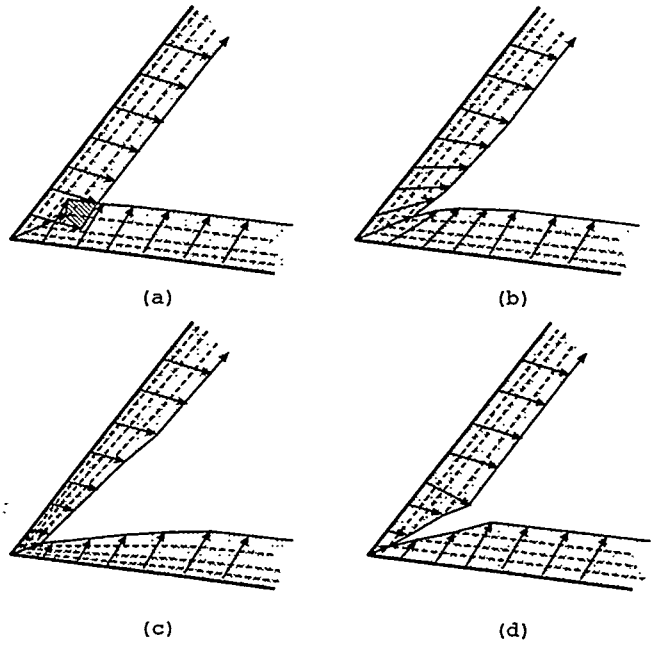


Figure 5: Fixing growth curve crossover in 2D (a) Mesh with crossover at corner (b) Mesh with crossover fixed by smoothing (c) Mesh with crossover fixed by shrinking (d) Mesh with crossover fixed by deletion

The edge recovery procedure inserts the vertices of the edge to be recovered into the existing surface mesh using projection along mesh face normals. A local mesh optimization is performed to eliminate any other poorly shaped faces created during the vertex insertion process. Next, a path of edge connected mesh faces is found from one vertex to another, again by projection of the edge to be recovered onto planes of the mesh faces. Mesh edges that cross the projected edge are successively swapped to recover the edge. After recovering the edges on the outer boundary of the set of quads to be introduced into the mesh, the mesh faces overlapping the boundary layer mesh faces are deleted and the boundary layer quads put in their place. The mesh resulting from face retriangulation is subjected to checks and remedies to ensure that it is not self intersecting [21].

5.4 Prisms, blends and transition elements

The bulk of the elements in the boundary layer are from the boundary layer prisms. Boundary layer prisms are grown on mesh face uses by connecting the three growth curves at the face vertices which share each mesh face use. The tetrahedronization of each boundary layer prism gives rise to three tetrahedra. Boundary layer prisms can be thought of as being formed by three quads that are grown from the edges of the triangular mesh face. There are eight possible combinations of diagonals for the quads of a prism. Of these only six are configurations which can be tetrahedronized without the insertion of any new points inside the prism. Therefore, in assigning directions for the diagonals of the quads in the boundary layer mesh, care must be taken not to assign directions such that some prisms cannot be tetrahedronized. This is done by a very simple algorithm based on numbering of the surface mesh vertices. Given a surface mesh with any arbitrary assignment of unique numbers (ids) for the mesh vertices, the ids of vertices of a face in either clockwise or counterclockwise direction cannot be strictly increasing or strictly decreasing. Using this notion, the diagonals of boundary layer quads are constrained to go from the lower node of the growth curves of vertices with a lower id to the upper node of growth curves of vertices with a higher id ensuring the validity of all prism tetrahedronizations. The tetrahedronization of boundary layer prisms is done using two templates.

Boundary layer blend polyhedra are created between the exposed sides of two prisms from adjacent faces. Only simple blends, in which the gap between the two quads is filled by elements without the creation of new points, are considered here. Two quads originating from the same edge may have separate growth curves at each end or share a common growth curve at one end giving rise to a total of three types of blend polyhedra [15]. In the more general situation, it is proposed that if the gap to be filled between two quads is too large then additional growth curves be

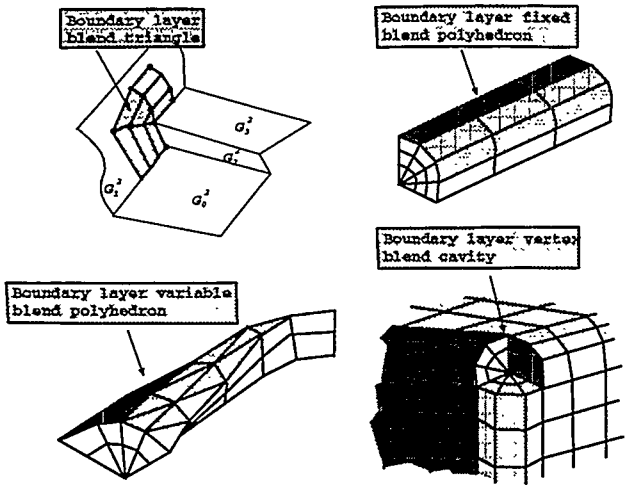


Figure 6: Blend elements

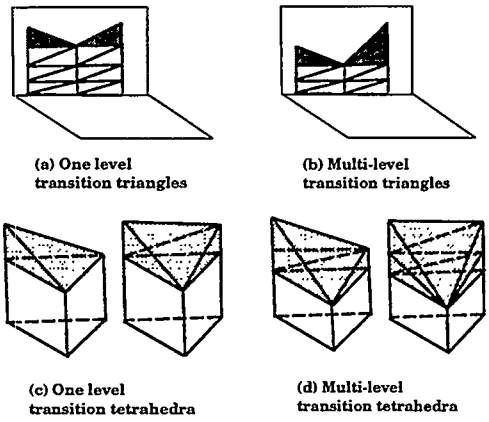


Figure 7: Transition elements

introduce to produce a blend mesh as shown in Figure 6.

Since boundary prisms can be made only by connecting corresponding nodes on the component growth curves, faces from boundary layer quads with more nodes than the others remain exposed. These stretched faces are closed off from the isotropic mesh generator meshing the rest of the volume by transition triangles for boundary layer quads and transition tetrahedra for boundary layer prisms. Transition triangles are formed atop boundary layer quads with a level difference between the two growth curves by connecting the top nodes of the two growth curves and forming an element similar to the lower triangle of a boundary layer quad. Atop a prism with one level difference between its component growth curves, there may be one or two transition tetrahedra depending on whether one or two growth curves have fewer nodes than the others (Figure 7). If the level difference is more than one then transition tetrahedra are stacked on top of each other.

6 Fixing Boundary Layer Intersections

When boundary layer elements are generated on model faces that are too close to each other the layers may run into each other in which case the polyhedral cavity that is presented to the isotropic mesher is self-intersecting. Since the isotropic mesher expects a polyhedral cavity with no self-intersections, this situation must rectified. Boundary layer intersection is fixed by local shrinking of layers and/or local pruning of growth curves leading to deletion of elements. Correction of self intersections are done after element creation as it is simpler to find the mesh faces forming the polyhedral cavity left to be meshed and the neighborhood used for intersection checks can be more localized.

The technique used to detect self intersections looks at mesh faces that have fewer regions connected to them than they should in a completed mesh (*exposed* faces). These faces may be classified on model faces with no boundary layer, and top and side faces of the boundary layer prisms, blends and transition polyhedra. An octree is built in the domain and the exposed faces attached to the terminal octants. The octree serves as a localization structure for the intersection checks. Each exposed face is intersected with the set of faces in its neighborhood. If an intersection is detected, elimination of the intersection is attempted by local shrinking of the prism connected to the face while being constrained to keep connected elements valid. Since the algorithm only checks and fixes the intersection between *exposed* faces and not of entire prisms, blends or transition polyhedra, shrinking of two boundary layer constructs to correct interference may result in new intersections of other faces in the neighborhood. Therefore, the algorithm is iterative and continues until all intersections are fixed. The algorithm is made more efficient by recognizing that if an intersection between two faces has been detected and fixed there is a good possibility that the fix has caused new intersections or that there are already more intersections in the neighborhood. Therefore, the algorithm employs locally recursive intersection detection and correction in the neighborhood of a located intersection. If all the intersections between front faces cannot be fixed by shrinking prisms, then the growth curves are pruned using the same algorithm as described in Section 4.2. As in the case of deletion of elements to fix growth curve crossover, transition elements are introduced if adjacent prisms have differing number of layers.

7 Results and Discussion

Figure 8 shows the boundary layer mesh for capturing the flow in an expanding pipe. boundary layers are generated on the walls of both pipe sections and also on both sides of a surface introduced in the geometry to capture the free shear layer. The boundary layer mesh thickness on all the surfaces increases in thickness from the inflow to the outflow surface. In addition the number of nodes in the boundary layer mesh are different on each surface.

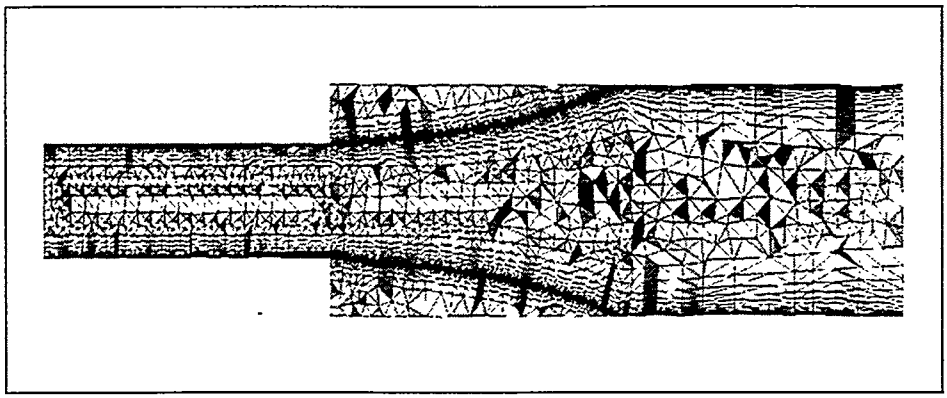


Figure 8: Anisotropic mesh for capturing flow in an expanding pipe including the free shear layers. The variation in the boundary layer thickness is clearly seen in the picture.

Figure 7 shows a cutaway of the boundary layer mesh for the space shuttle while Figure 10 shows the boundary layer mesh (in lighter shades) for the underbody of an automobile. The boundary layer mesh in these example was chosen to be thicker than normal for clarity of illustration. The mesh shown is valid, non-self intersecting and suitable for input to the isotropic mesh generator for completion of the mesh generation task. Using smoothing and compression of growth curves, the method has successfully resolved crossover of growth curves in areas of high curvature and interpenetration of the boundary layers in very confined spaces except in very few localized areas where elements had to be deleted.

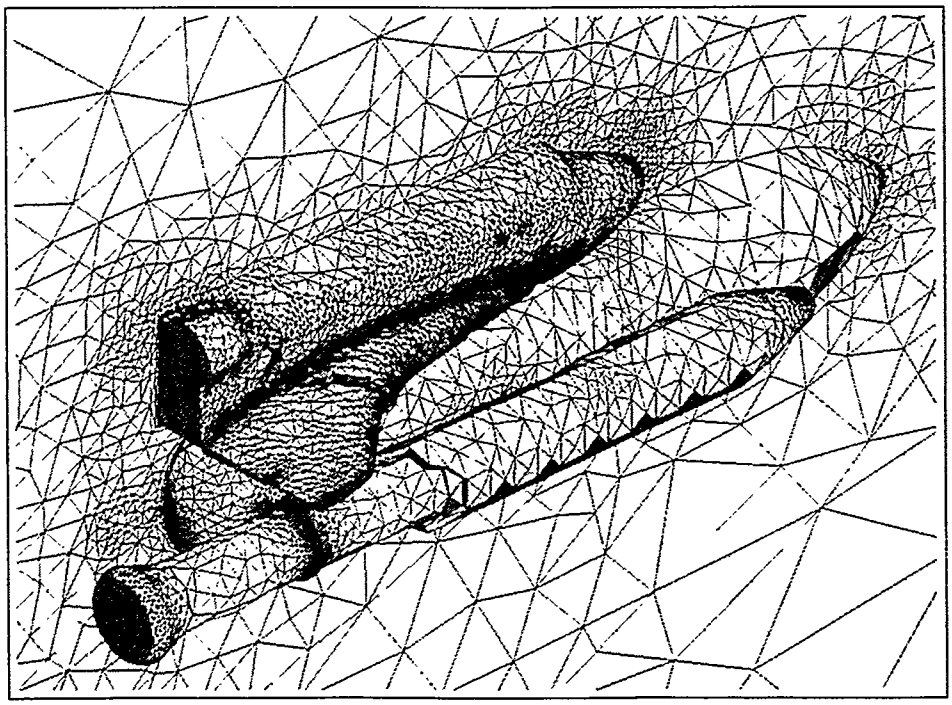


Figure 9: Cut-away of boundary layer mesh for a model of the space shuttle with center tank and booster rockets.

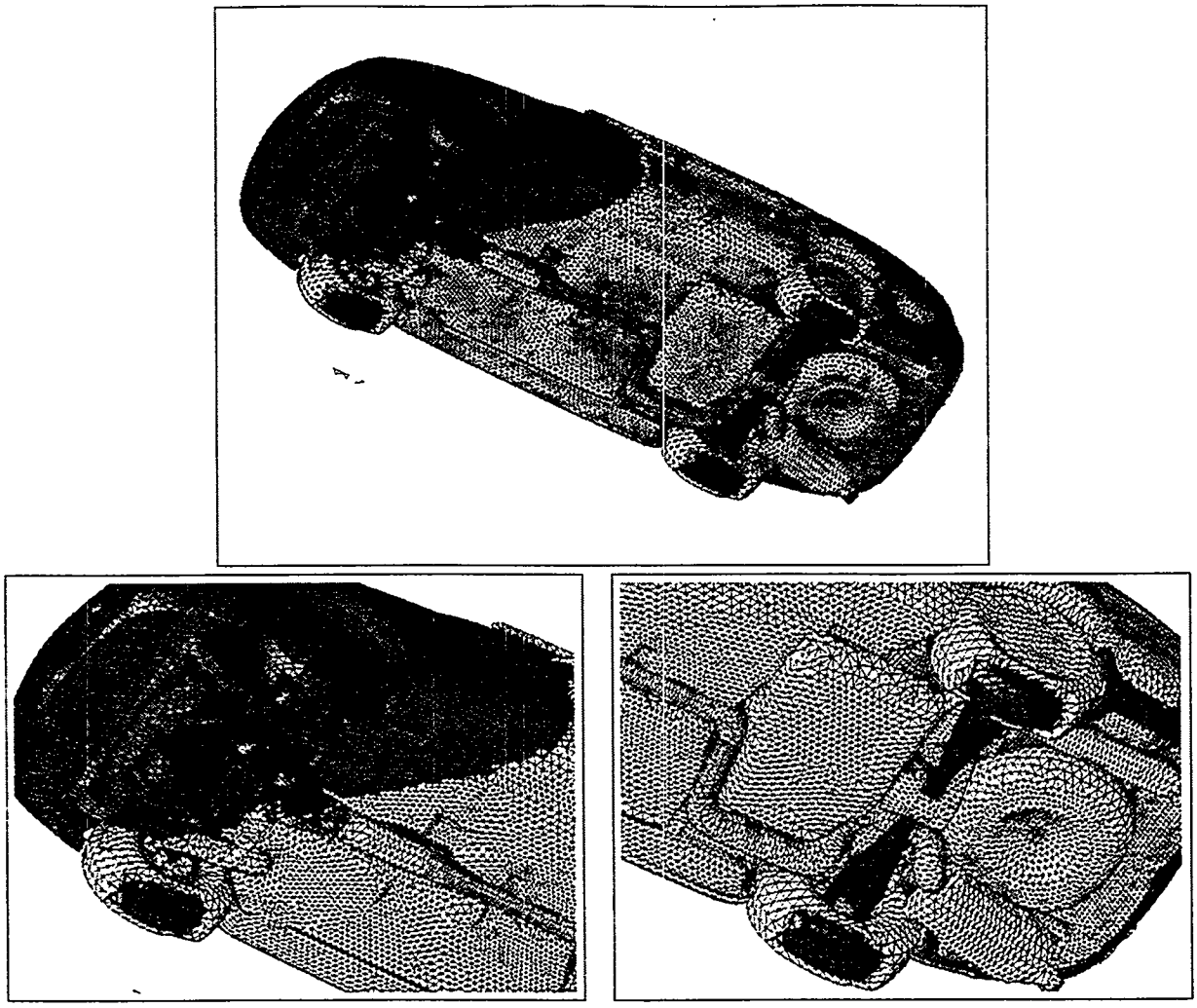


Figure 10: Underbody of a car showing cut-away of boundary layer mesh with zoomed-in views of the front and rear of the car.

The boundary layer meshes from the Generalized Advancing Layers procedure have been used for calculation of heat transfer coefficients for automobile configurations as well as for simulation of the Czochralski process of bulk crystal growth [22] (Figure 11).

References

[1] S.D. Connell and M.E. Braaten. Semistructured mesh generation for three dimensional navier-stokes calculations. *AIAA Journal*, 33(6), 1995.

[2] Y. Kallinderis and S. Ward. Hybrid prismatic/tetrahedral grid generation for complex 3-d geometries. In *AIAA-93-0669*, 1993.

[3] Y. Kallinderis, A. Khawaja, and H. McMorris. Hybrid prismatic/tetrahedral grid generation for complex 3-d geometries. In *AIAA-95-0211*, 1995.

[4] R. Löhner. Matching semi-structured and unstructured grids for navier-stokes calculations. In *AIAA-93-3348-CP*, 1995.

[5] S. Pirzadeh. Viscous unstructured three-dimension grids by the advancing-layers method. In *Proceedings of the 32nd Aerospace Sciences Meeting & Exhibit*, number AIAA-94-0417, Reno, NV, Jan 1994.

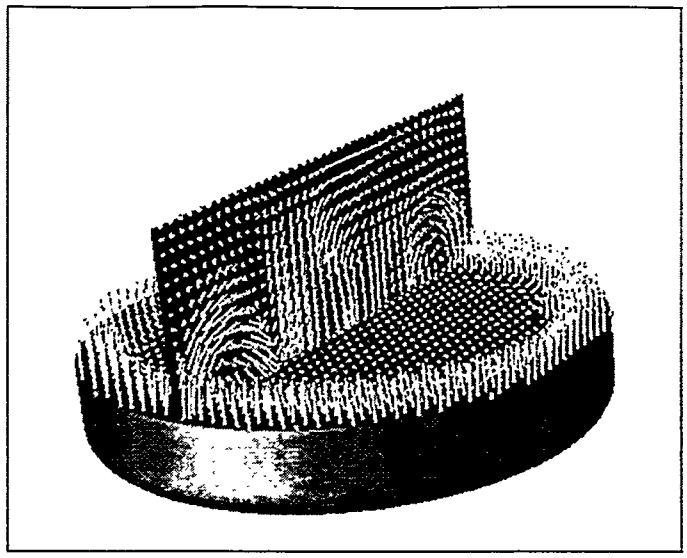


Figure 11: Velocity vectors horizontal and vertical-horizontal planes at t=50 seconds for growth of indium-phosphide crystal. The mesh has 6 anisotropic layers at the wall with a first layer thickness of 0.05 and a total of 634001 elements. (Courtesy: S. Adjerid et.al. [22])

[6] M.J. Castro-Diaz and et. al. New progress in anisotropic grid adaptation for inviscid and viscous flow simulations. In *4th International Meshing Roundtable*, Albuquerque, NM, Oct 1995. Sandia National Labs.

[7] M.J. Castro-Diaz and et. al. Anisotropic unstructured mesh adaptation for flow simulations. *Int. J. Numer. Meth. Engng.*, 25(4):475, 1997.

[8] O. Hassan and et. al. Mesh generation and adaptivity for the solution of compressible viscous high speed flows. *Int. J. Numer. Meth. Engng.*, 38(7):1123–1148, 1995.

[9] O. Hassan and et. al. Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *Int. J. Numer. Meth. Engng.*, 39:549–567, 1996.

[10] M.J. Marchant and N.P. Weatherill. Unstructured grid generation for viscous flow simulations. In *4th International Conference on Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, pages 151–162, Swansea, Apr 1994.

[11] D.J. Mavriplis. Adaptive mesh generation for viscous flows using delaunay triangulation. *Journal of Computational Physics*, 90:271–291, 1990.

[12] D.L. Marcum. Generation of unstructured grids for viscous flow applications. In *AIAA-95-0212*, 95.

[13] D.L. Marcum and N.P. Weatherill. Generation of unstructured grids for viscous flow applications. In *AIAA-95-1726*, 95.

[14] Dmitri Sharov and Kazuhiro Nakahashi. Hybrid prismatic/tetrahedral grid generation for viscous flow applications. *AIAA Journal*, 36(2):157–162, Feb 1998.

[15] Rao V Garimella. *Anisotropic Tetrahedral Mesh Generation.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY 12180, August 1998. in preparation.

[16] E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. In M. J. Wozny, J. U. Turner, and K. Priess, editors, *Geometric Modeling Product Engineering*, pages 107–130. North Holland, 1990.

[17] K. J. Weiler. *Topological Structures for Geometric Modeling.* PhD thesis, Rensselaer Design Research Center, Rensselaer Polytechnic Institute, Troy, NY, May 1986.

[18] M. Mäntylä. *Introduction to Solid Modeling.* Computer Science Press, Rockville, Maryland, 1988.

[19] M. Mortenson. *Geometric Modeling.* J. Wiley and Sons, New York, 1985.

[20] B. Kaan Karamete, Rao Garimella, and Mark S. Shephard. Recovery of an arbitrary edge on an existing surface mesh using local mesh modifications. *IJNME*, 1998. in submission.

[21] Hugues L. de Cougny. *Parallel Unstructured Distributed Three Dimensional Mesh Generation.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, May 1998.

[22] S. Adjerid, J.E. Flaherty, K. Jansen, and M.S. Shephard. Parallel finite element simulations of czochralski melt flows. In S.N. Atluri, editor, *Proceedings of ICES98*, Atlant, GA, 1998. to be published.

Revisiting the Elimination of the Adverse Effects of Small Model Features in Automatically Generated Meshes

M.S. Shephard, M.W. Beall and R.M. O'Bara
Scientific Computation Research Center
Rensselaer Polytechnic Institute, Troy, NY
shephard@scorec.rpi.edu

Abstract

This paper is a follow-up to a previous one [7] in which an algorithm to eliminate the adverse effects of small model features at the mesh level was presented. The present paper provides three additions to the original procedure. They are (i) a more efficient implementation in which those features that can be eliminated after surface mesh generation are eliminated before volume meshing, (ii) a method to define an up-dated model topology based on the results of the mesh modifications, and (iii) a demonstration of how this procedure can be used to perform substantial geometric model feature elimination.

1. Introduction

Automatic mesh generation procedures must interact directly with the geometric model as housed within a computer aided design system. In many cases, a number of the geometric details in the model are not important for the purpose of performing a specific analysis. Since the inclusion of these details can dictate the number and/or quality of elements in the mesh, it is desirable that these features not be present in the mesh sent to the analysis. An earlier paper [7] presented an algorithm for the automatic elimination of the influence of small geometric model features on the resulting mesh. In the current paper, consideration is given to improving the efficiency of the procedure, accounting for the changes to the model topology to account for the modifications made, and demonstrating the ability of the procedure to eliminate a sizable number of local geometric features from a model.

A key deterrent to the effective use of automatic mesh generation procedures is the time and effort required to provide the automatic mesh generator with the geometry required for the analysis at hand. The source of geometric complications includes [4,9,16,17]:

1. Combining geometry from different sources that do not properly match each other so that small gaps and overlaps exist.
2. Employing a method of geometry data transfer that loses topological adjacencies and does not include the geometric tolerance information needed to ensure that they can be reconstructed to be the same as in the original model.
3. The geometric model is valid; however, geometric features present will force the creation of many more elements than desired and/or the shape of these elements will not be satisfactory.

The first two situations are characterized by ambiguities in the model which require interpretation, thus, these situations can not automatically be resolved. On the other hand, since the

geometric model in the third is valid, it should be possible to devise algorithmic procedures which can automatically generate a mesh with the desired element sizes and gradations.

The starting point for the mesh generation process, including the elimination of the influence of small geometric features, is a valid geometric model for which there is a complete boundary representation. For purposes of the current discussion, a small geometric feature is one that has a characteristic dimension which is substantially smaller than that requested for the element sizes in that region of the domain. The next section outlines the original algorithm [7] developed to eliminate the influence of these features on a mesh. Section 3 introduces the key concept of the classification of mesh entities against the geometric model entities. Section 4 introduces a change to the original algorithm in which a pass of small feature removal is performed after surface mesh generation. Since most of these small features can be detected after surface meshing, eliminating them prior to volume mesh generation will improve the efficiency of the remaining meshing process. Section 5 addresses a more fundamental difficulty with respect to tracking the relationship between the model and mesh entities. Section 6 demonstrates the application of the procedure for the elimination of small, and not so small, model features.

2. Approach to Eliminate the Influence of Small Geometric Model Features

Two basic approaches to eliminate the adverse influence of small geometric model features on the quality of automatically generated meshes are to (i) modify the geometric model before providing it to the mesh generator, or (ii) automatically mesh the original model and perform local mesh modification in those portions of the model where small model features have produced poorly shaped elements when the element sizes are maintained at those specified by the user. Automatic geometric simplifications procedures have been development for specific situations [16,17,19]. However, it is not obvious that an automated set can be developed for all situations of interest. In addition, one would want be sure decisions to eliminate geometric detail are coordinated with the local information on element sizes which becomes quite straightforward in the second approach.

The second approach of modifying the mesh in the vicinity of the small geometric model features has the advantages of directly addressing the needs of a given mesh, and can be performed using a set of mesh modification operations [3,6,10,13]. Since the topology and geometry of the individual mesh regions is simpler than that of the geometric model regions, it is also more straightforward to evaluate the consequences of performing local modification.

The basic steps in this approach to the elimination of the adverse influence of small geometric model features are:

1. Generate a mesh employing the user specified mesh entity size distributions, and prevent the mesh generator from introducing smooth mesh gradations when disproportionately small mesh entities are caused by a small model feature. Allowing the mesh generator to automatically grade the mesh from disproportionately small model features greatly increases the number of elements created. An example of a turbine blade platform given in reference [7] shows the number of elements generated increased from 9767 to 12950 when smooth gradations were enforced from just two small model edges in a model that contained well over 100 model edges.

2. Identify and flag mesh entities that need to be deleted in order to eliminate the mesh entities with poor quality metrics.
3. Ensure that deletion of the flagged mesh entities results in geometrically valid mesh entities and their elimination does not introduce undesired dimensional reductions.
4. Apply the mesh modification operator(s) to delete the desired mesh entity and update classification and pointwise geometric information as required to ensure mesh validity.

Although the basic steps to eliminate the adverse influence of small geometric model features on the mesh are reasonably straightforward, there are a number of key technical issues that must be addressed in the successful implementation of a procedure. These include [7]:

1. Deciding which measures of element quality should drive the procedure. For example, a measure which is concerned with small dihedral angles is appropriate since geometric features that are small compared to the requested element sizes will always cause small angles when the mesh generator is constrained from introducing substantial mesh refinement as caused by the small model feature. Note that the use of small dihedral angles as the element quality measure does assume that the mesh optimization procedure within the mesh generator is allowed to split mesh entities opposite large dihedral angles in the original mesh. (Such split operations introduce only a local halving of the opposite mesh entities.) If such splitting operations are constrained, then element quality measures that capture both large and small dihedral angles must be employed.
2. Determining if the mesh modifications required to eliminate the adverse influence of small model features will introduce an invalidity (such as a dimensional reduction [7]) into the resulting mesh.
3. Applying the mesh modification operators needed to eliminate the adverse influence of the small model feature.
4. Maintaining the relationships between the mesh and the model as needed to ensure the validity of the mesh and to properly support future mesh modifications, such as adaptive refinement where some of the new nodes need to be placed on the model geometry and eliminated geometric model features may reappear due to the element sizes decreasing.

The original procedure given in reference [7] presented a set of procedures to deal with these four issues. Sections 4 and 5 introduce modifications to issues 3 and 4 aimed at improving the efficiency of the procedure, and making it a more flexible process with respect to its application as part of a general mesh generation and enrichment environment.

3. Relationship of the Mesh to the Model

A key component of automatic mesh generation directly from a solid model representation is ensuring the validity of the mesh with respect to the geometric model. The procedures that have been devised to address this issue focus their attention on understanding the association of the mesh to the geometric model [14,15]. The structure of the topological entities defining the model and the mesh provides the basic tool for understanding this relationship.

Topology provides an unambiguous, shape independent abstraction of the model and mesh. Maintaining the relation between the domain and the mesh is simplified, and many operations can be performed more naturally through the application of topological adjacencies. The rep-

resentation of general geometric domains typically include loop and shell topological entities, and, in the case of non-manifold models, entity uses for the vertices, edges, loops, and faces [23]. Focusing on the topological representation of the mesh, with the specific topological assumptions given below, each topological mesh entity of dimension d, M_i^d, is bounded by a set of topological mesh entities of dimension $d-1$, $M_i^d\{M^{d-1}\}$. A region is a 3-d entity with a set of faces bounding it. A face is a 2-d entity with a set of edges that bound it. An edge is a 1-d entity with two vertices bounding it. In three dimensions (d=3) the full set of mesh topological entities are:

$$T_M = \{M\{M^0\}, M\{M^1\}, M\{M^2\}, M\{M^3\}\}$$

where $M\{M^d\}$, $d = 0, 1, 2, 3$ are respectively the set of vertices, edges, faces and regions which define the primary topological elements of the mesh domain.

Critical to the understanding of the relationship of the mesh with the geometric domain is the concept of classification of a mesh with respect to its geometric model.

Definition: *Mesh Classification Against the Geometric Model* [5,14,15] - *The unique association of a topological mesh entity of dimension d_i, $M_i^{d_i}$ to a topological geometric model entity of dimension d_j, $G_j^{d_j}$ where $d_i \le d_j$, is termed classification and is denoted $M_i^{d_i} \sqsubset G_j^{d_j}$ where the classification symbol, $\sqsubset$, indicates that the left hand entity, or set, is classified on the right hand entity.*

Multiple $M_i^{d_i}$ can be classified on a $G_j^{d_j}$. A mesh region, M_i^3, is classified in the domain region, G_j^3, in which it lies. A mesh face, M_i^2, is classified in a domain region, G_j^3, or on the domain face, G_j^2, on which it lies. A mesh edge, M_i^1, is classified in a domain region, G_j^3, on the domain face, G_j^2, or on the domain edge, G_j^1, on which it lies. Finally, a mesh vertex, M_i^0, is classified in a domain region, G_j^3, on the domain face, G_j^2, on the domain edge, G_j^1, or on the domain vertex, G_j^0, on which it lies. Mesh entities are always classified with respect to the lowest order object entity possible.

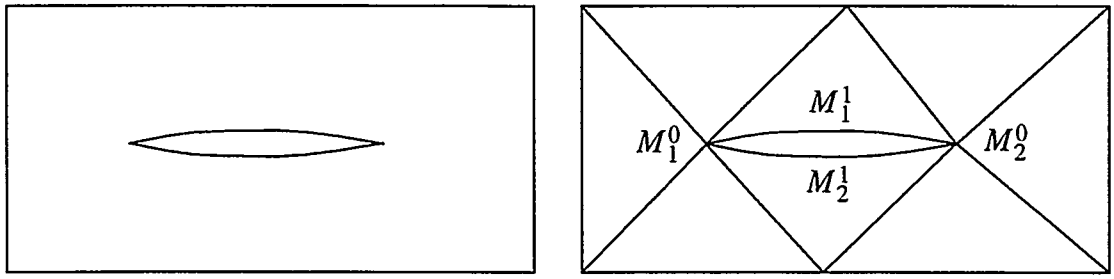
Restrictions on the topology of a mesh which allow the use of only the primary topological entities are [5]:

1. Regions and faces have no interior holes.
2. Each entity of order d_i in a mesh, M^{d_i}, may use a particular entity of lower order, M^{d_j}, $d_j < d_i$, at most once.
3. For any entity $M_i^{d_i}$ there is a unique set of entities of order $d_i - 1$, $M_i^{d_i}\langle M^{d_i-1}\rangle$ that are on the boundary of $M_i^{d_i}$ if at least one member of $M_i^{d_i}\langle M^{d_i-1}\rangle$ is classified on $G_j^{d_j}$ where $d_j \ge d_i$.

The first restriction means that regions may be directly represented by the faces that bound them, and faces may be represented by the edges that bound them. The second restriction allows the orientation of an entity to be defined in terms of its boundary entities (without the introduction of entity uses). For example, the orientation of an edge, M_i^1 bounded by vertices M_j^0 and M_k^0 is uniquely defined as going from M_j^0 to M_k^0 only if $j \ne k$.

The third restriction means that an interior entity (defined as $M_i^{d_i} \sqsubset G_i^{d_j}$ where, $d_j \ge d_i$ and at least one of $\partial(M_i^{d_i}) \sqsubset G_i^{d_j}$) is uniquely specified by its bounding entities. This condition only

applies to interior entities; entities on the boundary of the model may have a non-unique set of boundary entities as illustrated with a model and a coarse mesh of a plate with a hole in Figure 1. Here, the mesh is sufficiently coarse yielding the mesh and model topology identical on the hole boundary. The two mesh edges, M_1^1 and M_2^1, on the hole boundary have the same set of vertices, M_1^0 and M_2^0.



(a) Geometric Model (b) Mesh

Figure 1. Example of mesh entities on the boundary having non-unique boundary entities.

4. Performing Small Feature Removal After Surface Mesh Generation

The original version of the algorithm performed the small feature removal process after a volume mesh of the original model was generated. The advantage of this approach is that it identifies all classes of small geometric model features, ranging from the simple case of a short model edge, to the case of two model faces that come much closer together than the basic element size requested in that vicinity. The disadvantages are that the generation of the initial volume mesh with all small geometric model features still present is complex and computationally expensive, and the mesh modification needed does introduce a reasonable computational cost. Since most of the small features introduce undesirable elements at the surface mesh level, eliminating these elements prior to volume meshing will be computationally cheep and will simplify the volume meshing process.

A careful examination of the mesh validity checks and the dimensional reduction determination checks indicates that, with slight modification, the original set of checks used at the volume mesh level can be used when applying mesh modifications on a surface mesh only. The basic reason that the checks can be performed given only a valid surface mesh is that the surface mesh explicitly separates the various material regions. It therefore provides the information needed to determine local dimensional reductions caused by surface mesh modifications, even in the case of non-manifold geometric models.

An examination of a typical set of solid models with small geometric model features indicated that a majority of the small geometric model features that adversely affect the mesh can be determined from the surface mesh. Mesh modifications can be applied at the surface mesh level to eliminate the influence of these features providing a good surface triangulation to be used by the volume meshing procedure.

There are some small model features that can not be easily detected from the surface mesh. These situations are limited to cases when the distance between non-adjacent portions of the

surface mesh are much smaller than the requested element size. After the volume mesh is generated, the existence of these situations can be determined.

The strategy that has been devised to be a more efficient method to eliminate the influence of small model features from a model is:

1. Generate a valid surface mesh for the original geometric model not allowing added gradation to control element shapes due to the existence of small model features.
2. Eliminate disproportionately small mesh edges in the surface mesh using surface mesh level modification operations, thus providing a good surface mesh for volume meshing.
3. Generate the volume mesh from the resulting surface triangulation.
4. Use mesh modification procedures to eliminate disproportionately small mesh edges in the volume mesh using modification operations.

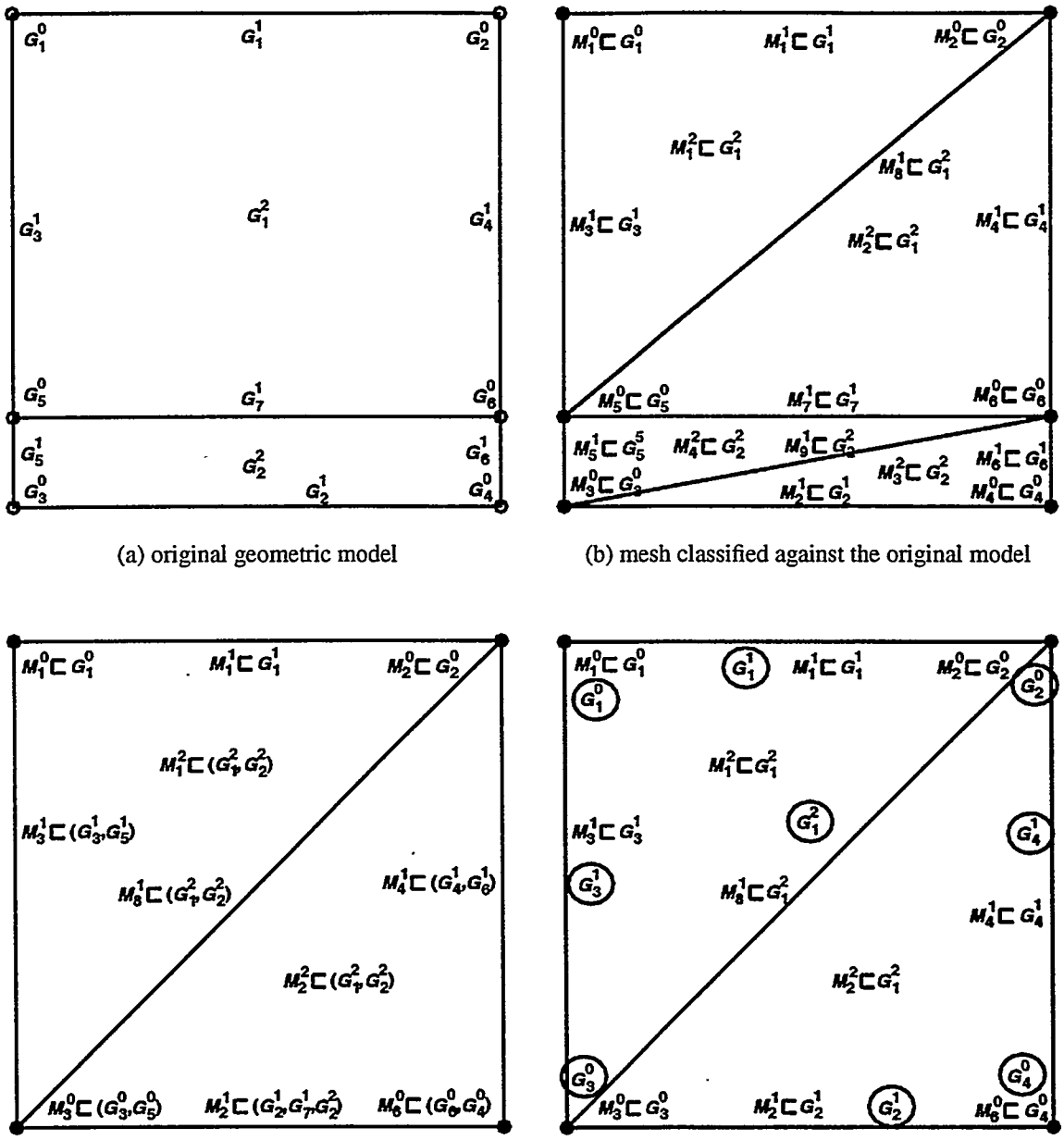
The above strategy assumes that the mesh optimization procedures include split operations that are applied to the surface and volume meshes before attempting to eliminate the influence of small model features. If this is not the case the procedure must employ a general element shape measurement capturing both small and large angles.

Since the adverse influence of most of the small geometric model features are eliminated at the surface mesh level, and providing the volume meshing procedure a good surface mesh everywhere improves the efficiency of the volume meshing process, this approach introduces substantial computational efficiencies.

5. Accounting for Small Feature Removal at the Geometric Model Level

The process of eliminating the influence of a small geometric model feature on a mesh will yield mesh entities which span at least part of multiple geometric model entities. Therefore, they can no longer be classified against the original geometric model. The first approach to address this issue introduced the concept of multiple classification in which the mesh entities are classified against the model entities that they span [7]. Figure 2(a)-(c) demonstrates this process for a simple two-dimensional example. Figure 2(a) shows the geometric model entities while Figure 2(b) shows the mesh generated for that model and indicates the mesh entity classifications. Figure 2(c) shows the mesh after elimination of the small model features and indicates the multiple classifications of the appropriate mesh entities. Although the multiple classification approach does provide the information needed to support operations like adaptive mesh refinement, etc., the multiple classifications approach has two disadvantages. The first is the complication introduced into the data structures and data operators needed to support the multiple classifications. The second is the need to support the multiple classification concepts in all the mesh generation and modification procedures.

The alternative to the multiple classification approach is depicted in Figure 2(d) which shows the final mesh now classified against the modified model representation (the model entities are indicated with the circles around them). The advantage of this approach is that the mesh has a simple classification against a model which avoids complications in the basic mesh generation and modification procedures. It has the second advantage in that it has the potential of supporting geometric modifications as performed by a variety of idealization processes that induce geometric simplifications and/or dimensional reductions [1,2,8,12,20,21,22]. The



(a) original geometric model

(b) mesh classified against the original model

(c) final mesh with multiple classification

(d) final mesh classified on modified model

Figure 2. Association of the mesh and model.

approach presented here focuses its attention on tracking of modifications with respect to the topological representation, which is similar to the virtual topology used in references [16,17, 18,19].

To be an effective tool, a mechanism needs to be defined that can track the modifications so that it is possible during adaptive analysis processes to recover the original model information when needed. Figure 3 depicts a basic approach to addressing this issue. The central idea is a straightforward one in which the topological representation of the original model (linked back to the solid modeler) is maintained, as well as copies of each of the idealized model topologies

for which an independent analysis is performed (that is, a point in the process for which one wants knowledge of the current model representation). Since the size of the topological representation is small compared to the mesh or the shape information in the model, this approach does not dramatically increase the storage requirements.

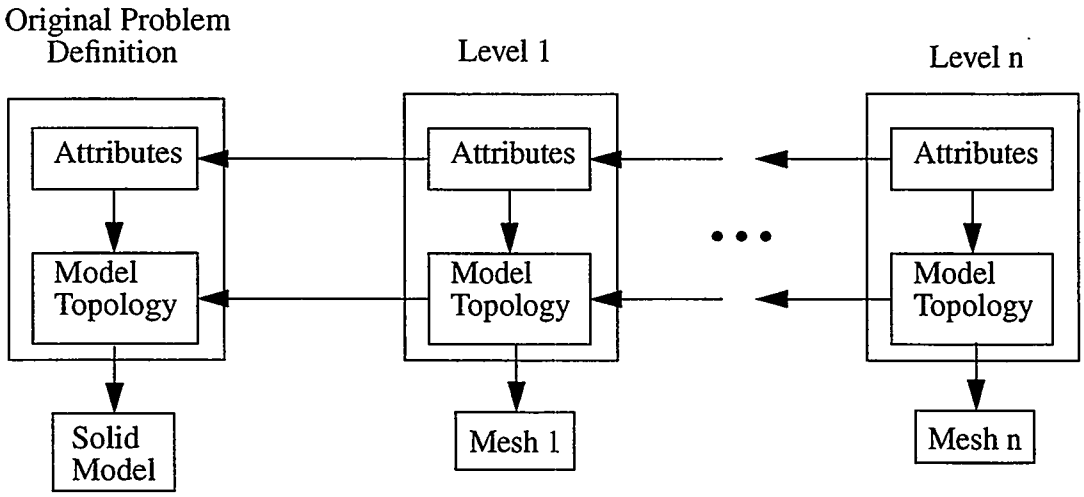


Figure 3. Basic structures for analysis idealization.

In addition to the model topologies, Figure 3 also depicts an attribute structure. This structure is needed to store the idealization process information used to map between the models. For example, in the case of a dimensional reduction it may require storage of a functional form for the "thickness" of the portion of the model that has been reduced. The requirements to support the information needed for a full set of idealization processes has not yet been worked out. The subset that is presented here will support the geometric simplification processes caused by the mesh modification processes needed to eliminate the influence of small model features addressed in the present work where dimensional reductions are not allowed.

The model topology up dates caused by performing mesh modifications employ the current mesh classification information, and knowledge of the base topological modification driving the mesh modification, to key the model topological operators given next.

5.1 Operators to Support the Evolution of the Model and Attribute Structures

The idealization process affects both the model and the attributes defining the next idealization level. To maintain a consistency and understanding between two model levels, the application of the idealization rules must result in a valid boundary representation of the model. In addition, the idealized components of the model must be linked to their original representation in order to be able to reverse the process.

Reduction Operators. A set of operations can be defined to perform simplifications of the model topology. Each of these replaces entities of higher dimension by an entity of lower dimension. The entity that remains in the model stores a reference to the original higher order entity that it replaces. The operators are:

1. **Collapse edge:** Given an edge and one of it's vertices, collapse the edge so that it is represented by the given vertex.
2. **Collapse degenerate face:** Given a degenerate face, which is one with just two edges, and one of its edges, collapse the face so that it is represented by the given edge. A reference to the original face and deleted edge is retained by the surviving edge. Any non-manifold vertices interior to the face are collapsed to one of the vertices of the surviving edge.
3. **Collapse single edge face to vertex:** Given a face that is comprised of one loop with a single edge, collapse the face so that it is represented by a vertex. A reference to the original face is retained by the vertex.
4. **Collapse degenerate region:** Given a degenerate region, which is one with only two faces remaining, and one of it's faces, collapse the region into the given face, deleting the region and its other face. Any non-manifold vertices interior to the region are collapsed to one of the vertices of the surviving face.
5. **Collapse single face region to vertex**: Given a region that is comprised of a single shell consisting of one face, collapse the region to a vertex. A reference to the original region is maintained by the vertex.

Although possible to state the above operators in terms of a set of more basic Euler operations, the above set provides a more convenient means for tracking the model changes.

Careful application of the above set of reduction operators can be used to topologically reduce any model entity to a vertex in the model. Note that in the current case where we need to trace back to the original model entities during adaptive mesh refinement, a vertex is the minimum representation that is possible. If the resulting vertex was allowed to be deleted, there would be no connection back to the original representation of the entities and restoration of the original topology would not be possible.

It is possible to use the reduction operators to eliminate voids in the model by first defining model entities that topologically fill the void and then collapsing those entities.

The history information about each operation is stored on the surviving entity in the model. This history information consists of the operation performed and a list of the entities that were deleted as a result of the operation. If this surviving entity is eliminated in a subsequent operation, the surviving entity from the subsequent operation will refer to it as a part of it's history. Thus, in the process of a series of reductions, the history of the operations form a tree structure that stores all of the information about the deleted entities.

6. Application of Small Feature Removal Procedure

Some geometric modeling systems provide functionalities to automatically stitch small misalignments and gaps that may arise during model construction. Depending on the modeler and how the user performs the modeling operations, these operations will introduce a number of very short model edges and thin model faces. The original implementation of the small feature elimination procedure was focused on these situations [7]. Figure 4 demonstrates a typical situation where the application of the procedure eliminated elements with extremely small angles caused by the small model edges used to stitch the model, increasing the value of the worst angle by over an order of magnitude.

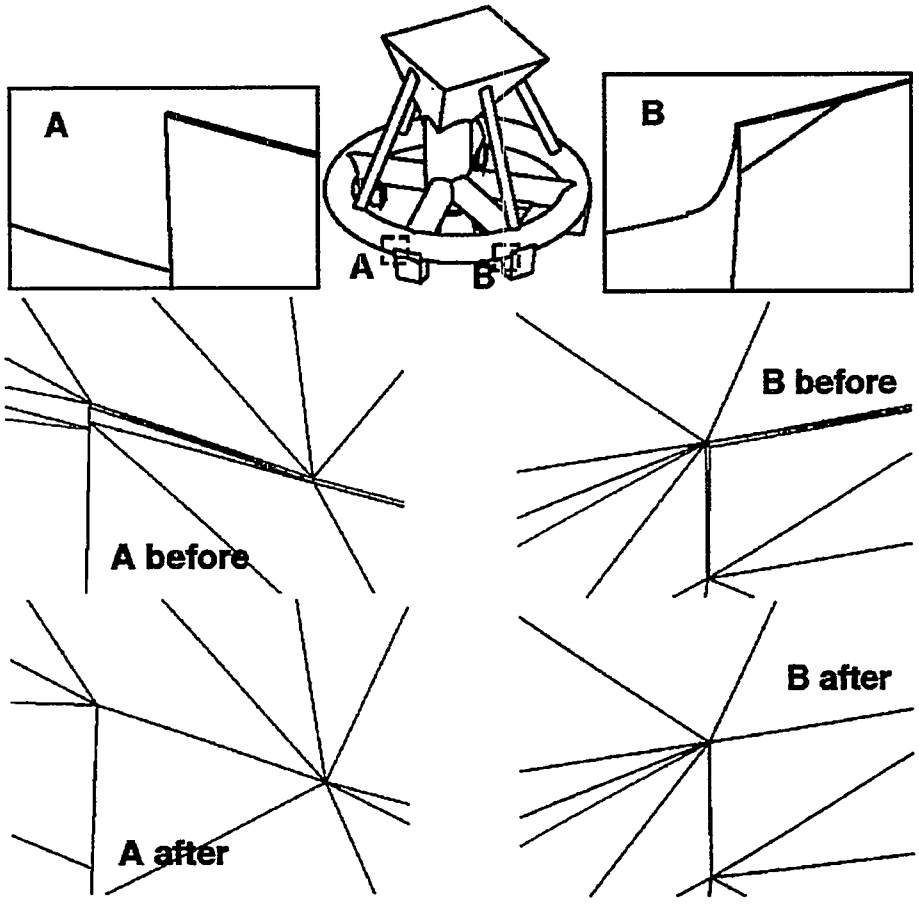
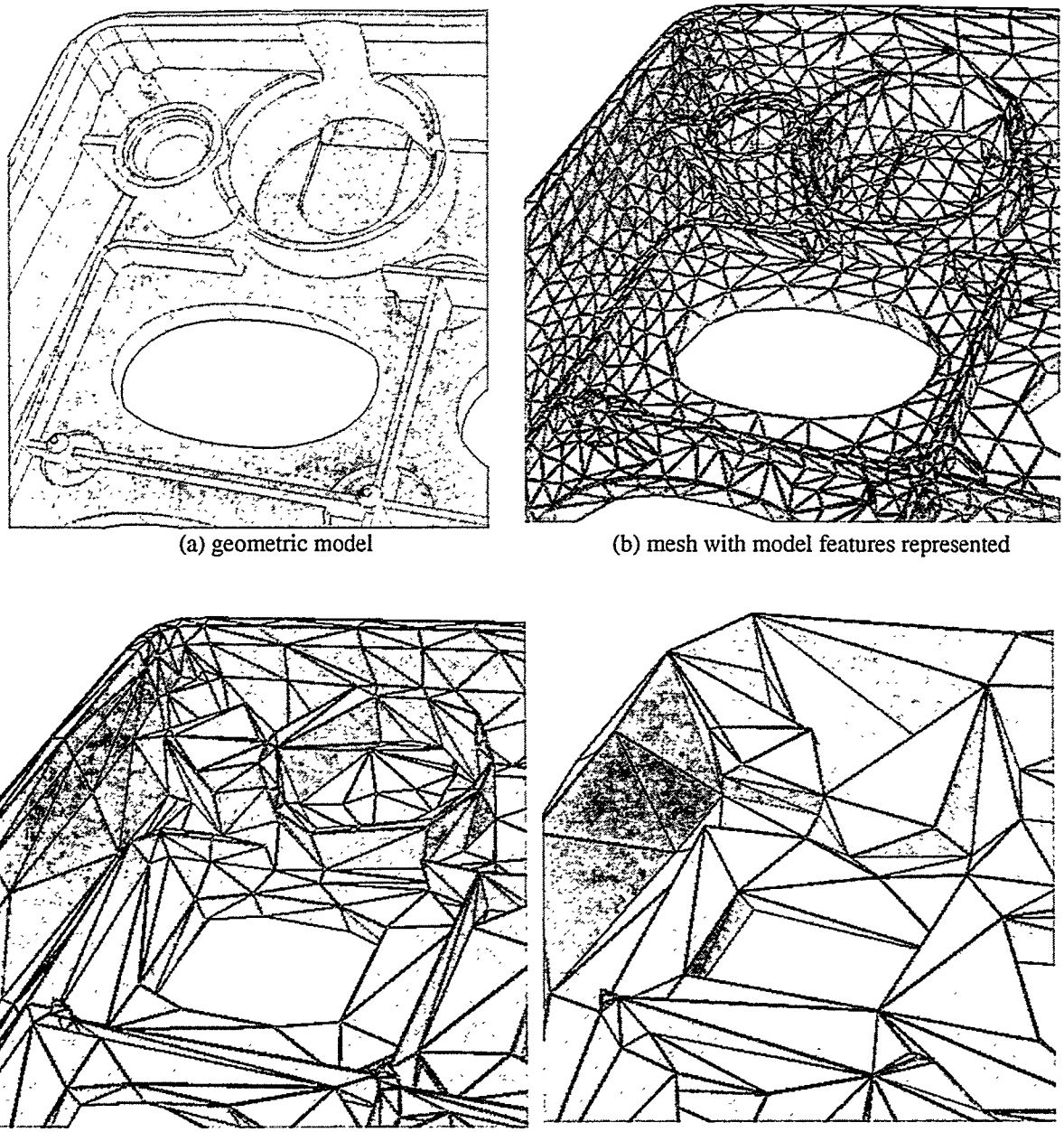


Figure 4. Mesh improvement due to elimination of small model features.

Although originally designed for the elimination of small geometric model features, geometric simplification through mesh modification can be used to perform much more dramatic alterations to a model, thus allowing the mesh to be analyzed to be much coarser than one that explicitly represents all the model features. An example of this usage of the procedure is shown in Figure 5. Figure 5(a) shows a close-up of a portion of a geometric model with a large number of geometric features, while Figure 5(b) shows a mesh generated that explicitly represents all the model features. Figure 5(c) shows a coarser mesh created which includes the application of mesh modifications to eliminate the explicit representation of the smaller model features. A still coarser mesh in which nearly all the local model features are eliminated is shown in Figure 5(d).

7. Closing Remark

This paper has presented improvements to a procedure that can eliminate the influence of small geometric model features on a mesh using mesh modification. Applying the procedure to the surface mesh before the volume mesh is generated improves the efficiency of the procedure. The introduction of the modified topological model representations, replacing the previous multiple classification method, (i) allows the procedure to be used in conjunction with more standard mesh generation procedures and (ii) will allow the integration with a complete



(a) geometric model (b) mesh with model features represented

(c) coarse mesh after mesh modification (d) coarser mesh after mesh modification

Figure 5. Mesh coarsening by the mesh modification with model modification tracking.

set of model modification procedures, some of which may be applied to the geometric model before the mesh is generated.

When used in conjunction with robust mesh generation procedures that can deal with very small model features, the current procedure is effective at automatically eliminating the influence of features that are very difficult to interactively eliminate at the model level and that, is some cases, do not have known algorithms for their automatic elimination at the model level.

The procedures is also quite effective in eliminating large numbers of model features when even those model features and smaller than the requested local element size. This ability pro-

vides an effective means for the generation of coarse meshes on highly detailed geometric models. Of course, the analysis results on such meshes can, at best, provide only overall solution information. Useful information on the influence of the local geometric features requires their explicit geometric representation in the mesh that is analyzed.

8. Acknowledgment

This work was partly supported by the National Science Foundation through Grant number ASC-9318184. The authors would also like to acknowledge the thoughtful comments provided by the reviewers which provided the guidance needed to greatly improve the original version of the paper.

9. References

[1] Armstrong, C.G., "Modeling requirements for finite element analysis", *Computer-Aided Design*, 26(7):573-578, 1994.

[2] Arabshahi, S., Barton, D.C. and Shaw, N.K., "Toward integrated design and analysis", *Finite Elements in Analysis and Design*, 9:271-293, 1996.

[3] E. Bansch, "Local mesh refinement in 2 and 3 dimensions", *Comp. in Science and Engng.*, 3:181-191, 1991.

[4] Butlin, G. and Stops, C., "CAD data repair", *Proc. 5th Int. Meshing Roundtable*, SAND96-3201, Sandia National Laboratories, pp. 219-229, 1995.

[5] Beall, M.W. and Shephard, M.S., "A General Topology-Based Mesh Data Structure", *Int. J. Num. Meth. Engng.*, 40(9):1573-1596, 1997.

[6] de Cougny, H.L. and Shephard, M.S., "Parallel Refinement and Coarsening of Tetrahedral Meshes", submitted 1998.

[7] Dey, S., Shephard, M.S. and Georges, M.K., "Elimination of the Adverse Effects of Small Model Features by Local Modifications of Automatically Generated Meshes", *Eng. with Computers*, 13(3):134-152, 1997.

[8] Donaghy, R.J., McCune, W., Bridgett, S.J., Armstrong, C.G., Robinson, D.J. and McKeag, R.M., "Dimensional Reduction of Analysis Models", *Proc. 5th Int. Meshing Roundtable*, SAND96-3201, Sandia National Laboratories, pp. 307-320, 1995.

[9] Jones, M.J., Price, M.A. and Butlin, G., "Geometry management support for auto-meshing", *Proc. 4th Int. Meshing Roundtable*, SAND95-2130, Sandia National Laboratories, pp. 153-164, 1995.

[10] Liu, A. and Joe, B., "Quality local refinement of tetrahedral meshes based on bisection", *SIAM J. Sci. Comput.*, 16:1269-1291, 1995.

[11] Price, M.A., Stops, C. and Butlin, G., "A medial axis toolkit for meshing and other applications", *Proc. 4th Int. Meshing Roundtable*, SAND95-2130, Sandia National Laboratories, pp. 219-229, 1995.

[12] Rezayat, M., "Midsurface abstraction from 3d solid models", *Computer-Aided Design*, 28(11): 904-915, 1996.

[13] Rivara, M.-C., "A 3-D refinement algorithm suitable for adaptive and multi-grid techniques", *Comm. in Applied Num. Meth.*, 8:281-290, 1992.

[14] Shephard, M.S. and Georges, M.K., "Reliability of Automatic 3-D Mesh Generation", *Comp. Meth. Appl. Mech. and Engng.*, 101:443-462, 1992.

[15] Schroeder, W.J. and Shephard, M.S., "On Rigorous Conditions for Automatically Generated Finite Element Meshes", *Product Modeling for CAD and Manufacturing*. ed. J. U. Turner J. Pegna and M. J. Wozny. p. 267-281. North Holland, Amsterdam, 1991.

[16] Sheffer, A., Blacker, T.D. and Bercovier, M., "Toward the solution of fundamental issues in CAD-FEM Integration", *Computational Mechanics: New Trends an Applications*, S. Idelsohn, E. Onate and E. Dvorkin, EDS, CIME, Barcelona, Spain, CD Subject entry 7-1-14, 1998.

[17] Sheffer, A., Blacker, T.D., Clements, J. and Bercovier, M., "Virtual topology construction and application", *Geometric Modeling Theory and Practice*, Springer-Verlag, pp. 247-259, 1997.

[18] Sheffer, A., Blacker, T.D. and Bercovier, M., "Virtual topology operators for meshing", *Proc. 6th Int. Meshing Roundtable*, Sandia Nat. Lab., pp. 49-66, 1997.

[19] Sheffer, A., Blacker, T.D. and Bercovier, M., "Clustering: automatic detail supression using virtual topology", *Trends in Unstructured Mesh Generation*, ASME, AMD-Vol. 220, pp. 55-65, 1997.

[20] Shephard, M.S., "Finite element modeling within an integrated geometric modeling environment: Part II - Attribute specification, domain differences, and indirect element types", *Engineering with Computers*, 1:72-85, 1985

[21] Shephard, M.S., Korngold, E.V., Collar, R.R. and Baehmann, P.L., "A Modeling Framework for Controlling Structural Idealizations in Engineering Design", *Computers & Structures*, 37(2):181-191, 1990.

[22] Storti, D.W., Turkiyyah, G.M., Ganter, M.A., Lim, C.T. and Stal, D.M., "Skelton-based modeling operations on solids", *Proc. 1997 4th Symp. on Solid Modeling and Applications*, ACM, NY, New York, p. 141-154, 1997.

[23] Weiler, K.J., "The radial-edge structure: a topological representation for non-manifold geometric boundary representations", *Geometric Modeling for CAD Applications*, M.J. Wozny, H.W. McLaughlin, J.L. Encarnacao, editors, North Holland, 3-36, 1988.

Automatic Grid Generation with Almost regular Delaunay Tetrahedra

Alexander Fuchs*
Department of Mathematics A
University of Stuttgart
Pfaffenwaldring 57, D-70569 Stuttgart, Germany
Email: fuchs@mathematik.uni-stuttgart.de

Abstract. *An algorithm for constructing almost regular triangulations (ARTs) for three-dimensional polygonal domains is described. Ideally such triangulations consist entirely of congruent tetrahedra with nearly equal edges. The new feature of this method is that the position of the vertices is adjusted, before any connecting edges are assigned. This leads to grids with very few irregular vertices, i.e. most interior vertices are shared by 24 tetrahedra as for regular tessellations of $\mathbb{R}^3$ with congruent tetrahedra.*

keywords. delaunay tetrahedra, tessellation of $\mathbb{R}^3$, almost regularity

1 Introduction

Grid generation algorithms are essential tools in many areas of modern applied mathematics. Typical examples are finite element methods (FEM, [2]) and geometric modelling (CAGD, [4]). In both applications the quality of the grids strongly influences the accuracy of numerical computations. For example, irregularly shaped tetrahedra result in large condition numbers of the stiffness matrices for Galerkin methods. Similarly, spline approximation on irregular partitions requires non standard smoothness conditions [18, 15, 14, 5].

Ideally, a triangulation should consist entirely of equilateral tetrahedra. Unfortunately, this is not possible. However, as will be shown in section 2, there exist congruent tetrahedra having nearly equal edges. These tessellations have a regular lattice structure, i.e. every vertex has 14 neighbours and is shared by 24 tetrahedra.

While such regularity can seldom be achieved it is natural to insist on "almost regularity". An almost regular triangulation (ART) should have only few irregular vertices, i.e. most of the interior vertices are shared by exactly 24 tetrahedra. Such combinatorial optimization also leads to good geometric properties. Clearly for an ART smoothing of the vertices usually results in very small deviations of the angles from the optimal value. Figure 1 shows an example. The triangulation consists of 860 vertices and has just 183 irregular vertices. The percentage of angles less than 12.0° is lower than 0.12% and the maximum angle is 168.2°.

Offhand the construction of ARTs seems rather difficult since the combinatorial requirements may result in global constraints on the tetrahedra. We therefore employ an indirect approach, and extend our 2D-algorithm (see [10]) to 3D. We first construct a set of inner vertices according to a density function ϱ. This function is defined over the entire domain and describes the edge-length of the triangulation. In the second step we optimize the position of the vertices without assigning any connecting edges. Then, we form a Delaunay triangulation. More precisely, our algorithm for constructing an ART of a polygonal domain consists of three steps:

(ART1) construct an initial configuration and scale the density function,

(ART2) adjust the position of the vertices by minimizing a penalty functional,

(ART3) form a Delaunay triangulation.

As usual, the density function controls the size of the tetrahedra which may vary throughout the domain. We do not discuss its construction here, which may depend on the geometry as well as on the local error of finite element calculations. The functional in the second step is minimal for regular triangulations. The optimization will therefore yield an almost regular configuration of the vertices. After these preprocessing steps the Delaunay triangulation gives excellent results, although the geometric advantages of two-dimensional Delaunay ([19],[17]) do not hold for three dimensions.

*supported by SFB 404, University of Stuttgart

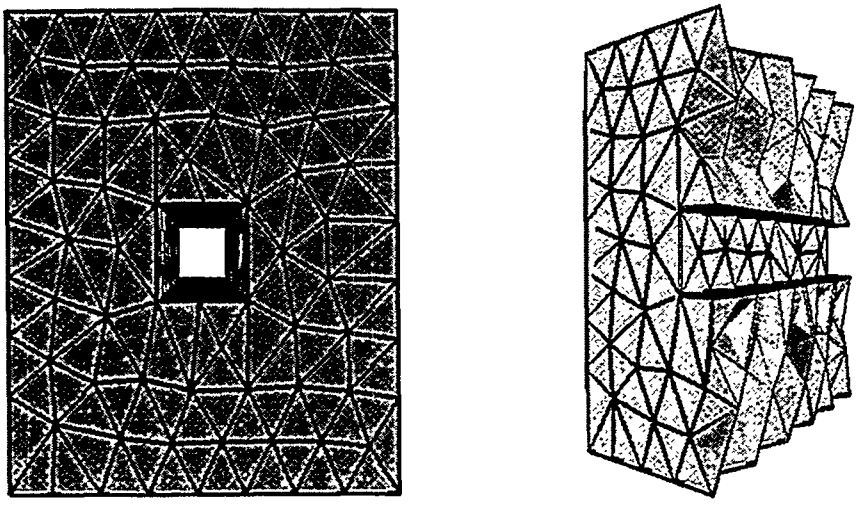
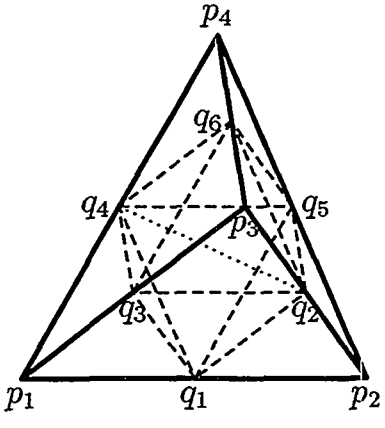


Figure 1: Example of an ART

In sections 3–5 the steps of the ART-Algorithm and the underlying theory will be described in detail. In section 6 we discuss a number of examples illustrating the performance of our method. We conclude with some remarks on possible extensions.

2 Regular Tessellations with Tetrahedra

A regular triangulation is a partition of $\mathbb{R}^3$ into congruent tetrahedra T_i, so that their intersection is either empty or a vertex, edge or face. Unlike in two dimensions, there exists no canonical regular triangulation since one cannot partition $\mathbb{R}^3$ with equilateral tetrahedra. Of course for applications in finite element methods and geometric modelling, the tetrahedra should be nearly equilateral. Another natural requirement is invariance under subdivision. If a tetrahedron T is split into 8 tetrahedra by halving the edges as is indicated in Figure 2, the small tetrahedra should by congruent to T scaled by the factor $\frac{1}{2}$. Subdivision invariance is particularly important for grid refinement. Moreover, it is easy to show that a subdivision invariant tetrahedron generates a regular triangulation of $\mathbb{R}^3$. One should note, however, that neither an equilateral tetrahedron nor the one produced by the tessellation of the standard cube (corners (0,0,0),(1,0,0), (1,1,0) and (1,1,1)) are subdivision invariant.



T_1 : p_1, q_1, q_3, q_4
T_2 : p_2, q_1, q_2, q_5
T_3 : p_3, q_2, q_3, q_6
T_4 : p_4, q_4, q_5, q_6
T_5 : q_1, q_2, q_4, q_5
T_6 : q_1, q_2, q_4, q_3
T_7 : q_2, q_4, q_6, q_5
T_8 : q_2, q_4, q_6, q_3

Figure 2: Subdivision of a tetrahedron T.

With the following theorem we characterize all subdivision invariant tetrahedra.

Theorem 1 *A tetrahedron in standard coordinates with vertices*

$$p_1 = 0,\ p_2 = v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix},\ p_3 = v_2 = \begin{pmatrix} x_1 \\ y_1 \\ 0 \end{pmatrix} \text{ and } p_4 = v_3 = \begin{pmatrix} x_2 \\ y_2 \\ z \end{pmatrix},$$

having $x_1, x_2, y_2 \geq 0$ and $y_1, z > 0$ (cf. Figure 2), is subdivision invariant, iff one of the following 4 conditions is fulfilled:

Condition 1: $0 \leq x_1 < \frac{1}{2}$

$x_2 = 2x_1 \qquad y_1^2 = 1 - x_1^2$

$y_2^2 = \dfrac{(1+x_1)(1-2x_1)^2}{1-x_1} \qquad z^2 = \dfrac{(1+x_1)(1-2x_1)}{1-x_1}$

Condition 2: $y_2 > 0$

$x_1 = \frac{1}{3} \qquad x_2 = \frac{2}{3}$

$y_1 = 2y_2 \qquad z = \sqrt{3}y_2$

Condition 3: $0 \leq x_1 < \frac{1}{2}$

$x_2 = 1 - x_1 \qquad y_1^2 = 1 - x_1^2$

$y_2^2 = \dfrac{x_1^2(1+x_1)}{1-x_1} \qquad z^2 = \dfrac{(1+x_1)(1-2x_1)}{1-x_1}$

Condition 4: $0 < x_1 < 3$

$x_2 = \frac{1}{2}(x_1+1) \qquad y_1^2 = x_1(3-x_1)$

$y_2^2 = \frac{1}{4}x_1(3-x_1) \qquad z^2 = \frac{1}{4}(3-x_1)$

Proof. It is easy to show that tetrahedra T_1, T_2, T_3 and T_4 are congruent to $\frac{1}{2}T$. In the interior of T there are 2 different tetrahedra which have to be mapped to the tetrahedron T_1 by suitable Euclidean movements. This results in the conditions 1 to 4. The explicit computation of the congruence mappings is straightforward but rather tedious. Since the arguments are not crucial for the subsequent development of the grid generation algorithm we refer to [9]. ∎

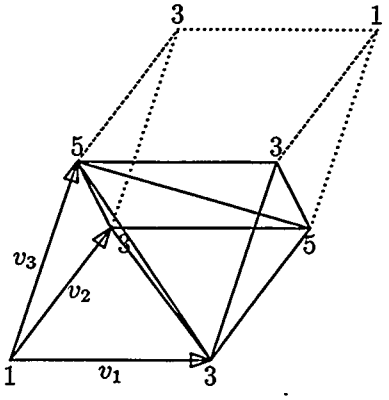


Figure 3: Tetrahedra forming a parallelepiped.

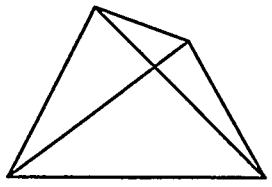


Figure 4: Tetrahedron T_*.

Any of the tetrahedra defined by Theorem 1 gives rise to a regular triangulation. Analyzing the structure of such triangulations in more detail, we first notice that 6 tetrahedra form a parallelepiped as shown in Figure 3. The labels at the vertices in Figure 3 refer to the number of tetrahedra sharing this vertex. Moreover, we see that 24 tetrahedra meet in a regular vertex of a partition with parallelepipeds and each regular vertex has 14 neighbours.

As a prototype for our grid generation algorithm we choose a subdivision invariant tetrahedron which differs from an equilateral tetrahedron as little as possible. Figure 5 shows the standard deviation of the dihedral angles of the tetrahedra satisfying the conditions of Theorem 1 as a function of the free parameters. We select a tetrahedron T_* satisfying condition 1 with $x_1 = 1/3$ (see Figure 5).

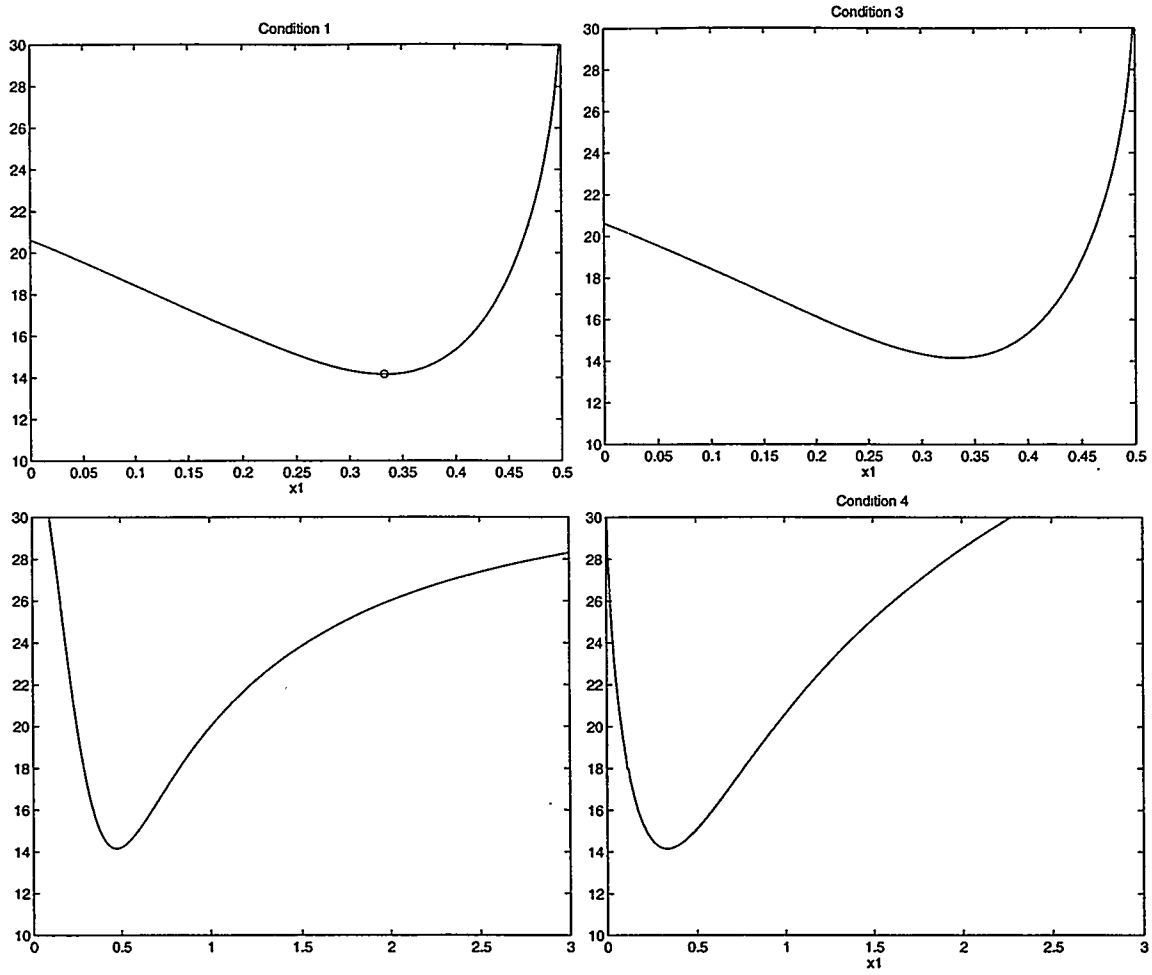


Figure 5: Standard deviation of dihedral angles

T_* has 3 congruent faces with angles 54.74°, 54.74°, 70.52° and the standard deviation to 60° is $\approx 7.44°$. The lengths of the edges of the tetrahedron T_* are 1, 1, 1, 1, $2/\sqrt{3} \approx 1.15$, $2/\sqrt{3}$ with a standard deviation of $(2-\sqrt{3})/3 \approx 0.089$ from the equilateral case, and the dihedral angles are 60°, 60°, 60°, 60°, 90°, 90°, having a standard deviation of 14.15° from 70.53°, the dihedral angle for the equilateral tetrahedron.

3 Construction of an initial Configuration

We can obtain an initial configuration with the following simple algorithm. We choose an approximate centre C_0 of the domain and construct the polyhedron formed by the 24 tetrahedra sharing C_0 (see Figure 6). Now we scale the polyhedron, until it covers the entire domain Ω. Then we divide the tetrahedra which, according to the density function, are too large and repeat this, until all tetrahedra are small enough. For a constant density function, we get a fully regular triangulation. Finally we project points outside the domain, which have connection to an inner point, on the boundary of Ω. All points within and on the boundary of the domain are then used as an initial set of vertices. At this point the exact position of the points on the boundary is not crucial, since no connecting edges are assigned in the subsequent optimization procedure.

The number of vertices obtained in this way will, in general, not exactly match the density function. Since the density function plays an important role in the following optimization process (cf. Section 4), we have to rescale the density

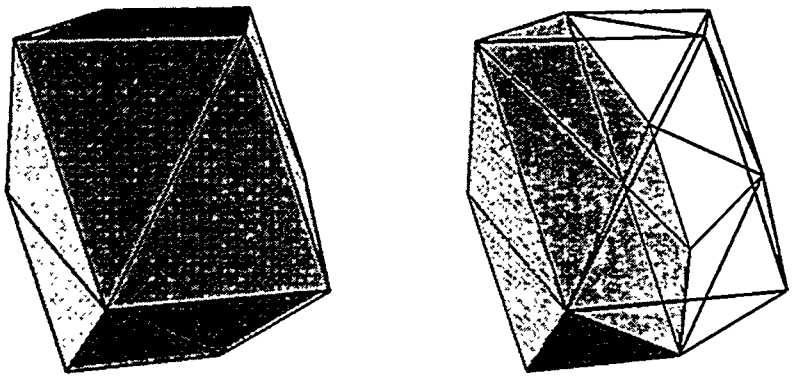


Figure 6: Polyhedron, formed by 24 tetrahedra (solid and upper half transparent)

function $\varrho \to s \cdot \varrho$ ($s \approx 1$) with an appropriate scaling factor s, in order to match the number of the constructed vertices best. Therefore, we have to analyse the relation between the density function and the number of vertices. The polygonal domain Ω to be triangulated can be multiply connected and is specified by the edges and vertices making up the boundary polygons. For example the domain in Figure 7 has 2 boundary polyhedrons with a total of 24 edges and 16 vertices.

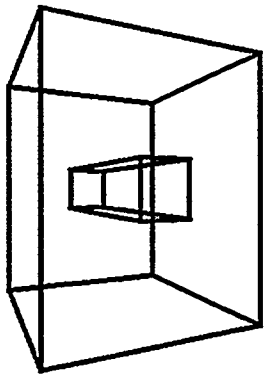


Figure 7: Polygonal Domain

To estimate the number of interior vertices X_j, $j = 1, \ldots, n_i$ in terms of the density function ϱ, we assume that the triangulation is regular, i.e. that 24 tetrahedra share an interior and 12 tetrahedra share a flat boundary vertex. The number of tetrahedra sharing a vertex on a boundary edge varies as can be seen in Figure 3. But in general all "types" of edge vertices occur with the same frequency, so we assume that 6 tetrahedra share an edge vertex. Denoting by n_i, n_b and n_e the number of interior, boundary and edge vertices of the triangulation, it follows that the total number of tetrahedra is

$$n_t = (24n_i + 12n_b + 6n_e)/4. \tag{1}$$

From Figure 3 it can be seen that 6 triangles share a boundary vertex and an edge vertex. It follows that the number of triangles on the boundary of the domain is

$$n_f = (6n_b + 6n_e)/3. \tag{2}$$

Hence we obtain for the number of vertices $n = n_i + n_b + n_e$ the formula

$$n = \frac{1}{6}n_t + \frac{1}{4}n_f + \frac{1}{4}n_e. \tag{3}$$

Now we relate n_t and n_f to the density function ϱ, dropping our assumption that the triangulation is regular. The length of the edges should be approximately equal to the value of ϱ at their midpoints. If we denote by V_0 the volume of the tetrahedron T_* (cf. Section 2), a regular triangulation with constant density function yields

$$n_t \cdot V_0 \cdot \varrho^3 = \mathrm{vol}(\Omega),$$

and we can compute n_t in terms of the volume of the domain. For non constant ϱ we use the approximation

$$\int_\Omega \frac{1}{\varrho^3} \approx \sum_\tau \mathrm{vol}(\tau) \frac{1}{\varrho_\tau^3} \approx \sum_\tau V_0 = V_0 n_t, \tag{4}$$

where the sums are taken over all tetrahedra τ of the triangulation and ϱ_τ denotes the average value of ϱ on τ. In order to relate n_f to the density function, we note that there may exist boundary triangles with different areas, depending on the tetrahedron T_* chosen (for our choice all boundary triangles have the same area). Since in general all triangles should occur with roughly the same frequency we have

$$\int_{\partial\Omega} \frac{1}{\varrho^2} \approx F_0 n_f, \tag{5}$$

where F_0 is the average area of the boundary triangles. Similarly we estimate the number of edge vertices by

$$\int_{\partial\partial\Omega} \frac{1}{\varrho} \approx E_0 n_e, \tag{6}$$

denoting by E_0 the average edge length of the tetrahedron T_*.

Combining equations (3), (4), (5) and (6) we obtain for the number of vertices

$$n \approx \frac{1}{6V_0} \int_\Omega \frac{1}{\varrho^3} + \frac{1}{4F_0} \int_{\partial\Omega} \frac{1}{\varrho^2} + \frac{1}{4E_0} \int_{\partial\partial\Omega} \frac{1}{\varrho}. \tag{7}$$

We can now use equation (7) to scale the density function, $\varrho \to s \cdot \varrho$, so that equation (7) is maintained for the number of vertices constructed with our algorithm.

This estimate was derived for regular triangulations only. However, it remains fairly accurate for an ART, in particular when the number of irregular vertices is small. In any case, a slight deviation from the optimal number of points has no major impact on the quality of the triangulation as will become apparent from the next section. Merely the length of the edges will not correspond exactly to the density and the triangulation might be slightly distorted at the boundary.

4 Adjusting the Vertices

Adjusting the vertices to improve the geometric characteristics of a triangulation is a standard procedure [1, 6]. Unlike to smoothing in previous work (cf. e.g. [7, 8, 16]), the new feature of our approach is to we optimize the position of the vertices, before assigning any connections to them. There associated Delaunay triangulation should be almost regular. This can be achieved by minimizing a functional f which penalizes configurations producing irregular vertices. The choice of our functional is based on the following conjecture.

A vector H is a permutation of D ($H, D \in \mathbb{R}^n$) if the equations

$$\begin{aligned} (h_1 + \cdots + h_n)^{1/1} &= (d_1 + \cdots + d_n)^{1/1} \\ (h_1^2 + \cdots + h_n^2)^{1/2} &= (d_1^2 + \cdots + d_n^2)^{1/2} \\ &\vdots \\ (h_1^n + \cdots + h_n^n)^{1/n} &= (d_1^n + \cdots + d_n^n)^{1/n} \end{aligned} \tag{8}$$

hold.

A proof for this assumption has yet not been found, but we can prove the following stronger theorem.

Theorem 2 *A vector H is a permutation of a vector D iff the equations*

$$\begin{aligned}(h_1+\cdots+h_n)^{1/1} &= (d_1+\cdots+d_n)^{1/1}\\ (h_1^2+\cdots+h_n^2)^{1/2} &= (d_1^2+\cdots+d_n^2)^{1/2}\\ &\vdots\\ (h_1^{2n}+\cdots+h_n^{2n})^{1/(2n)} &= (d_1^{2n}+\cdots+d_n^{2n})^{1/(2n)}\end{aligned} \tag{9}$$

hold.

To proof this, we need the following lemma.

Lemma 1 *Equations (9) imply for every polynom $p(x)$ of degree $2n$*

$$p(h_1)+p(h_2)+\cdots+p(h_n)=p(d_1)+p(d_2)+\cdots+p(d_n)\ . \tag{10}$$

Proof. Let $p(x)=a_{2n}x^{2n}+a_{2n-1}x^{2n-1}+\cdots+a_0$ be the monomial representation of the polynom, then the following holds

$$\begin{aligned}&p(h_1)+\cdots+p(h_n)\\ &= a_{2n}(h_1^{2n}+\cdots+h_n^{2n})+a_{n-1}(h_1^{2n-1}+\cdots+h_n^{2n-1})+\cdots+n\cdot a_0\\ &= a_{2n}\left((h_1^{2n}+\cdots+h_n^{2n})^{1/(2n)}\right)^{2n}+a_{n-1}\left((h_1^{2n-1}+\cdots+h_n^{2n-1})^{1/(2n-1)}\right)^{2n-1}+\cdots+n\cdot a_0\\ &\quad \text{(because of (9))}\\ &= a_{2n}\left((d_1^{2n}+\cdots+d_n^{2n})^{1/(2n)}\right)^{2n}+a_{n-1}\left((d_1^{2n-1}+\cdots+d_n^{2n-1})^{1/(2n-1)}\right)^{2n-1}+\cdots+n\cdot a_0\\ &= a_{2n}(d_1^{2n}+\cdots+d_n^{2n})+a_{n-1}(d_1^{2n-1}+\cdots+d_n^{2n-1})+\cdots+n\cdot a_0\\ &= p(d_1)+\cdots+p(d_n)\end{aligned}$$

and thus (10) holds. ∎

Having this lemma, the proof of Theorem 2 is quite simple.

Proof of Theorem 2.

"⇒" : If H is a permutation of D, then the equations (9) are trivially fulfilled.

"⇐" : (complete induction)
For $n=1$ we have $h_1=d_1$, and thus H a permutation of D.
Now use Lemma 1 with the polynom $p(x):=(x-d_1)^2\cdot(x-d_2)^2\cdots(x-d_n)^2$. Then for a solution of equations (9) we have $p(h_1)+\cdots+p(h_n)=p(d_1)+\cdots+p(d_n)=0$. Because $p(x)\geq 0$ it has to be $p(h_i)=0$ and thus $\forall h_i\exists d_j : h_i=d_j$.
Without loss of generality let now be $h_n=d_n$, then we can reduce the dimension of (9) by one. ∎

Now we can define a functional, which penalizes the error of equations (8). We denote by $d_1\leq d_2\leq\cdots$ the distances of the vertices of a regular triangulation to a fixed vertex. For an almost regular triangulation, the corresponding distances $\|X_i-X_k\|$, $k=1,2,\ldots,n$ should be close to a permutation of D. This means that

$$\left|\left(\sum_{\substack{k=1\\k\neq i}}^{n}\|X_i-X_k\|^j\right)^{1/j}-\left(\sum_{\substack{k=1\\k\neq i}}^{n}d_k^j\right)^{1/j}\right|$$

is small for each i and j. This motivates the following definition of our penalty functional f:

$$f(X_1,\cdots,X_n) := \sum_{i=1}^{n}\sum_{j=1}^{m}\left(\left(\sum_{\substack{k=1\\k\neq i}}^{n}\varphi(h_{i,k})^j\right)^{1/j} - \left(\sum_{\substack{k=1\\k\neq i}}^{n}\varphi(d_k)^j\right)^{1/j}\right)^2, \quad h_{i,k} := \frac{\|X_i - X_k\|}{(\varrho(X_i)+\varrho(X_k))/2}.$$

We have replaced $\|X_i - X_k\|$ by the normalized distance $h_{i,k}$ which is a good approximation to the corresponding component of D for neighbouring points since $(\varrho(X_i) + \varrho(X_j))/2$ describes the average side length; $h_{i,k}$ is approximately equal to the number of tetrahedra between the i-th and k-th vertex. For a constant density ϱ, which results in an uniformal spacing of the vertices, this replacement is just a normalization of the distances. If we have a non constant density function, which results in a non-uniformal spacing of the vertices, then $h_{i,k}$ normalizes the distances of neighbouring vertices in the lattice. Of course, small distances are most relevant. Therefore, we introduce a cut-off function φ which vanishes for arguments larger than d_l.

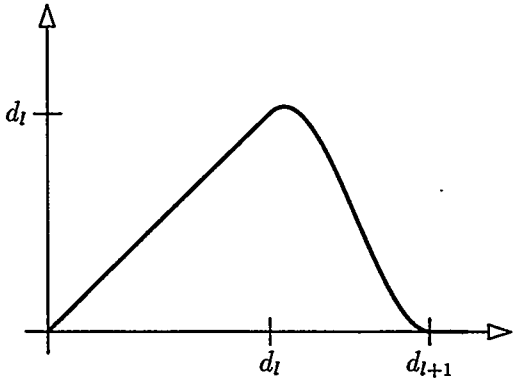


Figure 8: Obvious but poor choice of φ.

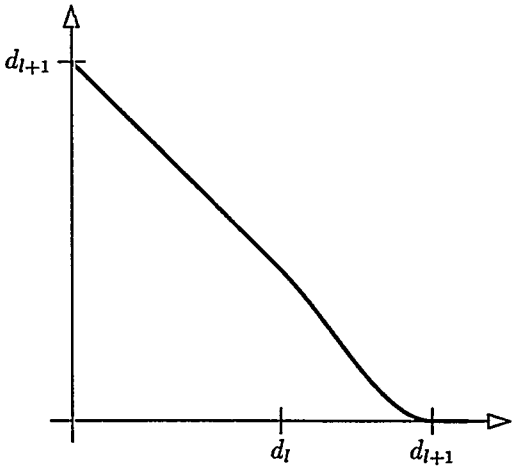


Figure 9: Good choice of φ.

The obvious choice of φ is shown in Figure 8. The drawback of this choice is, that we cannot distinguish a distance in the interval $[d_l\ d_{l+1}]$ and $[d_l\ 0]$. Therefore we prefer another (monotone) choice of φ as shown in Figure 9. We notice that we only have to compute $l+1$ distances in vector D, because $\varphi(x) = 0$ for $x \geq d_{l+1}$. In practice it suffices to take one layer of neighbouring vertices, so that $l = 14$. For our choice of tetrahedron T_* we have $D^2 = [d_1,\cdots,d_{l+1}] = [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 4/3\ 4/3\ 4/3\ 4/3\ 4/3\ 4/3\ 8/3]$.

Because of the cut-off function $\varphi(\cdot)$ most of the terms in equations (8) are zero. Therefore it makes sense not to take the sum over $j = 1,\cdots,n$ and to introduce a cut-off parameter m, having $l \leq m \leq n$. In practice we get good results for $m = 2 \cdot l$.

One notices that, because of the local character of the 'cut-off' function φ, the sum over k will contain only few nonzero terms. Therefore, with an appropriate data structure, the function f can be computed in $O(n)$ operations. One also has to take into account, that the vector D differs for vertices inside and at the boundary of Ω, thus the vertices X_i have to be split into 2 sets.

Of course there is great deal of heuristic in our choice of the penalty functional f. Strictly speaking, our analysis is valid only for a regular grid which is compatible with the boundary. However, because of the local character of φ, one can expect that f will be nearly minimal for ARTs if the density function is smooth. In fact, in all numerical tests (cf. Section 6) we have obtained very good results.

The numerical implementation of the minimization procedure is straightforward from a mathematical point of view. We employ a conjugate gradient method, and project the "search-direction" for vertices on $\partial\Omega$ on the boundary. The actual C++-Code ($\geq$ 35000 lines) is rather complex. More generally than described in this paper, the algorithm can handle free an fixed vertices on the boundary as well as internal constraints. A number of additional strategies are employed to improve the efficiency of the program. To speed up the convergence of the iteration we can compute

first a coarse triangulation and obtain improved start configurations via subdivision. Also, we can parallelize most of the computations. Figure 10 shows a few typical steps of the smoothing process.

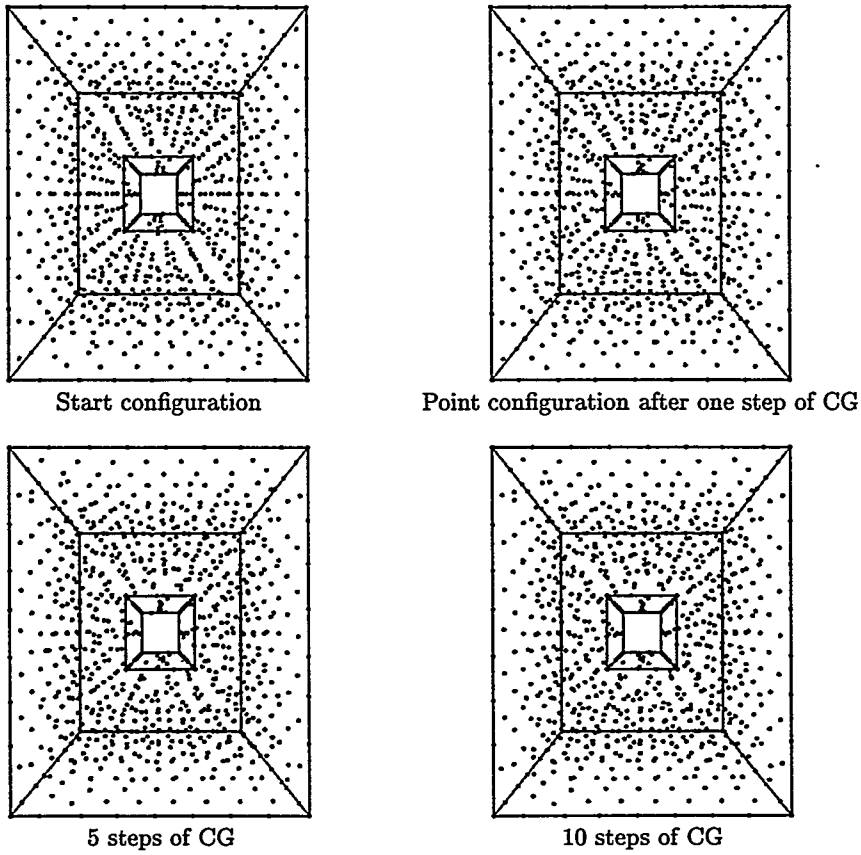


Figure 10: Smoothing Process

5 Triangulation

A Delaunay tessellation can be obtained as dual graph of the Voronoi-diagram, which consists of the sets of closest points to a given set of vertices [19]. To determine the Delaunay tetrahedra we use the following criterion.

Definition 1 (Delaunay Tetrahedron) *The vertices P,Q,R,S form a Delaunay tetrahedron, if there is no further vertex in its circumscribed ball.*

Methods for a sequential construction of the Delaunay tetrahedra sequentially are well known (cf. e.g. [19]). Since our vertices are constructed with respect to a density function, we can implement an efficient parallel algorithm. This algorithm will be described in detail in [11]. In order to keep the resulting triangulation conform to the boundary, we use a constrained Delaunay triangulation (cf. e.g. [3]). Of course, the placement of the edges is not essential on smooth parts of the boundary, where any polygon is just an approximation.

For the example considered in the previous section, Figures 11 and 12 show the resulting Delaunay triangulation.

For complex domains or domains with a lot of interior faces we can construct a density function, which "recognizes" the local complexity of the domain, and adapts the density. This has already been done for two dimensions (cf. Figure 13) and will be described for three dimensions in a forthcoming paper.

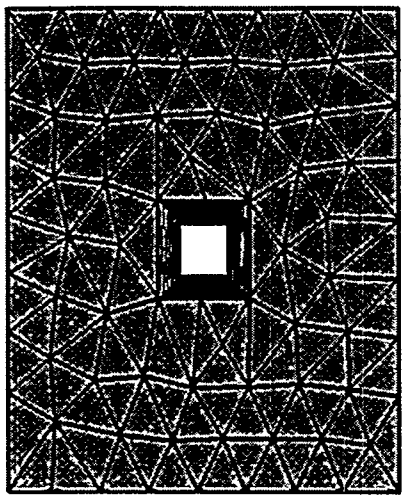


Figure 11: View from above

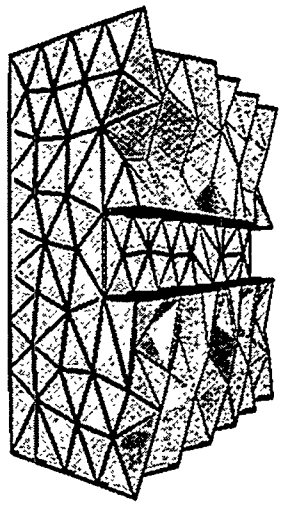
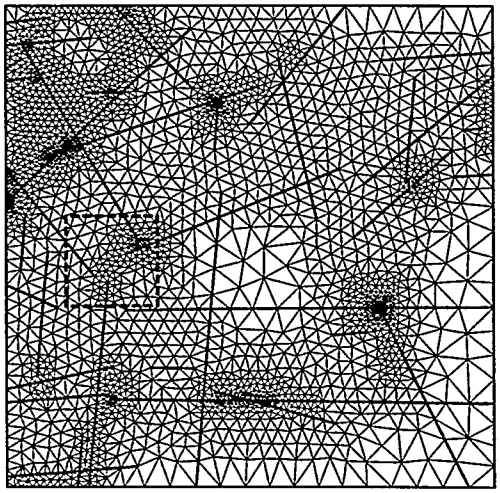
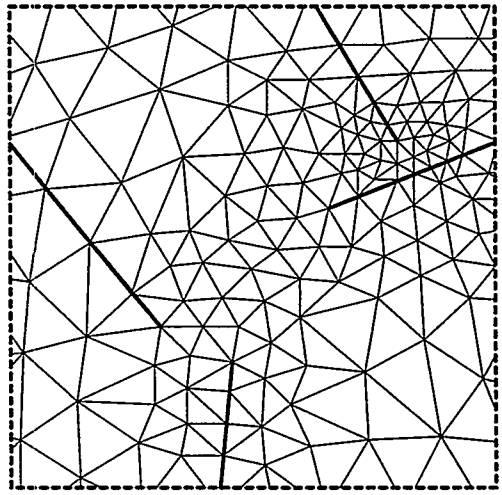


Figure 12: Cut through the domain



2d-net



zoomed net

Figure 13: Two dimensional net with interior edges and automatically adapted density

6 Examples

In the following we discuss various examples, illustrating the performance of our algorithm. For evaluating the quality of the results we use the following criteria which compare the ARTs with regular triangulations.

- Irregular vertices
  By n_{irr} we denote the number of interior vertices which do not have 14 neighbours. For an ART, n_{irr} should be small compared to n_i, the total number of interior vertices.
- Dihedral angles
  To emphasize the distribution of the dihedral angles, we give the minimum (α_{min}) and maximum (α_{max}) angle, as well as the percentage of angles lower than $6°$, $12°$ and bigger than $168°, 174°$.

- Condition of FEM matrices
  We denote by C the condition number of the stiffness matrix for the Galerkin approximation of the Poisson problem using piecewise linear basis functions. This condition number is compared with the corresponding condition numbers C_o and C_i for regular triangulations of a subset of a bounding domain. More precisely, we choose for C_o the tetrahedra which have an intersection with the domain (cf. Figure 14) and for C_i those which lie fully inside the domain (cf. Figure 15). The edge length is chosen as the average edge length of the triangulation.

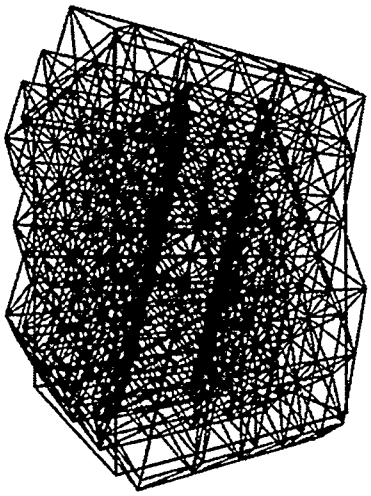


Figure 14: Outbounding Net

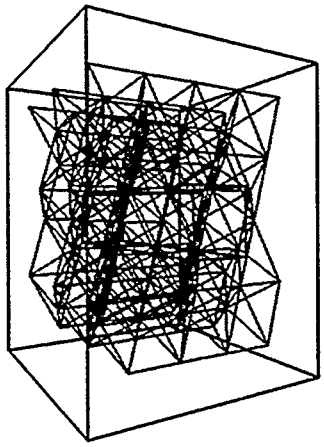
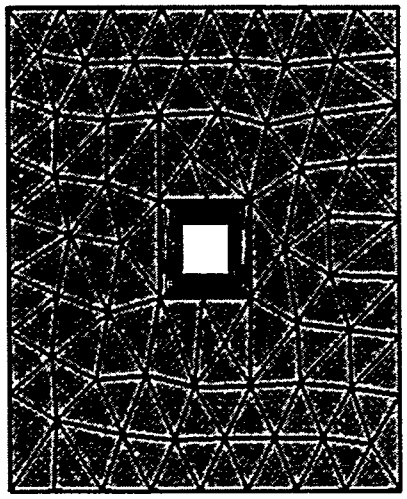
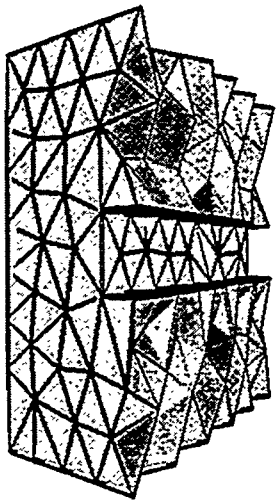


Figure 15: Inbounding Net

Figure 16 shows the triangulation of a cube with a hole. Since for this example the percentage of boundary vertices is rather high, there is not much flexibility to position the interior vertices. Hence, there are many irregular vertices. Nevertheless, the minimum and maximum angle is very good ($\alpha_{min} = 7.6°$, $\alpha_{max} = 168.2°$). Also the condition number for the FE-matrix is comparable to the one for regular meshes.

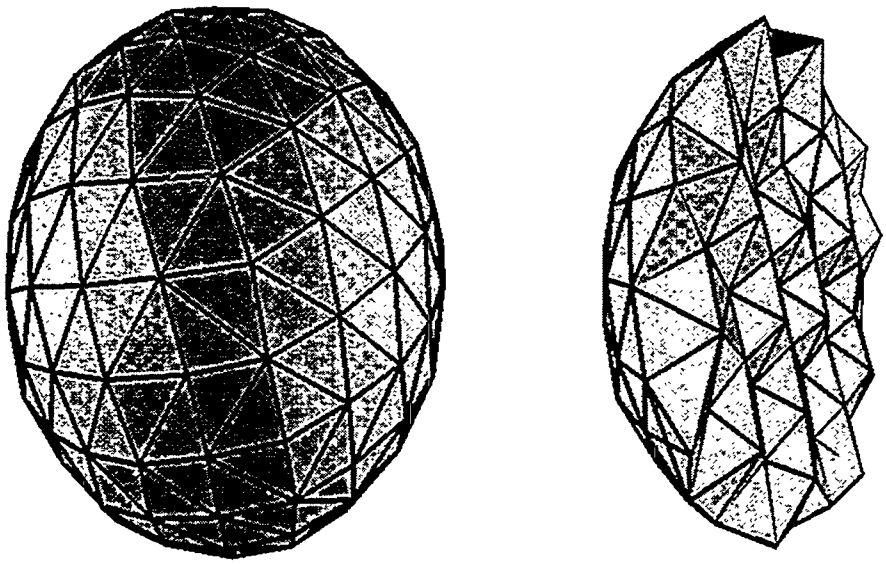




Vertices				Angle		% Dihedral angles				Condition		
Total	Inner	Boundary	Irreg.	α_{min}	α_{max}	$< 6°$	$< 12°$	$> 168°$	$> 174°$	C	C_o	C_i
860	353	507	183	7.6°	168.2°	0.00	0.12	0.02	0.00	12.69	23.49	3.66

Figure 16: Cube with a Hole

Figure 17 shows the triangulation of a ball. Surprisingly the triangulation consists entirely of regular vertices.



Vertices				Angle		% Dihedral angles				Condition		
Total	Inner	Boundary	Irreg.	α_{min}	α_{max}	$< 6°$	$< 12°$	$> 168°$	$> 174°$	C	C_o	C_i
369	175	194	0	21.4°	148.1°	0.00	0.00	0.00	0.00	15.10	16.43	8.08

Figure 17: Triangulation of a ball with cross section

In Figure 19 we show a constraint triangulation. Since for this case most of the interior vertices have to be positioned on the inner plane, the number of irregular vertices is rather high. Nevertheless, the minimum ($\alpha_{min} = 6.9°$) and maximum ($\alpha_{max} = 169.3°$) dihedral angle is acceptable and Figure 18 shows the distribution of the inner vertices and angles for this triangulation.

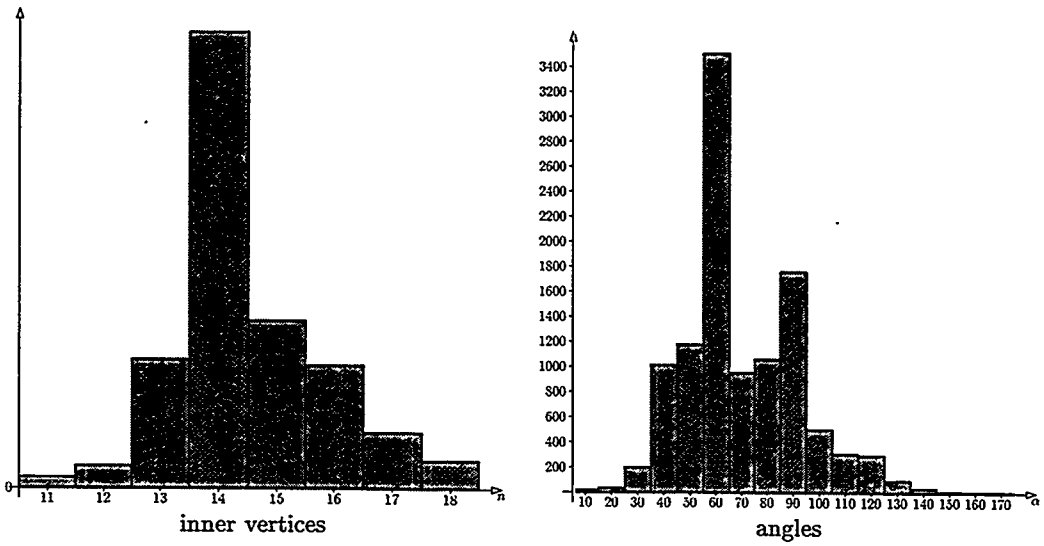
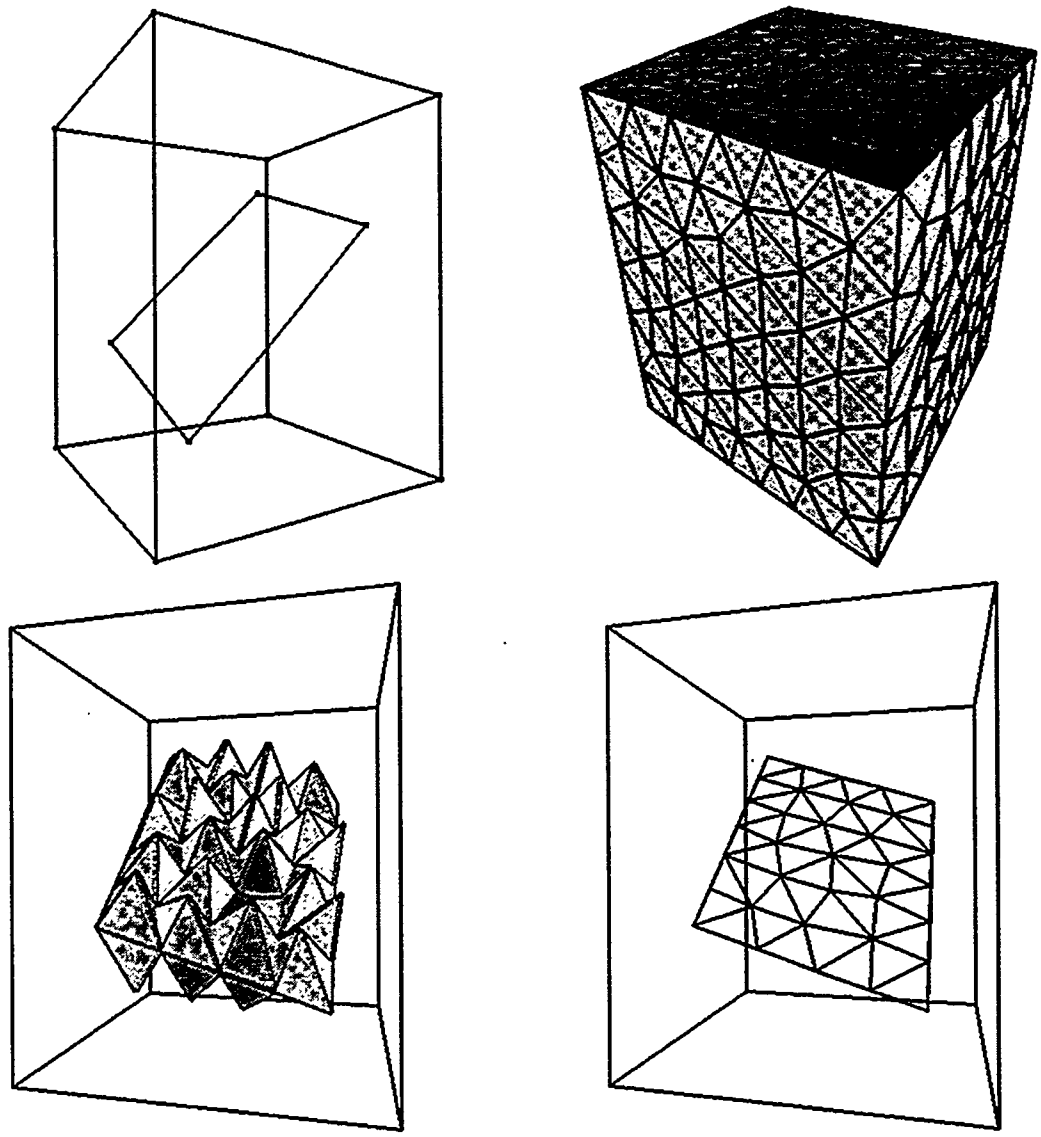


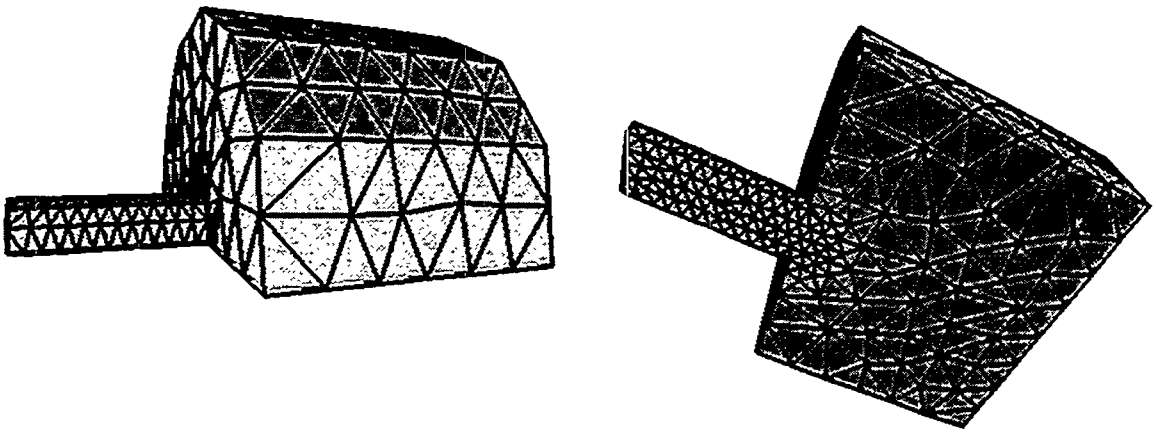
Figure 18: Distribution of inner vertices and angles.



Vertices				Angle		% Dihedral angles				Condition		
Total	Inner	Boundary	Irreg.	α_{min}	α_{max}	$< 6°$	$< 12°$	$> 168°$	$> 174°$	C	C_o	C_i
457	178	279	96	6.9°	169.3°	0.00	0.13	0.04	0.00	18.21	14.13	6.61

Figure 19: Cube with an inner plane

The last example (Figure 20) shows a graded triangulation.



Vertices				Angle		% Dihedral angles				Condition		
Total	Inner	Boundary	Irreg.	α_{min}	α_{max}	$< 6°$	$< 12°$	$> 168°$	$> 174°$	C	C_o	C_i
468	146	322	94	5.9°	174.3°	0.05	0.12	0.09	0.02	29.89	16.51	8.94

Figure 20: Graded Triangulation

7 Conclusion

We have presented a method for constructing almost regular triangulations (ARTs). The key idea is to adjust the vertices of Delaunay triangulations using a functional which penalizes irregular vertices. Our actual implementation of the triangulation algorithm is slightly more flexible than described in this paper. For example, one may choose features like interior faces and one has a number of options to control the optimization process. For such details we refer to [12] and the online description of our program which is available via anonymous ftp from *ftp.mathematik.uni-stuttgart.de* or via WWW from *http://www.mathematik.uni-stuttgart.de/mathe/preprints.html*. A corresponding algorithm for 2-dimensional problems is described in [10].

An advantage of our method compared to many existing algorithms is, that the smoothing and optimization of the point configuration is done before any connections are assigned. This provides greater flexibility, in particular when propagating irregularities of the boundary towards the interior of the domain. We intend to generalize our algorithm to NURBS solids and trimmed surfaces.

Acknowledgements

This work was performed at the University of Stuttgart. This work was supported by the SFB 404, University of Stuttgart.

References

[1] Scott A. Canann, Micheal B. Stephenson, and Ted Blacker. Optismoothing: An optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design*, 13:185–190, 1993.

[2] Susanne C. Brenner and Scott L. Ridgway. *The Mathematical Theory of Finite Element Methods.* Springer-Verlag, 1994.

[3] H. Borouchaki, F. Hecht, E. Saltel and P.L. George. Reasonably Efficient Delaunay Based Mesh Generator in 3 Dimensions. In *Proceedings, 4th International Meshing Roundtable*, pages 3–14. Sandia National Laboratories, 1995.

[4] G. Farin. *Geometric Modelling: Algorithms and new Trends.* SIAM, 1987.

[5] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press Inc. (London) Ltd., 1 edition, 1988.

[6] Lori A. Freitag, Mark Jones, and Paul Plassmann. An Efficient Parallel Algorithm for Mesh Smoothing. In *Proceedings, 4th International Meshing Roundtable*, pages 47–58. Sandia National Laboratories, 1995.

[7] Lori A. Freitag and Carl Ollivier-Gooch. A Comparison of Tetrahedral Mesh Improvement Techniques. In *Proceedings, 5th International Meshing Roundtable*, pages 87–100. Sandia National Laboratories, 1996.

[8] Lori A. Freitag. Tetrahedral Mesh Improvement Using Swapping and Smoothing. To appear in International Journal for Numerical Methods in Engineering.

[9] A. Fuchs. Optimierung von Delaunay-Triangulierungen. Master's thesis, Universität Stuttgart, 1996.

[10] A. Fuchs. Almost regular Delaunay-Triangulations. *Int. J. for Numerical Methods in Engineering*, 40:4595–4610, 1997.

[11] A. Fuchs. Parallel Delaunay triangulation. in preparation, University of Stuttgart, 1998.

[12] NAGD Research Group. Software. Report 96-5, University of Stuttgart, 1996.

[13] J. Hörner. Condition Numbers of FE-Matrices. private communication.

[14] K. Höllig and H. Mögerle. G-splines. *Computer Aided Geometric Design*, 7:197–207, 1990.

[15] J. Hoschek and D. Lasser. *Grundlagen der Geometrischen Datenverarbeitung.* B.G. Teubner, Stuttgart, 1989.

[16] N. A. Golias and T. D. Tsiboukis. An Approach to Refining Three-Dimensional Tetrahedral Meshes Based on Delaunay Transformations. *Int. J. for Numerical Methods in Engineering*, 37:793–812, 1994.

[17] D. T. Lee and B. J. Schachter. Two Algorithms for Constructing a Delaunay Triangulation. *Int. J. Comput. Inf. Sci. 9*, 9:219–242, 1980.

[18] D. Liu and J. Hoschek. GC^1 Continuity Conditions between Adjacent Rectangular and Triangular Bézier Surface Patches. *CAD*, 21(4):194–200, May 1989.

[19] Franco P. Preparata and Michael Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, 1985.

Hybrid Volume Meshing

The "Hex-Tet" Hex-Dominant Meshing Algorithm as Implemented in CUBIT

Ray J. Meyers (rjmeyer@sandia.gov)
Timothy J. Tautges (tjtautge@sandia.gov)
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM

Dr. Philip M. Tuchinsky (ptuchins@ford.com)
Ford Research Laboratory
Mail Drop 2122 SRL
P. O. Box 2053
Dearborn, MI 48121 USA

Abstract. This is a report of the current status of the Hex-Tet algorithm as implemented in the CUBIT toolset. The Hex-Tet algorithm begins by generating a partial hex mesh using an advancing front "Plastering" algorithm. The boundary of any remaining void is optionally "cleaned-up" to improve its shape and/or other properties. The quad boundary is then converted to a triangular boundary by one of three available methods. Finally, the triangle-bounded void is filled with tetrahedra using the tet-generation capabilities within CUBIT. The result is a mixed-element mesh containing hexahedra, tetrahedra, and optionally pyramids. A set of test problems is described which is used to generate data concerning the robustness and speed of the algorithm, as well as properties of the resulting meshes.

Keywords: Hexahedral Meshing, mixed-element mesh, hex-dominant meshing, unstructured mesh generation, plastering.

Introduction

The desire within the finite element analysis community for an automated all-hexahedral mesh generation algorithm is well documented. At Sandia National Laboratories, one effort to provide this capability is the plastering algorithm (Stephenson:1990, Canann:1992, Blacker:1993, Hipp:1994, Hipp:1995), a 3D extension of the very successful Paving algorithm for all-quadrilateral surface meshes. To date, the all-hexahedral algorithm has proved intractable. In recent years, attention was shifted to the "hex-tet" algorithm, which focuses on creating a hex-dominant mesh by filling the volume with as many hexes as possible, and then filling the remainder with tetrahedra (Dewhirst:1995). Tuchinsky and Clark (Tuchinsky:1997) give an excellent paper on past advancements on this algorithm at Sandia. Our report documents very recent developments with the Hex-Tet algorithm, concentrating on the commercialization of the algorithm for use within CUBIT, and on documenting the current robustness of the algorithm and the properties of resulting hex-dominant meshes. The following portions of the paper will document recent additions to the algorithm to enhance robustness, and describe a test suite of non-trivial problems used to evaluate the algorithm and the results obtained.

Adapting the Hex-Tet algorithm for CUBIT

Much recent work done on the Sandia Hex-Tet algorithm was conducted at Ford Motor Company (Tuchinsky:1997) using Ford software, including the Ford tet-generator. This latest plastering algorithm was

adapted for inclusion in the CUBIT system. A quality tet-generation algorithm has been recently incorporated in CUBIT, and these two were combined to provide an automated Hex-Tet algorithm.

To provide a transition from the quad-bounded void left by the hex-generator and the tri-bounded void needed by the tet-generator, three options were provided. The first option, labeled "Two" in the test data below, simply splits the quads into two triangles along the shortest diagonal. The second, labeled "Four" below, creates a center point for each quad and divides the quad into four triangles. The last, called "Pyramid", inserts pyramid elements between the hexes and tets, resulting in a fully conformal mesh containing three element types.

The pyramid generation algorithm used is essentially that described by Owen, Canann, and Saigal (Owen:1997), with a few additions to handle the very complex void geometries created by the HexToVoid algorithm. The most significant of these additions involved the potential for warpage on the quad faces of the void boundary. When a warped quad is divided into two triangles along the diagonal and the resultant triangle boundary is tetrized, it is possible that both of the triangles of the original face may be owned by a single tetrahedron. In this case, we must find the edge of the tetrahedron which is not part of the original two triangles, and split all tets in the model which share this edge. We can then proceed with pyramid construction as normal.

For most analyses, an ideal mixed-element mesh would contain several layers of all-hex mesh near the boundary, with the tets (and possible pyramids) isolated to relatively small areas in the interior. To that end, special attention was given to the hex generation algorithm to attempt to maximize the percent volume occupied by hexes in the model, and to minimize the number of faces on the original meshing boundary which remain after hex generation.

Void Cleanup

Some additional work was required to allow the Hex and Tet portions of the algorithm to work well together. In complex geometries, the void left by the Hex generator is often very complex, with a very high surface to volume ratio, especially for relatively thin-section models (Figure 1).

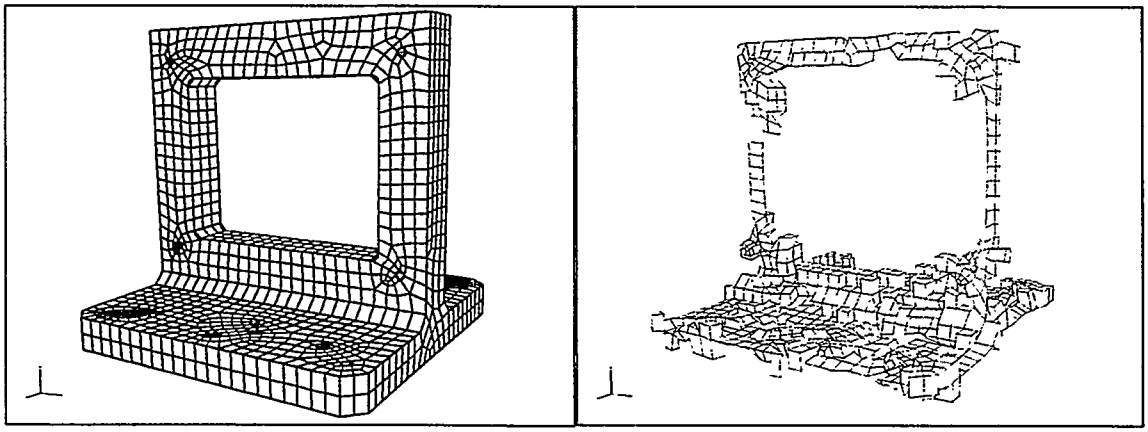


Figure 1: Winblock Model and Void Remaining after Hex Generation

The term "pancake void" was coined to describe the shape of void often seen when two advancing hex fronts would collide. Because of the fixed point density on the surface in addition to the complex geometry, these voids are very difficult for an automated tet-generator to fill with quality elements. To improve both robustness and tet quality, a void cleanup algorithm was devised to condition the void before tet generation.

The void cleanup algorithm attempts to improve the void by removing selected hex elements in such a way that the resulting void has improved properties for mesh generation. A distance tolerance was used to guarantee that any two non-adjacent quads on the void boundary would be separated by at least d, where d is some user-settable distance. This provides sufficient space for the tet generator to insert better elements. Additional Hex removal may also be performed given any of the following:

1. More than three faces of a single hex are on the void boundary.

2. An edge on the void boundary is connected to more than two faces on the void boundary.
3. The dihedral angle between any two faces on the void boundary is less than some minimum threshold value (typically about 45 degrees) or greater than some maximum value (typically about 315 degrees).
4. One of the interior angles of a boundary face is less than some minimum value (typically 45 degrees) or greater than some maximum value (typically 135 degrees).

Other criteria could also be used depending on the properties of the tet generation algorithm and on the desired results.

Testing the Hex-Tet Algorithm

Although some testing had been conducted on both the hex generator and the tet generator, a quantitive measure of the robustness of the combined algorithm was needed. To this end, a test suite of non-trivial problems was assembled for testing and evaluation. In all, seven models were chosen (see figures 2,3,8-11):

- Hook: This is a model of a real part, named for its general hook shaped appearance. The model contains several small and irregular facets.
- Ugly: Named either for its appearance or its behavior during meshing, this is actually a subset of the Hook model.
- Winblock: A blocky but difficult part with many through holes of widely varying diameter.
- TDDH: This model is best described as a soccer-ball shape with flat panels. It contains many angles between 120 and 150 degrees, which are very difficult for hexing algorithms to handle.
- Knee: A model of an artificial knee joint.
- Throw: A complex machined part, apparently part of a crankshaft assembly.
- Enterprise: A model of the well-known Starship, circa 1964.

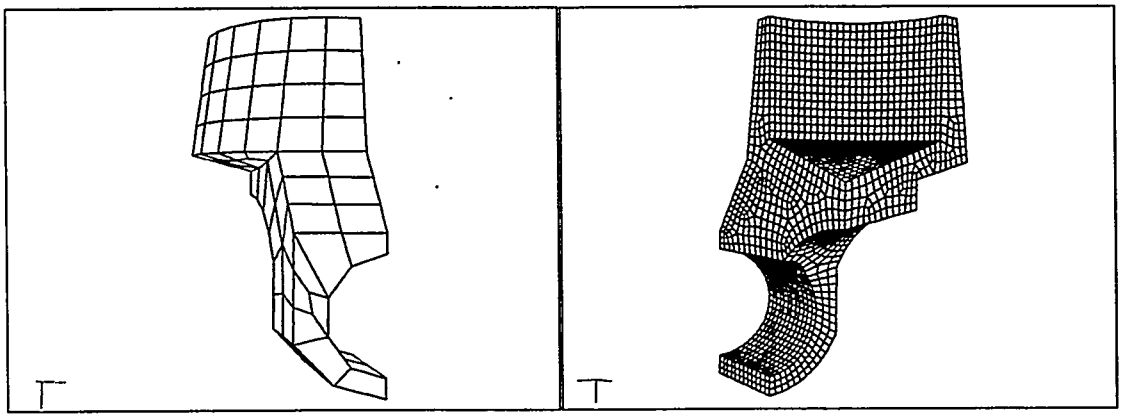


Figure 2: Hook Model, Low and High Resolutions

The models were chosen as examples of complex volumes for which current all-hex algorithms do not work well. Each model was tested over as wide a range of interval sizes as possible (from a maximum of 11 sizes for the Ugly model, to only two for the Enterprise). In general, the coarse mesh limit was governed by the coarsest mesh the current cubit surface meshing algorithms could produce for a given model with little or no human intervention. In general, the coarse meshes were too coarse to capture all of the detail in the model. The fine mesh limit was generally determined by the available dynamic memory limits of the computer used for these tests. Finer tests could be run on larger machines, but the range of intervals used for the models seems sufficient to draw meaningful conclusions on the capabilities of the Hex-Tet algorithm.

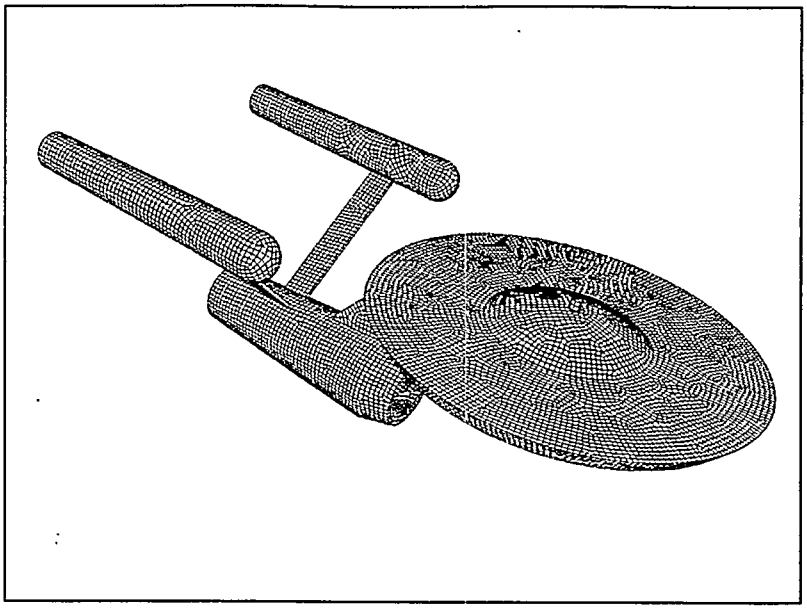


Figure 3: The Enterprise Model

Tests were performed for a total of 43 model/interval combinations with three transition methods for each. The table below summarizes the results for the test suite.

	Hook	Ugly	Winblock	Enterprise	Knee	TDDH	Throw	Totals
Surface Mesh Success	9/9	11/11	2/5	2/2	5/6	6/6	4/4	39/43
Hexing Success	9/9	11/11	2/2	2/2	2/5	6/6	3/4	35/39
Tet Success, trans: Two	9/9	11/11	2/2	2/2	2/2	6/6	2/3	34/35
Tet Success, trans: Four	8/9	10/11	0/2	0/2	0/2	5/6	2/3	25/36
Tet Success, trans: Pyramid	9/9	11/11	2/2	2/2	2/2	6/6	2/3	34/35

Figure 4: Results of Test Runs

Robustness

The success rate of the HexToVoid algorithm on a wide range of difficult models was surprising. For the Hex generation portion of the algorithm, the success rate was 35/39, or 90%. The failures were breakout problems, which showed up three times on the Knee problem and once on the Throw model.

The success rate for tetrizing the remaining voids was dependent on the transition method used. The success rate using the two-triangle transition was a very respectable 34/35 or 97%. This is excellent considering the very difficult voids the tet generator is asked to fill. Using the four-triangle transition, the rate was 25/36, or 70%. The large difference in success is probably due to the fact two good triangles can be formed from a skewed quad, whereas the four triangle transition on the same quad results in very thin triangles. The pyramid transition actually uses the two-triangle transition and tet generation, and then inserts the pyramid transition layer as a post-processing step. The success rate for the pyramid transition was identical to the two-transition at 97%.

For the models shown, the overall success rate for the Hex-Tet algorithm using the two-triangle or pyramid transition is 87%, while using the four-triangle transition results in 64% success for the tests performed. Interestingly, the HexToVoid algorithm performed remarkably well for large interval sizes (very coarse meshes).

Algorithm Effectiveness and Predictability

In addition to robustness, the testing included metrics which might indicate whether the algorithm is behaving sensibly. For example, we would expect that as resolution is increased, the percentage of the volume filled with hexes should increase, and the percentage of original boundary quads remaining after hex generation should decrease. Three of the models (Hook, Ugly, and TDDH) had enough data points for meaningful graphs. The graph for the Ugly model is shown below. The summary indication is that for very coarse models, the graphs jump around somewhat randomly. When the resolution becomes sufficient to accurately capture the details of the model, the behaviors then become monotonic and predictable. From this viewpoint, it appears that the hex generation portion of the algorithm behaves correctly and predictably as mesh size is scaled.

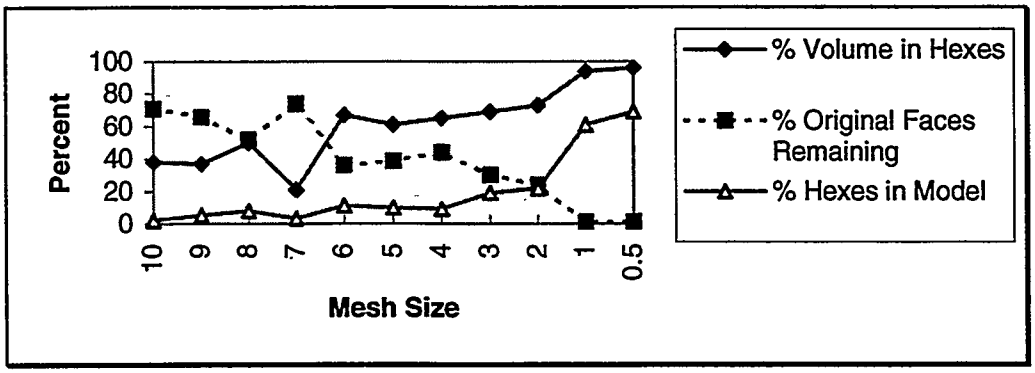


Figure 5: Algorithm Effectiveness For The Ugly Model

Algorithm Speed

Currently, the Hex-Tet algorithm is still very slow. The Hex generator, which takes most of the time, has been constructed for robustness, but as yet never modified for efficiency. The table below summarizes the average speed of the Hex and Tet generation portions of the algorithm, as well as overall speed.

Algorithm	Elements per Second
Hex Generation	9.84
Tet Generation	221.84
Overall	61.40

Figure 6: Average Timing Statistics for 43 Test Models

Element Quality

Idea of what constitutes a model of acceptable quality varies greatly from discipline to discipline and from individual to individual. For the purposes of this study, we chose to use a normalized average element jacobian as a reasonable indication of the quality of elements created. Of course, this does not mean the models would be acceptable for any given analysis problem. The jacobian was calculated as an average of the jacobian at each node, and was then normalized by dividing by the cube of the average edge length to give a normalized range of

$$-1.0 <= \text{jacobian} <= 1.0$$

As a computational minimum, all elements should have a jacobian greater than zero. Both the hex and tet generators currently prevent any element from being created which has a non-positive jacobian. However, global smoothing,

which in general will improve the quality of elements, can sometimes also create negative jacobians. The table below shows the ranges of jacobians found for the test suite. The absolute minimum and maximum reflect, respectively, the lowest minimum and highest maximum found over all 43 models.

Parameter	Hex Elements	Tet Elements
Average Minimum Jacobian	0.306	0.062
Average Maximum Jacobian	0.983	0.655
Average Jacobian	**0.843**	**0.330**
Absolute Minimum Jacobian	0.007	0.001
Absolute Maximum Jacobian	1.000	0.710

Figure 7: Quality Metrics for The 43 Model Test Suite

The quality of hex elements generated overall was excellent, although there were isolated hexes in some models which would not be acceptable. The quality of the tet elements generated was not as good. This is partly attributable to the fact that the tet improvement algorithms planned for CUBIT (Joe:1995) have not yet been implemented.

Conclusions

The Hex-Tet algorithm implemented in CUBIT appears to be a useful addition to the CUBIT toolset. The algorithm handled a wide range of non-trivial models with robustness, producing meshes of acceptable quality in most cases.

Although many of the models used to test the algorithm are large and complex, we do not recommend using the Hex-Tet algorithm as a silver bullet to mesh an entire assembly with the push of a button. Rather, it is a useful tool to be used in the overall context of the CUBIT package, where models can be easily decomposed as necessary, and the best algorithm for each volume or sub-volume can be invoked as necessary. We do see it as highly useful, however, for those geometries which cannot be decomposed or meshed using other current algorithms, provided the intented solver can handle the resultant mixed element mesh.

Further work still needs to be done to improve the speed and efficiency of the hex generation portion of Hex-Tet to make it a faster and easier tool to use.

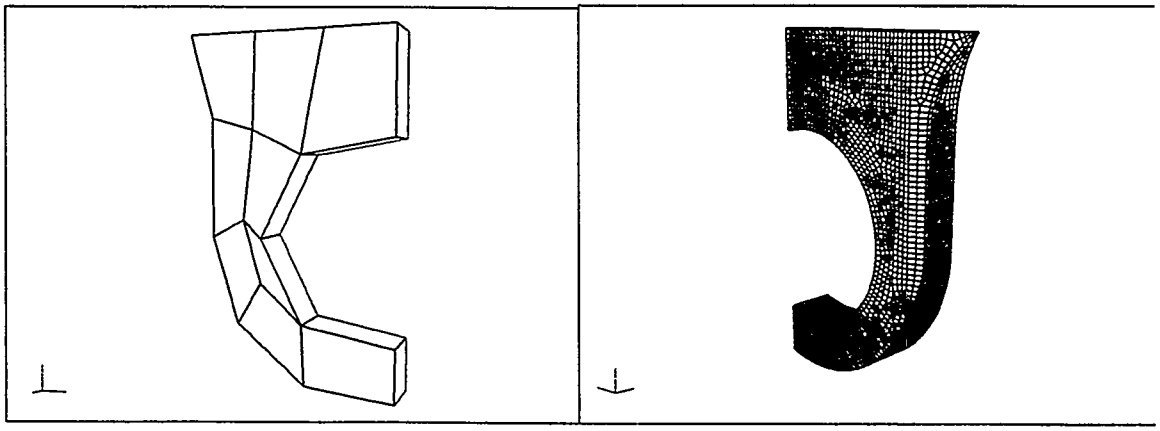


Figure 8: Ugly Model, Low and High Resolutions

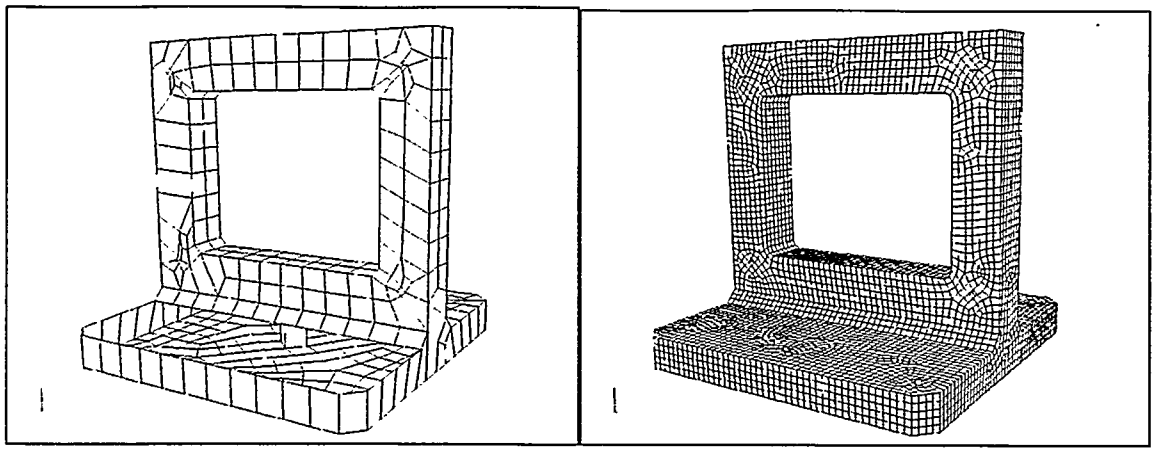


Figure 9: Winblock Model, Low and High Resolutions

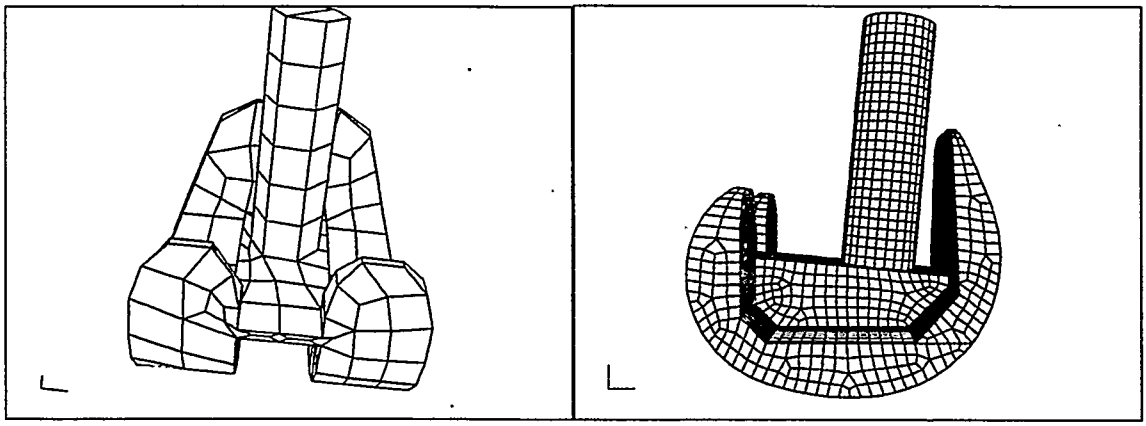


Figure 10: Knee Model, Low and High Resolutions

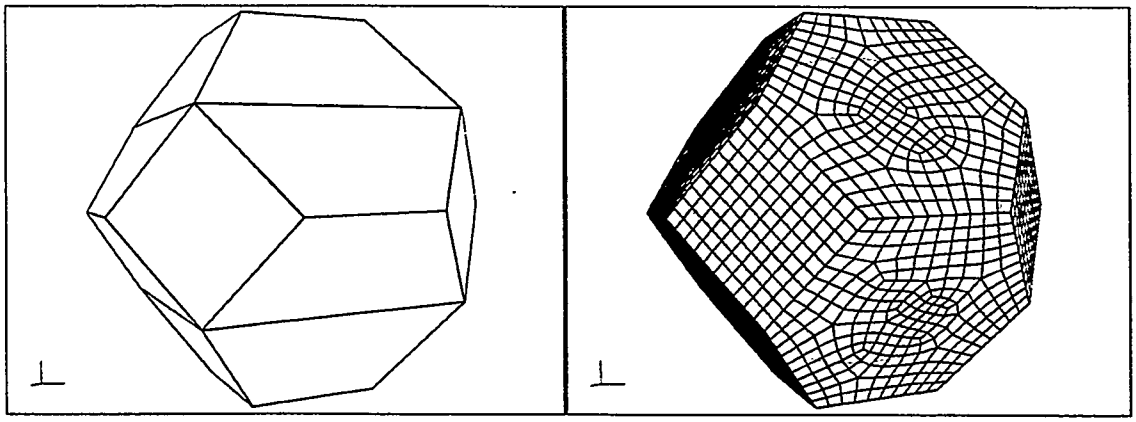


Figure 11: TDDH Model, Low and High Resolutions

Bibliography

Blacker, T. D. and Meyers, R. J., 1993, "Seams and Wedges in Plastering: A 3-D Hexahedral Mesh Generation Algorithm." , Engineering With Computers, Vol 9, pp. 83-93.

Canann, S. A., "Plastering: A New Approach to Automated, 3-D Hexahedral mesh Generation", American Institute of Aeronautics and Astronics, 1992.

Dewhirst, D. L., Vangavolu, S., and Wattrick, H., "The Combination of Hexahedral and Tetrahedral Meshing Algorithms", Proceedings, 4th International Meshing Roundtable, Sandia National Laboratories, October, 1995, pp. 291-304.

Hipp, J. R. and Lober, R., "Plastering: All-Hexahedral Mesh Generation Through Connectivity Resolution", Proceedings 3rd International Meshing Roundtable, Oct. 24-25, 1994, Albuquerque, New Mexico.

Hipp, J. R. and Clark, B., "Plastering Algorithm and New Knife Embedding and Face Insertion Implementation", Proceedings 4th International Meshing Roundtable, Oct. 16-17, 1995, Albuquerque, New Mexico.

Joe, B., "Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations", Siam Journal of Scientific Computing, Vol. 16, pp. 1292-1307, 1995.

Owen, S. J., Canann, S. A., and Sunil, S., "Pyramid Elements For Maintaining Tetrahedra to Hexahedra Conformability", Trends in Unstructured Mesh Generation, AMD Vol 220, ASME 1997, pp. 123-129.

Stephenson, M. B., Canann, S. B., and Blacker, T. D., "Plastering: a New Approach to Automated, 3-D Hexahedral Mesh Generation - Progress Report 1", Report SAND89-2192, Sandia National Laboratories, Albuquerque, New Mexico, 1990.

Tuchinsky, P. M., and Clark, B. W., 1997, "The 'Hex-Tet' Hex-Dominant Automesher: An Interim Progress Report" , Proceedings 6th International Meshing Roundtable, Sandia National Laboratories, October 1997, pp. 183-193.

A Hybrid Mesh Generation Method for Two and Three Dimensional Simulation of Semiconductor Processes and Devices.

Dan Wake, Klas Lilja and Victor Moroz
Avant! Corporation, TCAD Business Unit, 46871 Bayside Parkway, Fremont CA 94538
Phone: 1-510-413-8202 Fax:1-510-413-7743
E-mail : dan_wake@tcad.avanticorp.com

Keywords: hybrid mesh generation, octree, level set methods, semiconductors

ABSTRACT

We present a hybrid mesh generation algorithm, which has been developed for semiconductor process and device simulation. The method uses Cartesian elements (bricks or hexahedra in 3D and rectangles in 2D) whenever possible and non-Cartesian elements (prisms and tetrahedra in 3D and triangles in 2D) are only introduced when necessary to resolve material boundaries or other features of interest. Our approach utilizes a quadtree-octree data structure with a level set representation of material boundaries. This combination allows for an efficient and robust tracking of complex moving boundaries. Efficiency is obtained by localizing non-Cartesian grid generation and by obtaining fast connectivity information from the quadtree-octree data structure. Robustness is provided by the level set formulation, which easily handles topologically difficult interfaces, such as regions which begin simply connected but lose this connectivity after boundaries merge. Following a discussion of the method, results are presented.

INTRODUCTION

Semiconductor process and device simulations impose a number of unique requirements on the grid structure. Firstly, variations in mesh density are required to accurately capture the abrupt variations in the solutions that occur at junctions and at boundaries. Secondly, different features often have disparate length scales. In device simulation, conducting channels often are much smaller than other features of the device. In process simulation, surface layers, which need to be resolved, are often orders of magnitude smaller than the bulk substrates on which they are deposited. Utilizing a fine mesh throughout the device would be prohibitively expensive. This argues for a local refinement capability. The resolution is typically required normal but not tangential to the surface, so the refinement must be anisotropic as well. Thirdly, though many devices are essentially planar, important features are not Cartesian and we want to include them without introducing an unstructured grid throughout the solution domain. As the simulation of complex structures can be quite simulation of electrical characteristics subsequent to a simulation of the processing which generated the device. These two simulations have different mesh requirements, which calls for an unrefinement capability in addition to a refinement capability. expensive, keeping high-accuracy Cartesian elements in as much of the domain as possible is crucial. Lastly, it is desirable to perform a

These requirements for semiconductor simulation favor the quadtree-octree mesh generation method [1], in which the mesh is created by successive subdivision of rectangles/bricks. These types of meshes allow for a very easy and natural anisotropic refinement and unrefinement, and any type of search in the mesh is fast. For example, the complete refinement of an N element quadtree-octree mesh takes (on average) O(NlogN) operations [10]. The quadtree-octree mesh generation method has been applied successfully to fixed grid problems in both device simulation and diffusion simulation [2,3]. In this work we show how it can be used efficiently for

problems involving moving boundaries and adaptive meshing.

When applying the octree method to non-planar geometries the basic algorithm must be extended. In the modified quadtree-octree method [1] this is done by the introduction of new nodes at the intersection points of the geometrical boundary with the quadtree-octree elements. The intersected elements are then tessellated into triangles/tetrahedra, respecting the geometrical boundary. Variants of this method have been applied to the simulation of semiconductors [2][3]. The method appears to have some inherent shortcomings, though, as was pointed out in [4].

Moving boundary problems (oxidation, silicidation, deposition and etching are examples from the semiconductor industry) pose additional problems for the mesh generation. For 2D problems moving boundaries have traditionally been handled by the string algorithm, and 3D extensions to this algorithm have been presented [5]. However, for problems which need a boundary conforming mesh, the string algorithm requires the mesh to be unstructured. The use of an unstructured mesh severely complicates unrefinement and anisotropic refinement. In addition the problem of "collapsing elements" (de-looping) in the 2D string algorithm is considerably harder in 3D.

An attractive alternative to the string method for moving boundary problems is the level set method [6]. In this method the boundary is represented by the N-1 dimensional subspace of an N dimensional space in which the level set function is zero. The grid is not required to be boundary conforming in principle. The method offers elegant and stable boundary tracking, which naturally handles breaking and merging regions - problems that are hard to handle for any string algorithm. As will be discussed below the level set representation of the geometry also opens new possibilities for the use of a quadtree-octree mesh in complex geometries and for moving boundaries.

QUADTREE-OCTREE ALGORITHM

The basic quadtree-octree algorithm successively subdivides rectangles or bricks into smaller elements. The version we have chosen subdivides an element into two identical elements along a plane given by the refinement criteria. These subdivisions introduce unconnected ("green") nodes at the edges of adjacent elements, which are resolved by subdividing these elements into triangles or tetrahedra, pyramids, and prisms. The basic algorithm works if any edge has a maximum of one unconnected node (1-irregular mesh). Fortunately, by balancing the mesh tree during construction [7], it can be ensured that only 1-irregular mesh trees are constructed, and the time consuming task of converting a mesh to be 1-irregular can be avoided.

The advantage of a mesh tree, over an unstructured grid, for refinement and unrefinement purposes is apparent. The tree structure allows us to easily remove or add leaves for appropriate unrefinement or refinement.

BOUNDARY TRANSFORMATION

One of the problems with the modified quadtree-octree method used to handle non-planar geometries in [1][2][3] is that it introduces new nodes at irregular positions in the grid. The tessellation patterns of the elements intersected by a boundary can become very complicated and the tessellation needs to be redone every time further refinement, unrefinement, or boundary movement is performed. Problems also occur when the intersection points are close to a node in the tree, which can lead to reduced mesh quality. Instead of adding new nodes, we have chosen to transform the mesh by moving the nodes to new positions.

The algorithm we use to transform the grid uses a level set representation of the boundary. A boundary (T) is represented by the zero level set of a function $\phi(x)$. $\phi(x)$ maps the coordinate x to a real number, d, representing the signed distance to the surface in question. $\phi(x) = d$ if the point is inside of T, and $\phi(x) = -d$ if the point is outside of T. This boundary description also allows for refinement of the geometry based on the level set function, and can allow for boundary movement through the grid (below).

Level set methods allow us to solve our equations on an Eulerian grid yet retain accurate boundary tracking. We have chosen to perform a local geometrical transformation of the mesh to make it conform to the boundary. Figure 1 gives a simple illustration of the procedure – the positions of the octree nodes are transformed to make the mesh conform to the boundary. This geometrical transformation allows us to handle topological changes, maintains proper element shapes and

mesh density, and is local, i.e., only the mesh in the vicinity of the surface is affected.

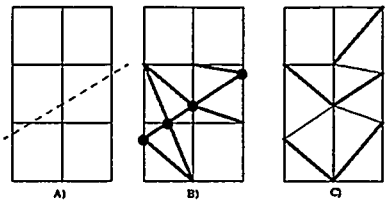


Figure 1. When boundary crosses the regular quadtree mesh (A), nodes can be added (B), or the mesh can be transformed (C). The thicker edges are the newly introduced ones.

The basic steps required to create a boundary conforming mesh via a local geometrical transformation are straightforward:

a) Identify edges intersected by the boundary

b) Move the nodes closest to the boundary to a location on the boundary

c) Tessellate all elements affected by the above operations, while respecting the region boundaries

Note that in c) the tessellation patterns required are the same ones as for the regular quadtree-octree, except that any diagonal edges along the geometrical boundary must be maintained.

The most difficult part of the algorithm is a consistent implementation of steps b) and c) together. The movement of nodes to the boundary must not result in highly distorted elements or interfere with other boundaries. The tessellation in c) must similarly produce acceptable non-cartesian elements. Our rule-based algorithm has proven successful in satisfying these requirements for all cases to date.

MOVING BOUNDARY

Using the level set formalism, we advance a material boundary through a structure by solving the level set equation:

$$\frac{\partial \varphi}{\partial t} + F|\nabla \varphi| = 0 \qquad (1)$$

or

$$\frac{\partial \varphi}{\partial t} + \bar{v} \cdot \nabla \varphi = 0 \qquad (2)$$

where F is the normal surface speed, and $\bar{v}$ the vector speed of the surface movement. Additionally we introduce an artificial diffusivity term

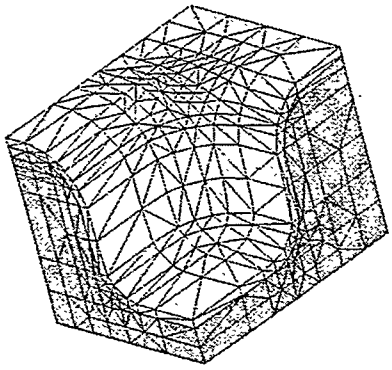
$$\varepsilon \Delta L \nabla^2 \varphi$$

where epsilon is a diffusion parameter and ΔL is the characteristic element size. This partial differential equation solution utilizes a finite volume discretization in space and an implicit time integration. The non-linear system is then solved with Newton iteration. Several other non-trivial issues are involved in moving the boundary using the level set equation (such as the handling of triple lines and points), but they are beyond the scope of this paper.

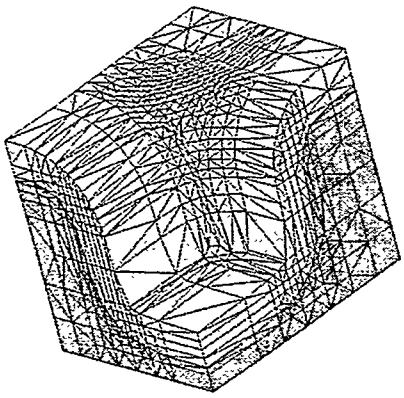
REFINEMENT AND UNREFINEMENT

The quadtree-octree data structure allows for efficient local grid refinement and unrefinement [1,2,3,10]. While both are important for fast and accurate simulations, the need for an unrefinement capability is especially important for semiconductor simulation. It is desirable to model the fabrication process and subsequently characterize the same device electrically. Each of these simulations requires high mesh density in different parts of the device. Also, device simulation can be very computationally intensive and it is therefore important to remove uneeded mesh. The quadtree-octree data structure is key to the refinement and unrefinement capability.

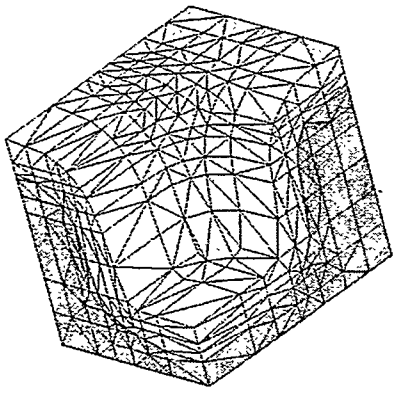
Figure 2 shows an example of the unrefinement capability. First an initial grid is generated consisting of a thin oxide layer on top of a silicon trench. An oxidation simulation is conducted, resulting in a thick oxide layer on top of the substrate. Note the large number of elements now in the oxide region. The mesh is then unrefined to a more suitable mesh density. The initial grid uses 5,170 elements and this grows to 15,114 elements after the oxidation. The unrefined mesh has 4,719 elements, less than the original mesh.



A)



B)



C)

Figure 2. Mesh at different stages of oxidation simulation. (A) Initial mesh with thin oxide layer on top of silicon. (B) Mesh after oxidation. (C) Mesh after unrefinement.

BOUNDARY DESCRIPTION

The level set method imposes additional requirements on the boundary description that are unique to the method. While our boundaries are simply the zeroes of a level set function, we must define the level set function throughout the domain. The most straightforward method for initializing the level set function at a point is to use the signed distance from the boundary surface in question. The sign is determined by whether a point is inside or outside of the body.

Our approach to this initialization problem is to define our boundary in terms of a union of geometric primitives. For any point in space, we can calculate a signed distance from each of these primitive components describing the structure. Taking the minimum over the collection gives us the distance to the surface. Figure 3 shows a more complex trench structure which is defined using polygons (the top surface, the trench walls and the trench bottom) and cylinders (for the rounded edges). This modular approach allows flexibility in defining many different structures of interest.

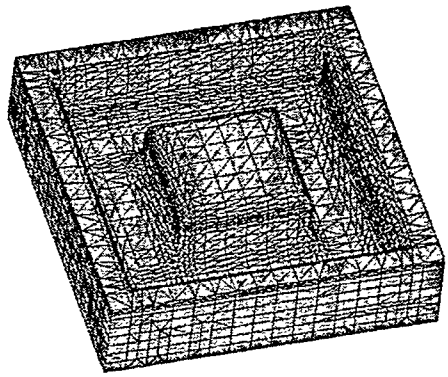


Figure 3. Silicon trench with rounded edges generated using a union of geometric primitives.

In Figure 4, we show how we may render curved objects within the quadtree-octree framework. A silicon hemisphere lies atop a silicon slab, both of which are covered with a thin oxide layer.

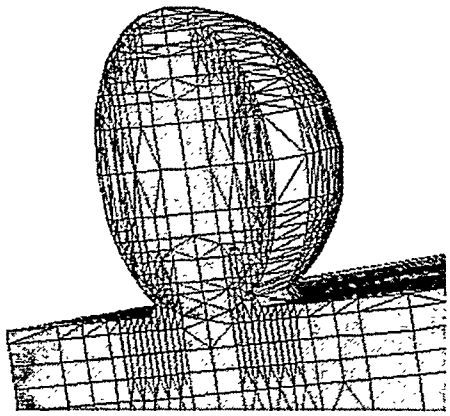


Figure 4. Half sphere of silicon with oxide layer.

Like most of the grid generation community, we are interested in the automation of grid generation and interfacing with other design tools (CAD, GUI's, etc). To this end, we utilize TmaLayout [9], a graphical editor for semiconductor layout definition. This editor is intended for the creation of 1D and 2D masks for etch simulation. We have used it to simulation etching of 2D and 3D structures.

SOFTWARE IMPLEMENTATION

The mesh generation algorithm has been incorporated into our semiconductor process and device modeling program Taurus [8]. Taurus is an object oriented C++ code that uses the finite volume method to solve the partial differential equations that govern the simulation. This method lends itself naturally to these mixed element meshes. The level set equation is solved along with the other equations in the simulation and does not significantly affect performance. Taurus also has an unstructured tetrahedral mesh generation capability provided by SCOREC (Scientific Computation Research Center) at Rensselaer Polytechnic Insitute [11], which is not discussed here.

RESULTS

We present a few typical application examples from semiconductor process simulation. The largest meshes we have generated using this algorithm are about 100,000 elements. All of the grids shown in these examples were generated quickly, usually in a few minutes on a Sun workstation. The most computationally intensive case, the three dimensional MOSFET, runs over night (and includes implanation, diffusion, deposition and etch simulation in addition to grid generation).

Trench Corners

STI (Shallow Trench Isolation) technology is widely used for semiconductor devices from memory to power devices. The presented mesh generation can be used to study the effect of different rounding radii on the structural and thermal stresses generated during processing, as well as electrical effects in device operation. Such a trench structure is shown in Figure 5 and the localized stresses calculated from the simulation are shown in Figure 6.

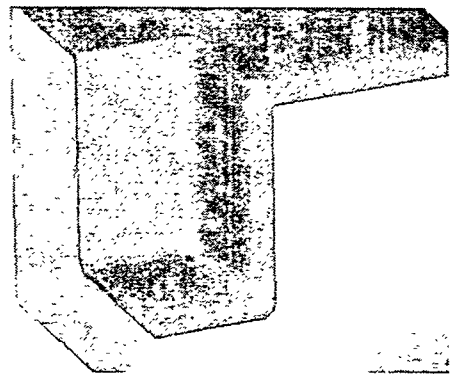


Figure 5. Trench structure. Oxide layer on top of a silicon trench with rounded corners.

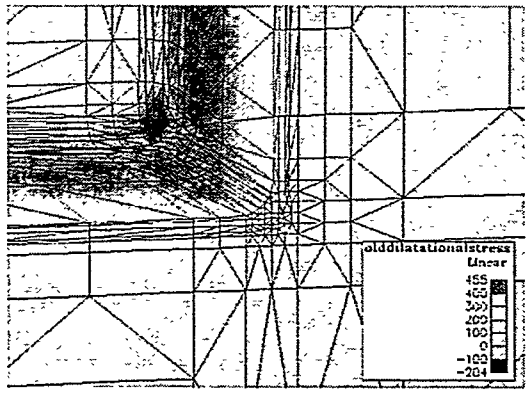


Figure 6. Mesh detail for the trench structure. The total mesh has 4310 nodes. The field shown in the figure is the dilatational component of the

mechanical stress generated by a thermal ramping process. The maximum tensile stress for the rounded corners is 455Mpa, to compare with 537Mpa for sharp (on the scale of this mesh) corners.

LOCOS Oxidation

Figures 7 and 8 show the mesh for LOCOS (Local Oxidation of Silicon) simulation. The initial thin oxide layer shown in Figure 7 grows into the silicon and expands, deforming the nitride mask on top, as is shown in Figure 8.

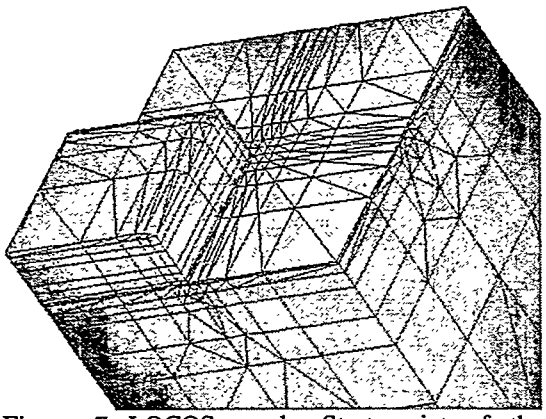


Figure 7 LOCOS mesh. Start point of the oxidation (1640 nodes).

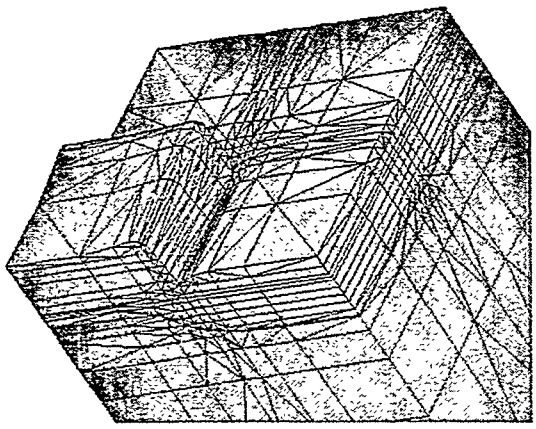


Figure 8. LOCOS mesh. Mesh after locos oxidation (2320 nodes).

Three Dimensional MOSFET

Figure 9 shows a simple 3D MOSFET (Metal Oxide Semiconductor Field Effect Transistor) useful for studies of channel length and width effects. Refinement based on the source and drain-doping profiles has been performed. The mesh contains 2870 mesh nodes. Figure 10 shows a more realistic MOSFET. A process simulation was performed and the device shown in the figure is the result. Subsequently, the model was used for device simulation. This mesh contains 99,404 elements and 41,114 nodes.

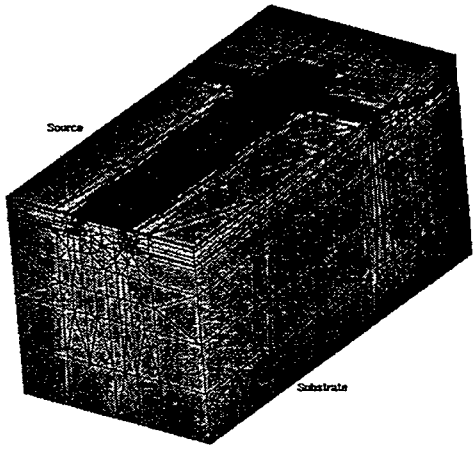


Figure 9. Simple 3D MOSFET structure for device simulation.

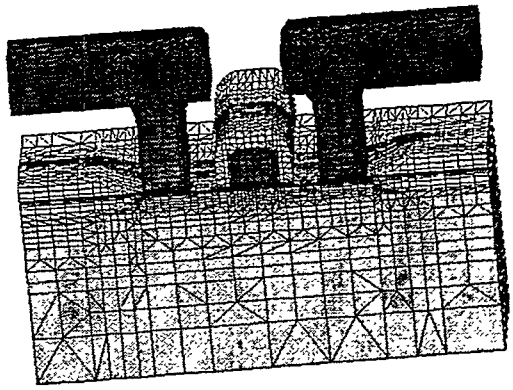


Figure 10. Realistic 3D MOSFET generated during process simulation and characterized using device simulation.

Mesh Generation using Etching and Deposition

Using etch and deposition steps, complicated structures can be generated and simulated. The deposition and etch steps manipulate the level set geometry description and, as described above, the mesh is generated to conform to this geometric description. This is shown in Figure 11 for a gear structure.

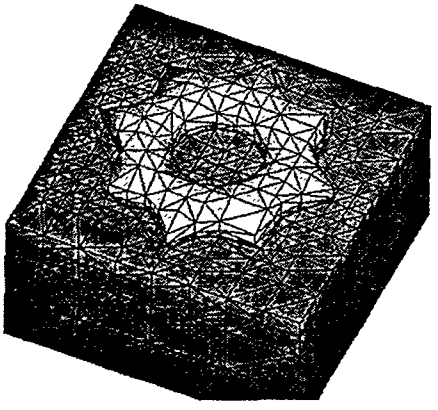


Figure 11. "Gear" structure generated via deposition and etch steps.

CONCLUSION

We have described a hybrid mesh generation method, which combines two powerful techniques: a quadtree-octree data structure for efficient access to mesh connectivity information and a level set representation of material boundaries for robust tracking of moving boundaries. The algorithm is well suited for both process and device simulation of semiconductor devices. The algorithm has been implemented in the commercially available program Taurus [8] and results generated with the code have been presented.

REFERENCES

[1] M. Yerry and M. Shephard, "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique", Int. J. Numer. Methods Eng., 20, pp. 1965-1990, 1984

[2] P. Conti, W. Fichtner, "Automatic Grid Generation for 3D Device Simulation", In Proc. of SISDEP 3 Conf., pp. 497-505,1988

[3] T. Chen, D. W. Yergeau, R. W. Dutton, "Efficient 3D Mesh Adaption in Diffusion Simulation", Proc. SISPAD'96, pp. 171-172, 1996

[4] P. Fleischmann, R. Sabelka, A. Stach, R. Strasser, and S. Selberherr, "Grid Generation for Three-Dimensional Process and Device Simulation", Proc. SISPAD'96, pp. 161-166, 1996

[5] S. Cea, M. Law, "Multidimensional Nonlinear Viscoelastic Oxidation Modeling", in Simulation of Semiconductor Devices and Processes, Vol. 6 , p.139-142,(Proc. SISDEP 95), eds. H. Ryssel, P. Pichler, Springer Verlag, 1995

[6] J. A. Sethian, D. Adalsteinson, "An Overview of Level Set Methods for Etching, Deposition, and Lithography Development, submitted for publication in IEEE Trans. Manufacturing, 1996

[7] Dan Yang, Kincho Law, Robert Dutton, "An Automated Mesh Refinement Scheme Based on the Level-Control Function", Proc. NUPAD'92, pp 181-186

[8] For information see: www.tcad.avanticorp.com/taurus.

[9] TmaLayout version 1.4.

[10] P. Conti, "Grid Generation for Three-dimensional Semiconductor Device Simulation". PhD Thesis, Swiss Federal Institute of Technology, Zurich.

[11] For information about the SCOREC mesh generation see: www.scorec.rpi.

Adaptive Hybrid Grids for Diverse Industrial Applications

Aly Khawaja *, Yannis Kallinderis †, and Harlan McMorris ‡

kallind@mail.utexas.edu

Dept. of Aerospace Engineering and Engineering Mechanics

The University of Texas at Austin

Austin, TX 78712

Abstract

An adaptive hybrid prismatic/tetrahedral grid generation method is described and applied to complex geometries for diverse industrial applications. The method consists of using a surface mesh generator to triangulate the geometry to be modeled. The volume mesh is then created using prismatic and tetrahedral elements. The prisms cover the region close to each body's surface for better resolution of viscous gradients, while tetrahedra are created in the remainder of the domain. An adaptive redistribution scheme is used to better resolve boundary layers. The grid generator is tested with various complex geometries and the resulting hybrid meshes are presented. The applicability of the adaptive hybrid grid generator over a wide range of geometries with minimal user interaction demonstrates the robustness and universality of the method.

1 Introduction

There is an ever increasing demand to perform flow simulations that incorporate the complete details of geometry as well as sophisticated flow physics. This has led to the development of numerical algorithms that can simulate the actual flow phenomena with greater fidelity. However, the success of these algorithms hinges on the grid that models the geometry. Grid generation methods for 2-D models have long existed and the general lack of complexity of these models has not quite challenged the efforts of mesh generation. However, demands for generating better 3-D geometric models for flow simulations involving complex geometries have completely changed the perspective of grid generation strategies. As a consequence, grid generation efforts have gained significance equal to that of numerical solver efforts.

Structured meshes consisting of blocks of hexahedra and unstructured grids consisting of tetrahedra have been the traditional means of discretizing 3-D flow domains [1, 2, 3, 4]. An emerging technique is that of

*Graduate Research Assistant, Member AIAA

†Associate Professor, Senior Member AIAA

‡Graduate Research Assistant, Member AIAA

using semi-structured prisms to discretize the viscous region near the surface of the geometry [5, 6, 7, 8, 9]. The prismatic elements are well suited to capture thin boundary layers. The quadrilateral faces normal to the surface provide good orthogonality and grid clustering capabilities, whereas the triangulation in the lateral direction allows flexibility in surface modeling. The structured prism layers allow implementation of directional multigrid acceleration schemes [10, 11]. The structure is also exploited when using algebraic turbulence models [12, 13]. Furthermore, numerical schemes that are semi-implicit in the normal direction can be easily used to alleviate stiffness of viscous flow computations. Finally, the structured nature leads to reduced memory requirements on the solver [7, 11].

Strongly directional viscous stresses exist in a relatively small region of the flow domain, primarily close to the geometric surface. In regions away from the surface, prismatic elements do not offer any particular advantage over other types of elements. Also, prismatic grids alone cannot cover multi-body domains. On the other hand, tetrahedra have the capability to fill any given volume of arbitrary shape. Hence, an effective strategy for grid generation would be to use both, the prisms and tetrahedra, by generating a hybrid grid [6, 7, 8]. The prisms cover the regions close to each of the bodies, while tetrahedra fill in the irregular gaps in between the prismatic layers. The tetrahedral elements may also be used to capture the viscous wakes that extend beyond the thin prismatic layers around the surface.

This paper presents an adaptive hybrid grid generation method and demonstrates the robustness of the generator by applying it to complex geometries for diverse industrial applications. A brief overview of the surface, prismatic and tetrahedral mesh generation procedures is given, along with a grid redistribution scheme. Resulting hybrid meshes for geometries from various fields such as the aerospace, turbomachinery and offshore industries are presented and discussed. Viscous flow simulations on adapted grids are also presented for a NACA 0012 wing and an offshore structure.

2 Surface Mesh Generation

Surface meshes are generated using a modified version of the advancing front method [14, 15]. The modified version uses an octree to control the spacing and stretching of points on the surface. The octree is created by starting with a master hexahedron that contains the entire domain. This hexahedron is recursively subdivided into eight smaller hexahedra called octants. This process is repeated until the octants match the local length scale of the body. Using this octree during mesh generation, the local element size is calculated using the size of the octants in that area.

2.1 Disparate Length Scales

The length scales of the surface triangles are governed by several factors. First, the local curvature is approximated by calculating dihedral angles on an isotropic initial mesh. The curvature based length scale is such that the triangulation is fine in highly curved areas (like trailing edges of blades) and relatively coarse in flat regions. The distance between surfaces is another length scale used for surface mesh generation. Here, the local length scale is small in regions where surfaces are in close proximity, thus allowing automatic clustering in such areas. Finally, the surface mesh generator also clusters points based on small length scales dictated by the

CAD definition of the model. In complex geometries, the size difference between small and large scale features is several orders of magnitude, and CAD feature based clustering allows proper refinement on tiny features while still maintaining large triangles elsewhere.

An example of a surface triangulation for a complex geometry with disparate length scales is shown in Figure 1. This case represents flow through a burner, complete with an annulus diffuser, swirl producer and a combustion chamber. The geometry has various complexities such as the fuel injection holes, severe cavities and twisted blades which produce the swirl. The geometry has periodic boundaries, and only one burner is being modeled along with periodic boundary conditions. The surface mesh has been omitted in this view to maintain clarity of the figure. Figure 2 shows a close-up of the surface triangulation for the swirl producing section. The surface consists of approximately 75,000 triangles. The disparate length scales impose a challenge for surface mesh generation. However, a combination of the curvature, proximity and CAD feature based clustering provides a good triangulation for the geometry. Views of the resulting hybrid mesh for this geometry can be found in Section 6.1.

3 Prismatic Mesh Generation

The surface mesh generated by the above described procedure is the starting point for the prismatic mesh generation. The method basically involves marching the surface triangulation away from the body in distinct steps, resulting in the generation of semistructured prismatic layers in the marching direction. The process can be visualized as a gradual inflation of the body's volume. Details on the procedure can be found in [5, 6, 7, 8]. A brief description is presented here.

3.1 Overview of Marching Procedure

There are three main aspects of the algebraic grid generation process: (i) determination of the directions along which the nodes will march (marching vectors), (ii) determination of the distance by which the nodes will march along the marching vectors, and (iii) smoothing operations on positioning of the nodes on the new layer. The marching direction is based on the *node-manifold*, which consists of the group of faces surrounding the node to be marched. The primary criterion to be satisfied when marching is that the new node should be visible from all the faces on the manifold (*visibility criterion*) [5]. The marching distances are proportional to the characteristic angle of the manifold of each node to be marched. This angle is computed using the average dot product between the pairs of faces forming the manifold. The relationship between the angles and the actual marching distance is such that the local curvature of the surface is reduced with each successive layer. After the initial directions and magnitudes are determined for each node, a number of Laplacian smoothing operations are performed on the final position of the node on the new layer, along with constraints to improve the overall quality of the generated mesh.

Another feature of the prismatic grid generator is that it allows boundary nodes to march *along* arbitrarily splined surfaces. The lack of being able to follow specified boundaries has been a key issue for most structured grid methods. The prisms march along the boundaries in the parametric domain and are then transformed back into real space, thus matching the exact geometry. The addition of this feature provides tremendous flexibility

to the grid generator.

3.2 Prismatic Mesh Generation in Narrow Gaps and Cavities

Treatment of narrow gaps and cavities between the surfaces to be modeled has been a major concern for structured and sem-istructured mesh generators. The current prismatic mesh generator is capable of automatically handling such cases using the Automatic Receding Method (ARM) [8]. The basic procedure here is to reduce the marching steps in cavities which are automatically detected by the generator. The amount of reduction depends on the size of the local cavity. Once the marching steps in the cavity regions have been reduced, a smooth transition to the regions outside the cavity is obtained by also reducing the marching steps of nodes in the immediate vicinity of the cavity to a certain extent.

4 Tetrahedral Mesh Generation

Tetrahedral meshes are generated using the same octree-advancing front method described for surface mesh generation [14]. The only difference between the surface and tetrahedral generation is the length scale used to create the octree. For hybrid prismatic/tetrahedral mesh generation, the local length scale is simply the local thickness of the last prismatic layer. This will ensure that the size of the tetrahedra in the direction normal to the outer prismatic surface is the same as the height of the neighboring prisms. This smooth transition in size from the prisms to the tetrahedra is important for accuracy of the numerical method. For an all tetrahedral mesh, the local length scale is the local edge length of the original triangulated surface.

Grids generated using an advancing front type scheme can contain regions of low quality within the mesh domain. These low quality regions must be altered before the mesh can be used with a flow solver. A method for improving low quality regions has been developed. This method removes low quality regions from the mesh and fills the resulting cavities using the same advancing front generator on the new front defined by the surface of these holes. The quality measure used in the present work is the volume ratio of the two tetrahedra sharing each face.

5 Adaptive Redistribution

Adaptive algorithms have been developed for hybrid grids which couple tetrahedral and prismatic local grid refinement strategies [11]. The present work introduces an efficient redistribution scheme to increase resolution of the boundary layers and wakes. The redistribution algorithm increases local grid resolution by clustering existing grid points in regions of interest. Since the number of grid points is fixed, re-clustering in one region may result in less resolution elsewhere when the initial grid does not include a sufficient number of points. Nevertheless, redistribution can be advantageous when the number of nodes is sufficient.

A measure of the grid resolution required normal to the *no-slip* wall is the value of $y^+ = \frac{u_\tau y}{\nu}$, with $u_\tau = \sqrt{\frac{|\tau_{wall}|}{\rho_{wall}}}$ being the wall friction velocity [16]. A criterion based on the values of y^+ at the wall is employed to *attract* nodes towards the wall so that an upper bound of y^+ is maintained at all the wall nodes. This procedure in essence determines a new value for the spacing of the first node off the wall at all locations on the wall

surface. The nodes in the prismatic region are then reclustered along the marching lines emanating from the corresponding wall node.

A NACA 0012 type of wing is considered to demonstrate the effectiveness of the developed redistribution scheme. Figure 3 shows the leading edge of the blade before and after prismatic grid redistribution. The redistributed figure clearly indicates the reclustering of the prismatic nodes toward the blade surface. Figure 4 shows the y^+ distribution along the blade before and after redistribution. The reduced $y+$ after redistribution enables better results with numerical simulations. A comparison of the pressure coefficient along the blade is shown in Figure 5. The numerical results match well with the experiments.

6 Resulting Hybrid Meshes

Hybrid grids were generated for a variety of geometries from the aerospace, turbomachinery and offshore industries to demonstrate the effectiveness of the developed grid generator. The particular cases discussed here include a burner, a torus, a High Speed Civil Transport (HSCT) aircraft configuration with flow-through engines, and a helical strake. The geometries present a wide variety of features of the grid generator such as periodicity, marching along arbitrary splined surfaces, symmetry planes and treatment of disparate length scales.

It should be noted that although the geometries presented here could be meshed via approaches other than the hybrid methodology applied, the generation procedure would not be as automatic and efficient. For example, structured approaches that require *blocking* would be extremely user intensive and time consuming. Also, all tetrahedral meshes would require substantially more elements for the same resolution in the boundary layers (typically each prismatic cell can be broken down into three tetrahedral cells). Another problem with all tetrahedral meshes is that the generation of viscous (anisotropic) elements is difficult and inefficient. Hence, the hybrid strategy chosen better addresses the demands of such complex geometries. The prismatic generator automatically creates high aspect ratio elements and the tetrahedral generator closes the remainder of the domain.

6.1 Burner Geometry

Views of the geometry were presented in Figures 1 and 2 (refer to description in Section 2.1). The surface consists of approximately 75,000 triangles. Since the case is periodic in nature, care needs to be taken to match the point placement from one periodic boundary to the other. Also, the periodic, inlet and outlet surfaces are arbitrary splined surfaces and have prismatic elements grown along them.

A hybrid mesh of the burner is seen in Figure 6, which is a cut along the axis. The view shows that the hybrid grid generator was capable of capturing all the fine features of the geometry, and also clustered points downstream of the swirl producing section. The mesh consists of 415,000 nodes, 521,000 prisms, and 748,000 tetrahedra. A cut across the swirl producing blades is shown in Figure 7. The view shows the hybrid nature of the mesh, and demonstrates the smooth transition in cell sizes even across different element types.

6.2 Combined Inlet Torus and Nozzle Ring for a Steam Turbine

This geometry consists of several small blades equally spaced circumferentially inside a toroidal housing. The case presents a challenge in grid generation efforts as there are very disparate length scales. Figure 8 shows views of the hybrid grid, which consists of approximately 350,000 prisms 30,000 pyramids, and 1,680,000 tetrahedra. The top portion of the figure shows the housing, whereas the bottom portions show close-ups of the blades. Fluid flows in through both ends of the housing and out through a right-angled bend past the blades. The surface mesh generator automatically clusters points in the vicinity of the blades due to proximity, small CAD features and high curvature. The figures indicate that the prisms have grown substantially around the housing (where there are no small scale features), and have automatically reduced the marching thicknesses around the blades. The tetrahedra match the thickness of the final layer of prisms to provide a smooth transition between element types.

6.3 High Speed Civil Transport (HSCT) Aircraft Configuration

The High Speed Civil Transport (HSCT) Aircraft is a next generation aircraft being designed to travel at supersonic speeds. It has a double-delta wing configuration emerging from the nose. This particular version consists of the conventional wing-body configuration along with a flow-through engine. A symmetry plane is used to enable modeling of only one half of the domain. Views of the surface of the HSCT aircraft are shown in Figure 9. The surface grid consists of 26,000 triangles. The top portion shows the entire aircraft without the triangulation. The flow-through engines are clearly visible. The bottom portion shows a close-up of the triangulation for the engine region. The inlet of the engine forms a 0.4° wedge and so points are clustered around it to capture the shock formation due to supersonic flow. The cavities inside the engine and in between the engine and the wing impose several length scales on the grid generator.

Views of the hybrid mesh for the geometry are shown in Figure 10. The bottom portion shows a close-up of the engine region. The grid consists of 190,000 prisms, 4,000 pyramids and 740,000 tetrahedra. It is observed that the element sizes are automatically reduced in the vicinity of the engine due to the cavity. Figure 11 shows a view of the aircraft with two perpendicular cuts. The clustering of the prismatic and tetrahedral elements in the vicinity of the engine inlet is evident.

6.4 Helical Strake Geometry

The final case considered to demonstrate the robustness of the developed grid generator is that of a helical strake geometry which is used to suppress vortex-induced vibrations (VIV) in offshore structures such as risers and spars. Of particular interest here is how incompressible fluids interact with the structures they surround, and in turn, how these structures respond to the external fluid forces. In some instances, this fluid-structure coupling can result in a resonant structural condition yielding severe structural displacements which can lead to failure of the structure. The helical strakes that are added to the cylindrical geometries disrupt the coherence of the vortex shedding, thereby substantially reducing the displacements [18].

A view of the hybrid grid for the geometry is shown in Figure 12. The grid consists of approximately 4,000 surface triangles, and 60,000 total cells. The figure shows the helical strakes attached at one end to the symmetry plane. A field cut of the hybrid mesh showing the signature of the prismatic and tetrahedral

elements is also shown. An incompressible, finite-volume solver with a pressure correction method was used to simulate low Reynold's number flow. The simulations were time accurate and were coupled with a rigid body structural response to obtain flow-structure solutions. A view of the streamlines over the geometry is shown in Figure 13. The streamlines clearly indicate the 3-D nature of the vortex structure. As mentioned above, the helical strakes are added to reduce the VIV and thus the displacements of the geometry. Figure 14 shows plots of the transverse displacement over time of cylinders with and without the strakes. It is seen that the amplitude of the displacement without the strakes was on the order of 0.35 (normalized with the diameter), whereas that with the strakes was under 0.03.

7 Concluding Remarks

An adaptive hybrid prismatic/tetrahedral mesh generator has been developed for use in modeling arbitrary 3-D geometries. The applicability, universality, and robustness of the grid generator are clearly demonstrated through complex geometries from various fields such as the aerospace, turbomachinery and offshore industries. The hybrid mesh generator was very successful in handling severe cavities and capturing widely varying length scales. The mesh generator allowed for marching along arbitrary parametric surfaces, and was also capable of generating periodic meshes. The adaptive redistibution scheme proved effective in better resolving boundary layers.

References

[1] R. Löhner, and P. Parikh, "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method", AIAA Paper 88-0515, 1988.

[2] J. Peraire, K. Morgan, and J. Peiro, "Unstructured finite element mesh generation and adaptive procedures for CFD", in *Application of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464. pp. 18.1-18.12, 1990.

[3] J. F. Thompson, Z.U.A. Warsi, and C. W. Mastin, "Numerical Grid Generation: Foundations and Applications", Elsevier Science Publishing Co., Inc., New York, 1985.

[4] M. A. Yerry, and M. Shephard, "Automatic 3-D Mesh Generation by the Modified Octree Technique", Int. J. Num. Meth. Eng., Vol. 20, pp. 1965-1990, 1984.

[5] Y. Kallinderis and S. Ward, "Prismatic Grid Generation for 3-D Complex Geometries", *Journal of the American Institute of Aeronautics and Astronautics*, Vol. 31, No. 10, pp. 1850-1856, October 1993.

[6] Y. Kallinderis, A. Khawaja, and H. McMorris "Hybrid Prismatic/Tetrahedral Grid Generation for Complex Geometries", AIAA Paper 95-0211, Reno, NV, January 1995.

[7] A. Khawaja, H. McMorris, and Y. Kallinderis, "Hybrid Grids for Viscous Flows around Complex 3-D Geometries including Multiple Bodies", AIAA Paper 95-1685, San Diego CA, June 1995.

[8] Y. Kallinderis, A. Khawaja and H. McMorris, "Hybrid Prismatic/Tetrahedral Grid Generation for Viscous Flows Around Complex Geometries," *AIAA Journal*, Vol. 34, No. 2, pp. 291-298, February 1996.

[9] K. Nakahashi, "Optimum Spacing Control of the Marching Grid Generation", AIAA paper 91-0103 , 1991.

[10] Parthasarathy, V. and Kallinderis, Y., "Directional Viscous Multigrid Method on Adaptive Prismatic Meshes", *AIAA Journal*, Vol. 33, No. 1, January 1995.

[11] V. Parthasarathy, Y. Kallinderis, and K. Nakajima, "A Hybrid Adaptation Method and Directional Viscous Multigrid with Prismatic/Tetrahedral Meshes", AIAA Paper 95-0670, Reno, NV, January 1995.

[12] Y. Kallinderis, "Algebraic Turbulence Modeling for Adaptive Unstructured Grids", *Journal of the American Institute of Aeronautics and Astronautics*, Vol. 30, No. 3, pp. 631-639, March 1992.

[13] A. Khawaja, Y. Kallinderis, and V. Parthasarathy, "Implementation of Adaptive Hybrid Grids for 3-D Turbulent Flows", AIAA Paper 96-0026, Reno NV, January 1996.

[14] H. McMorris and Y. Kallinderis, " Octree-Advancing Front Method for Generation of Unstructured Surface and Volume Meshes," *AIAA Journal*, in press.

[15] J. Peiro, J. Peraire, and K. Morgan, "FELISA System Reference Manual Parts 1 and 2", University of Wales, Swansea Reports, CR/821/94 and CR/822/94, 1994.

[16] Y. Kallinderis, J. R. Baron, "A New Adaptive Algorithm for Turbulent Flows", *Computers and Fluids Journal*, Vol. 21, No. 1, pp 77-96, 1992.

[17] J. J. Thibert, M. Granjacques, and L. H. Ohmann. *NACA 0012 Airfoil.* AGARD AR 138, 1979.

[18] K. Schulz and Y. Kallinderis, "Numerical Prediction of Vortex Induced Vibrations," in *Proceedings of the* 17^{th} *International Conference on Offshore Mechanics and Arctic Engineering*, 1998.

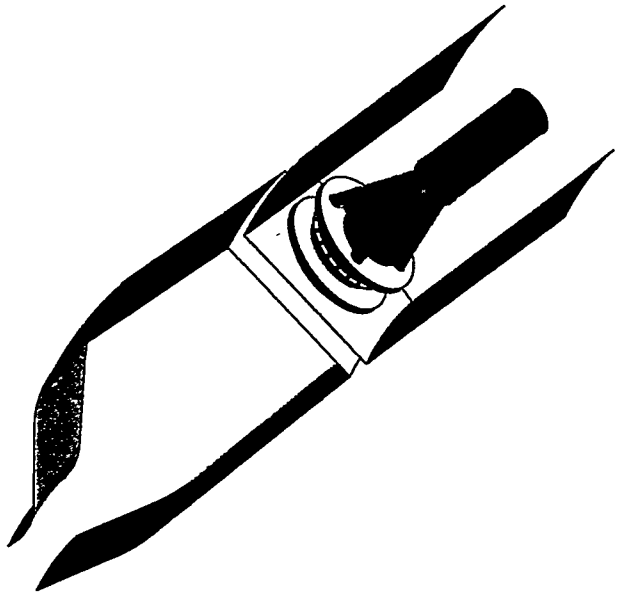


Figure 1: An isometric view of the surface of the burner geometry. The triangulation has been omitted for clarity. The geometry consists of the annulus casing, the diffuser, the swirl producer and the combustion chamber.

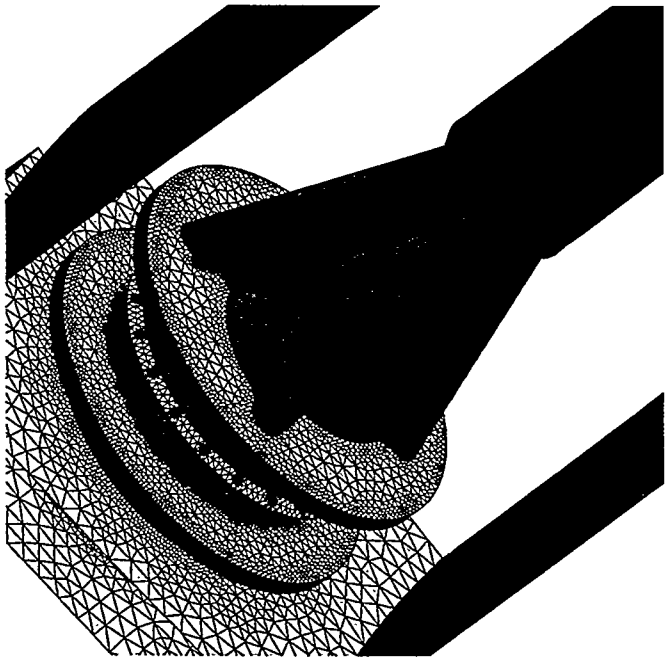


Figure 2: A close-up view of the swirl producing section of the burner geometry. The surface is made up of 75,000 triangles.

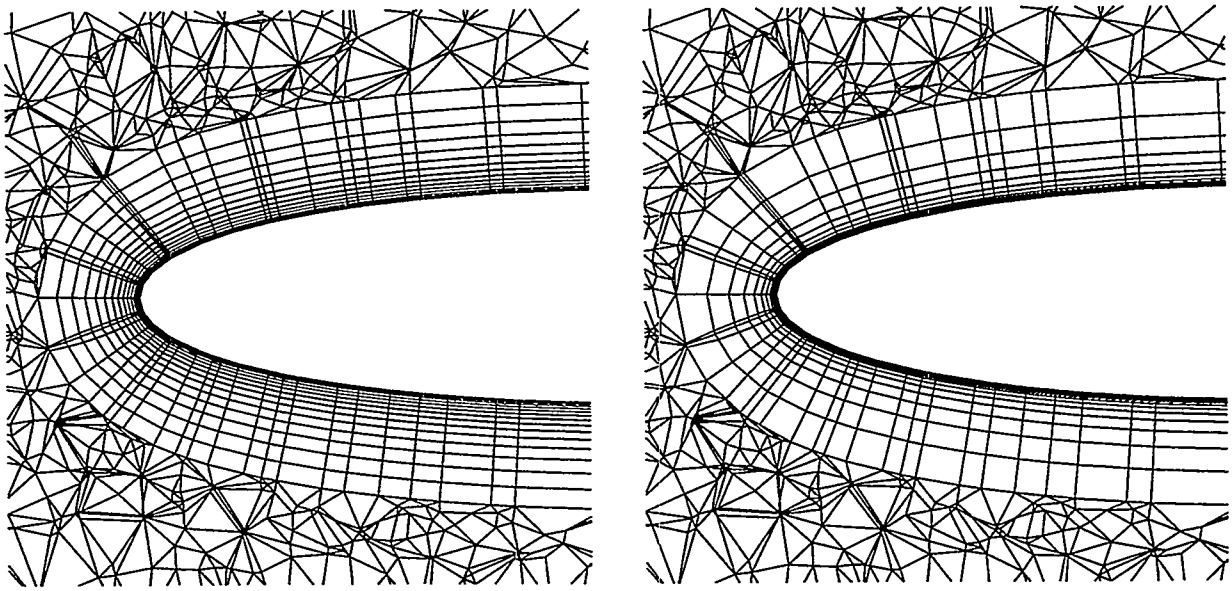


Figure 3: Close-up view of the prismatic layers at the leading edge of the NACA 0012 hybrid grid. The top portion shows the prisms before redistribution whereas the bottom portion shows the prisms after redistribution. Note that the redistributed grid clusters the grid closer to the wall surface to better resolve the boundary layer.

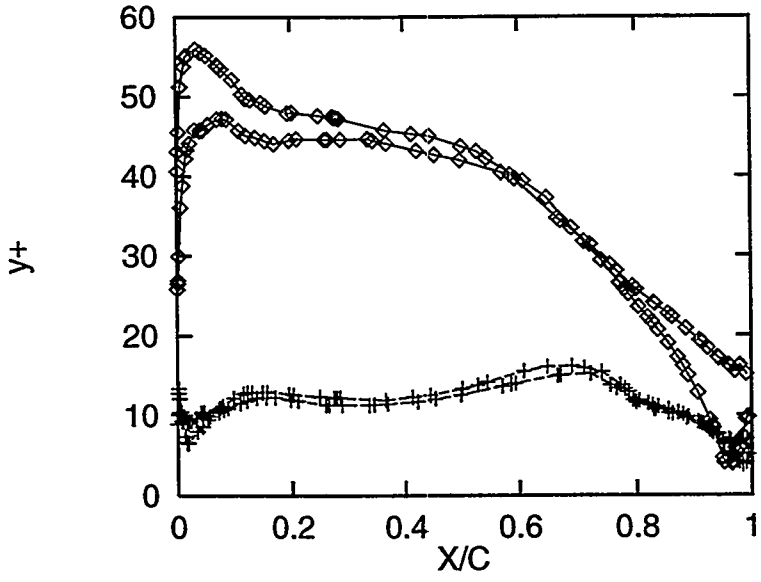


Figure 4: Comparison of the y^+ values before and after redistribution of the prism layers for the top and bottom surfaces of the NACA 0012 airfoil.
◇ Before redistribution
+ After redistribution

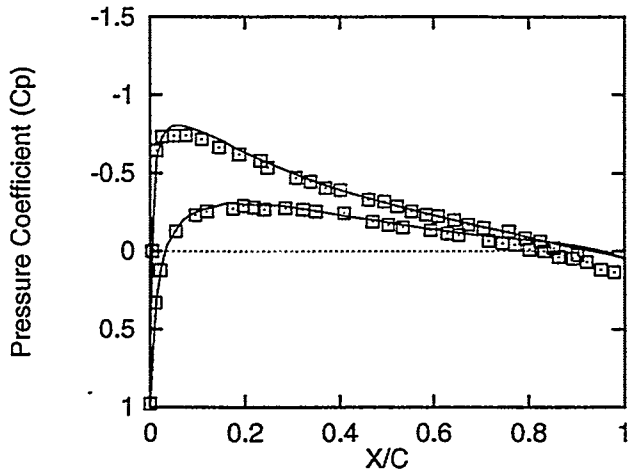


Figure 5: Comparison of pressure coefficient distributions with experiments [17]. Case of flow over the NACA-0012 wing with $Re = 2.91 \times 10^6$, $M_\infty = 0.5$, and $\alpha = 1.77^o$.
□ Experimental results
- Numerical results

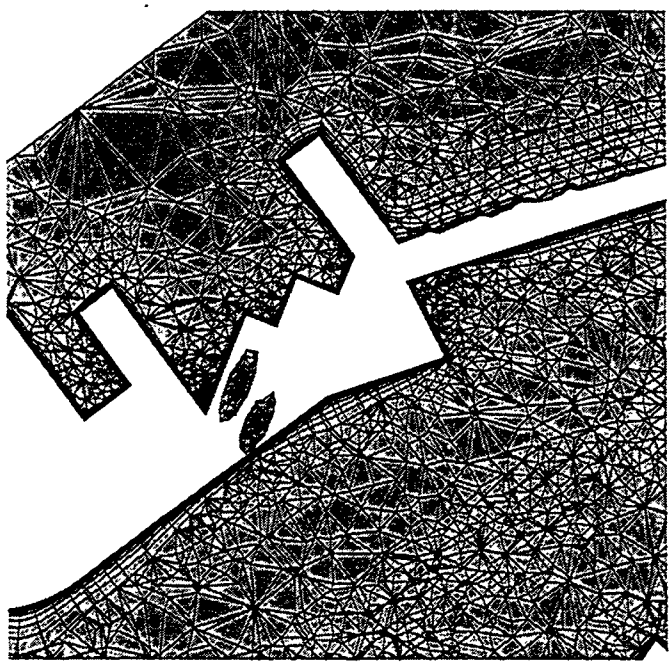


Figure 6: A close-up view of the hybrid cut along the axis of the burner geometry. The grid generator was capable of automatically handling the small length scales and the severe cavities.

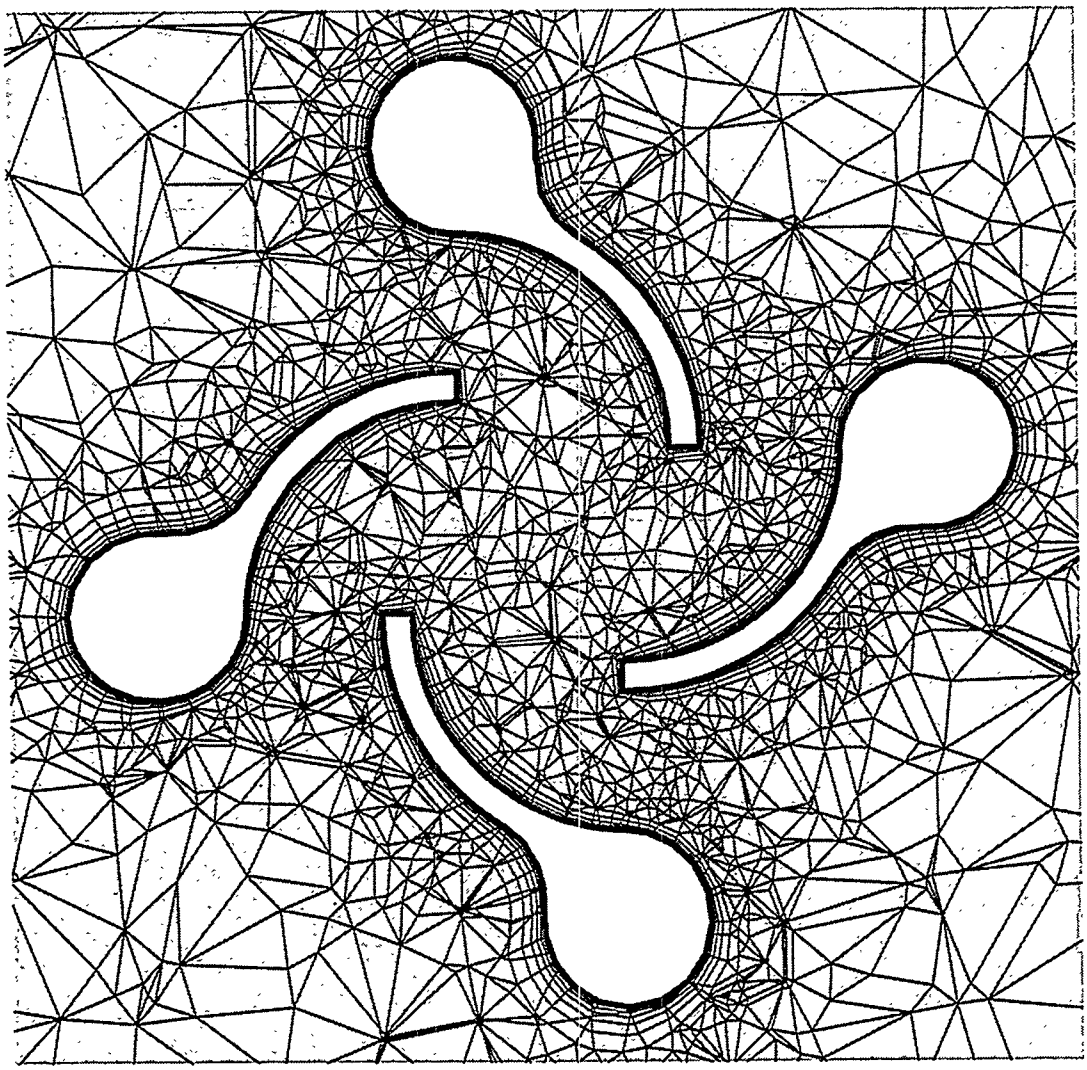


Figure 7: A cut across the swirl producing blades of the burner geometry. The hybrid nature of the mesh and the smooth transition in cell sizes is clearly visible.

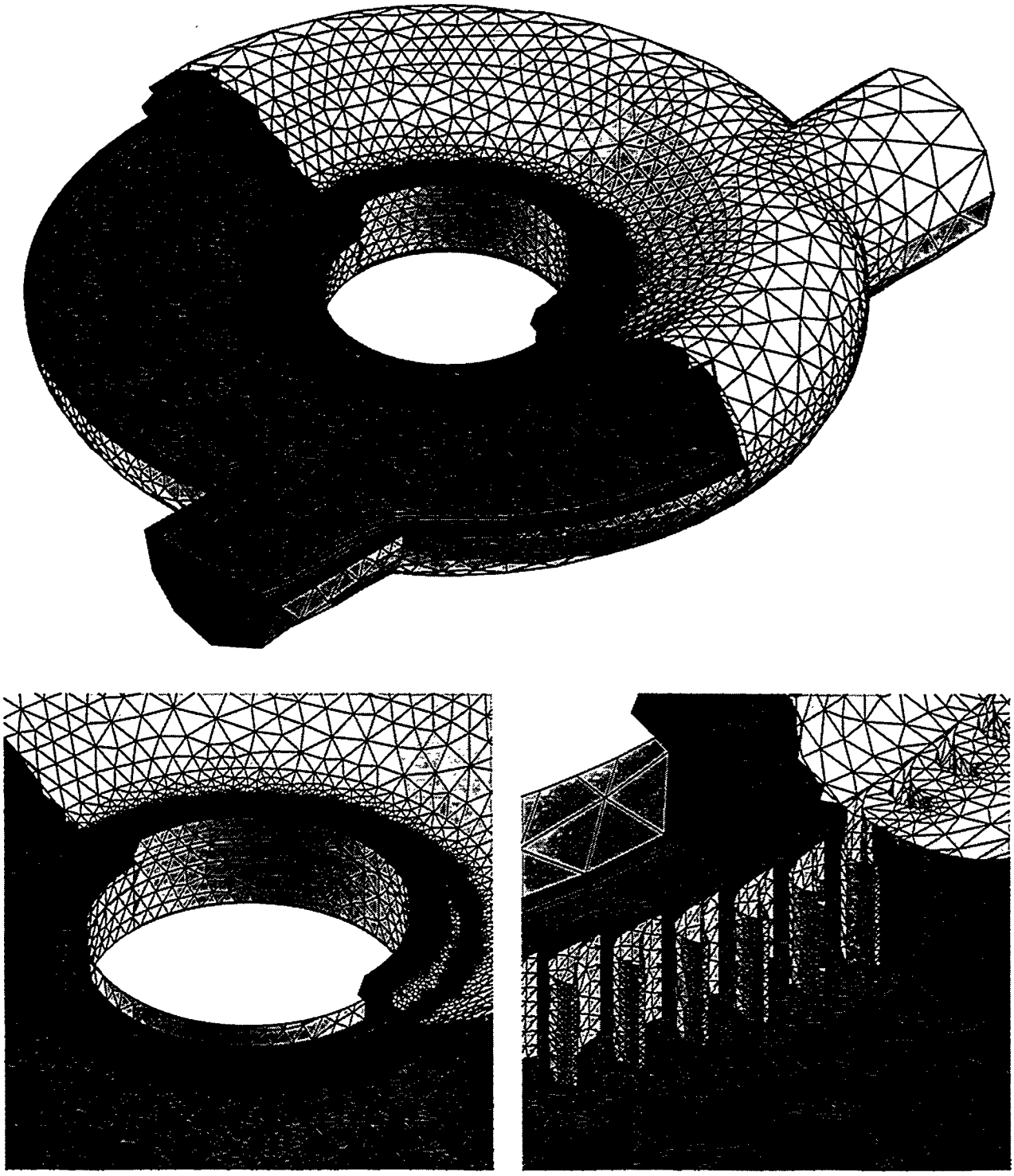


Figure 8: Views of the hybrid grid for the torus geometry. The case consists of several blades imposing vastly varying length scales. The top figure shows the housing and the bottom figures show closeups of the blades. The grid generator automatically treated the disparate length scales.

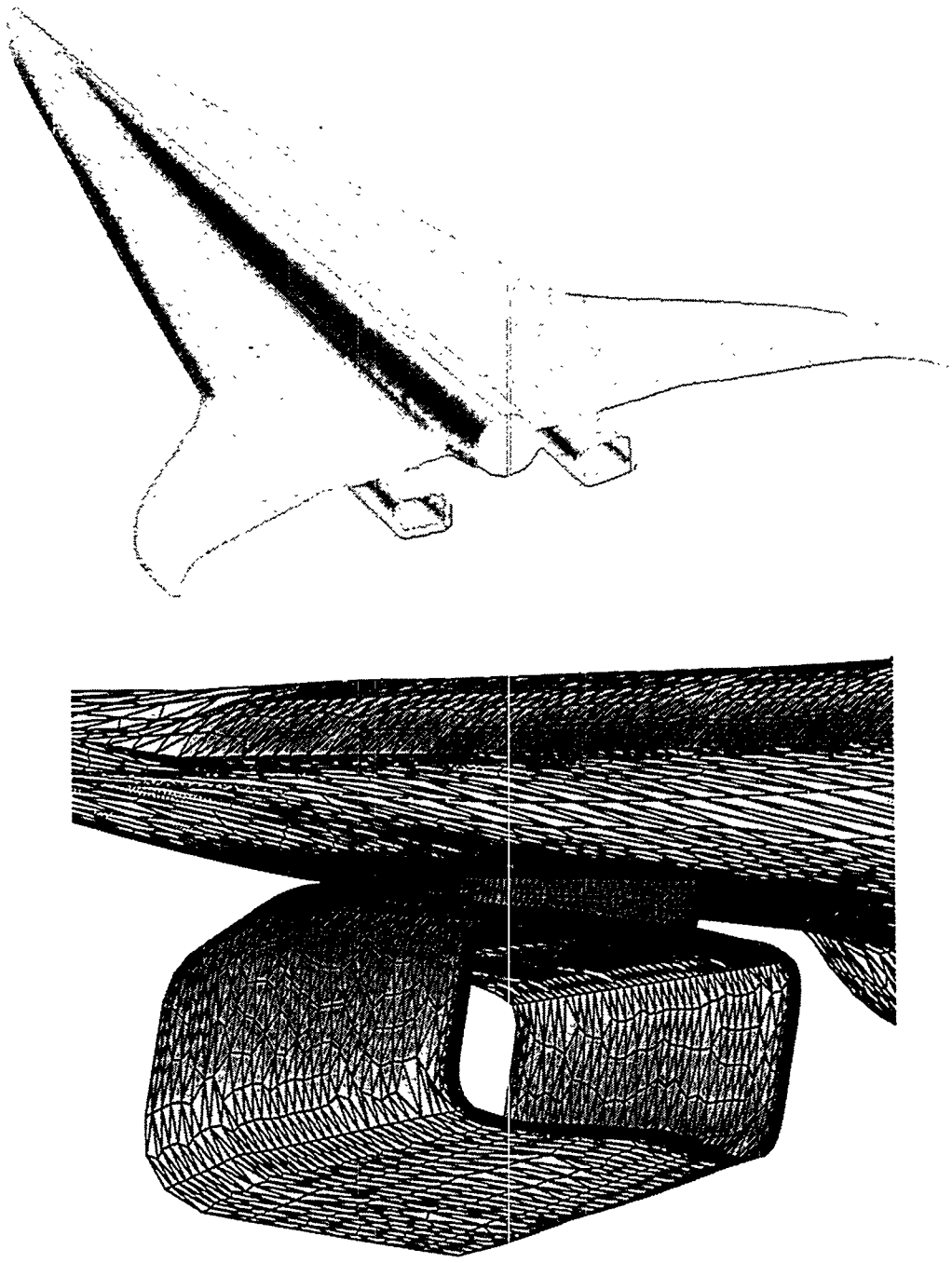


Figure 9: Views of the surface for the HSCT aircraft configuration with flow-through engines. The top portion shows the entire aircraft without the triangulation and the bottom portion shows a close-up of the engine inlet. The surface consists of 26,000 triangles.

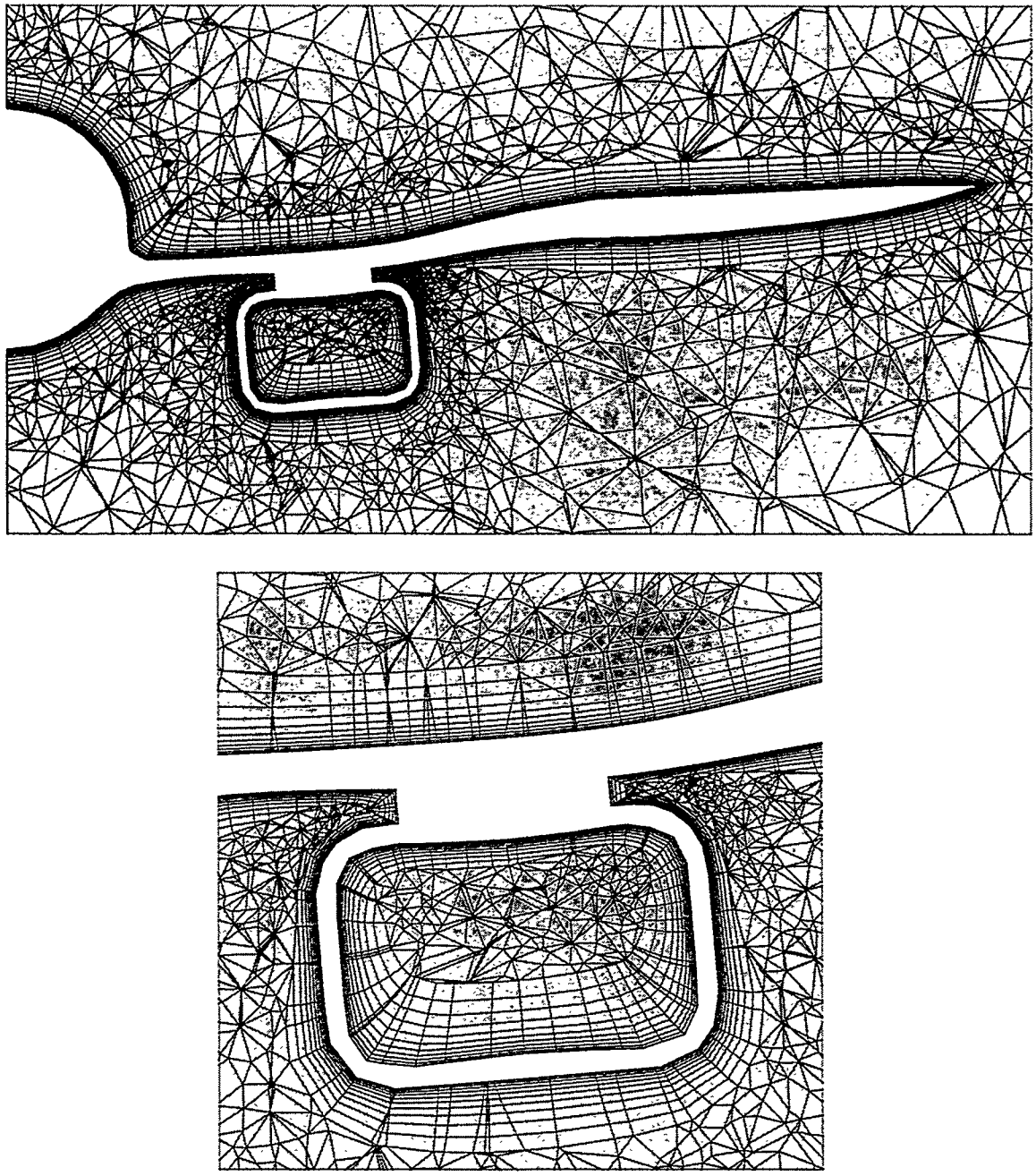


Figure 10: Plane-cuts of the hybrid mesh for the HSCT aircraft. The grid consists of 934,000 elements. The bottom portion shows a close-up of the grid around the engine cavity. The element sizes have automatically been reduced due to the cavity. A smooth transition in cell sizes is observed elsewhere.

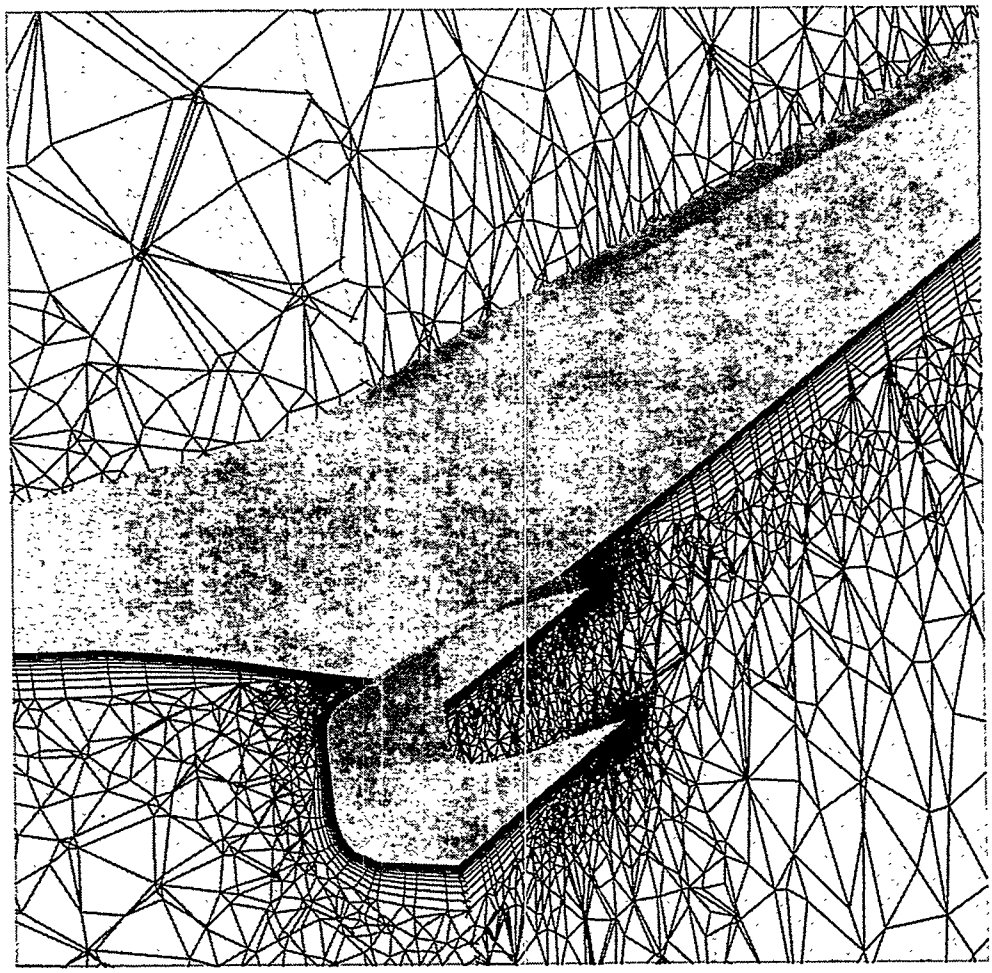


Figure 11: View of the HSCT grid with two perpendicular cuts. The view shows the clustering of elements at the engine inlet and the gradual increase in cell sizes away from it.

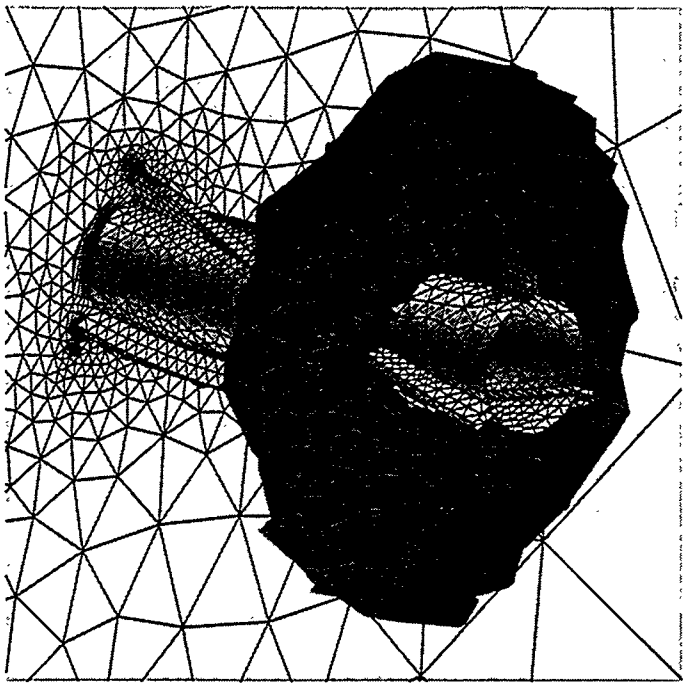


Figure 12: Hybrid grid for helical strakes geometry. The grid consists of 4,000 surface triangles and 60,000 total cells. The figure shows one end of the strakes attached to a symmetry plane, and also show a field cut of the hybrid mesh.

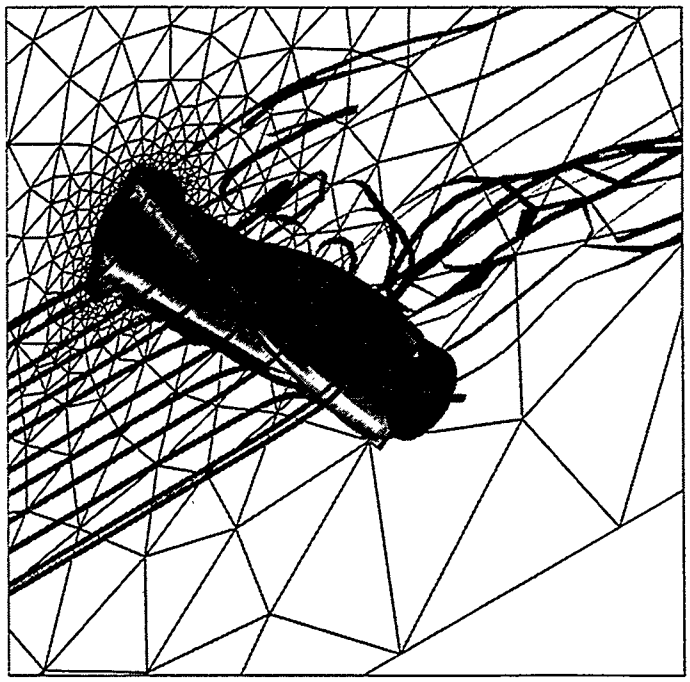


Figure 13: Streamlines over the helical strake geometry. The figure shows the 3-D nature of the vortex structure formation behind the geometry.

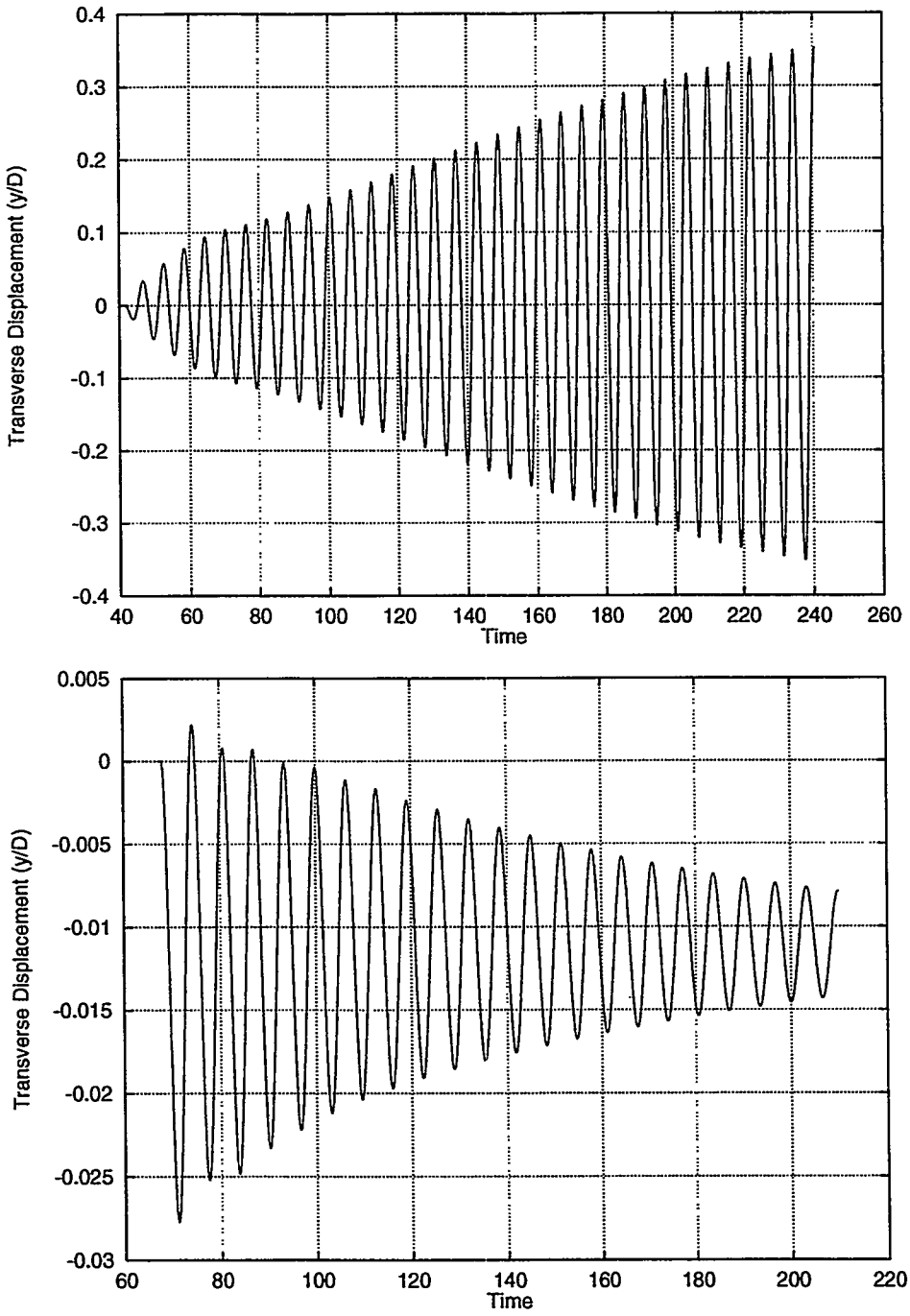


Figure 14: Displacement of elastically mounted structures vibrating freely in the transverse direction. The top figure represents a cylinder without strakes and the bottom figure represents the helical strakes geometry. It is seen that the strakes substantially reduce the transverse displacement.

Adaptive Techniques

Space-Time Meshing for Two-Dimensional Moving Boundary Problems

P. J. Zwart[1] and G. D. Raithby
Dept. of Mechanical Engineering, University of Waterloo
Waterloo, Ontario, Canada. N2L 3G1
pjzwart@sunwise.uwaterloo.ca, graith@sunwise.uwaterloo.ca

Abstract. *A new meshing strategy for two-dimensional space-time moving boundary methods is presented. The method breaks the space-time domain into time slabs, and tesselates each sequentially. Within each time slab, the mesh is generated by extruding the lower time plane, moving the boundary, adding and removing vertices near the boundary, and smoothing. Unlike other strategies, topological changes to the mesh are restricted to a thin layer near the boundary, thereby avoiding the need for global remeshing of the domain.*

Keywords: space-time, time slab, mesh generation, moving boundary

1 Introduction

The use of the finite volume method for applications in fluid mechanics and heat transfer is well-established. Its popularity stems from the direct manner in which it translates the underlying conservation principles into a practical numerical technique. Traditionally, the finite volume method has been used only to discretize the spatial domain; a finite difference method is used for time. Conservation principles, however, apply not only in space, but also in time. It makes sense to extend the finite volume principle to both space and time. We call this approach the *integrated space-time* (IST) finite volume method. It has already been developed for one-dimensional problems [12], and is currently being extended to two-dimensional cases.

Space-time methods have existed for some time in the finite element community, where they are typically used in conjunction with the discontinuous Galerkin method [6]. Among the space-time finite element methods which have been applied to moving-boundary problems are the characteristic streamline diffusion method [4,5] and the Galerkin least squares / space-time method [8,9].

Space-time methods have proven particularly popular for solving problems with moving boundaries, because they naturally incorporate terms arising from mesh motion. For instance, the Geometrical Conservation Law [1,10,11], which is particularly important for mass conservation, is identically satisfied if the space-time mesh completely fills the space-time domain. Space-time methods have also been applied to cases involving internal mesh motion, in order to achieve a conservative mesh adaptation algorithm [3].

The purpose of this paper is to present a space-time meshing procedure for two-dimensional moving boundary problems. Just as conventional discretization methods fill the spatial domain with a spatial mesh, so also space-time methods fill the space-time domain with a space-time mesh. Thus the mesh has an increased dimensionality — for two-dimensional problems, a mesh having two spatial dimensions plus a time dimension is required. Most often, the space-time mesh is split into time slabs, in order to decouple the discretization at a particular time from solutions at later times. There have been various approaches to meshing individual time slabs. Many of them use Lagrangian methods in which the space-time elements follow the flow, as in Hansbo [4,5]. In order to avoid mesh distortion and tangling, these methods may require a remeshing of the spatial domain and a nonconservative solution projection step every few time slabs. Another approach has been to remesh the spatial domain every time slab and tesselate the time slab with simplex space-time elements [3]. Such an approach seems expensive for higher-dimensional problems.

[1] Ph. D. student

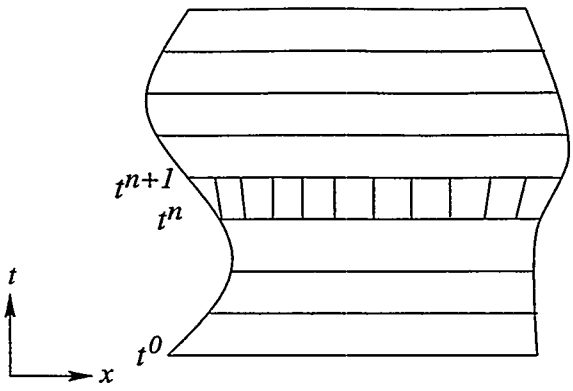


Figure 1: Division of space-time domain into time slabs.

The space-time meshing algorithm presented here avoids global remeshing entirely. Instead, it uses local mesh modifications near moving boundaries. We will first present the basic ideas behind the algorithm for one-dimensional moving boundary problems; then a more involved explanation of the method for two-dimensional problems will be presented. The solutions to some validation problems will also be provided.

2 One-dimensional problems

In this section we describe the space-time meshing algorithm for one-dimensional moving boundary problems. Space-time meshing involves filling the space-time domain with nonoverlapping *cells*, which in finite element methods are referred to as *elements*. To do so, we first subdivide the domain into *time slabs*, as shown in Figure 1. A time slab is bounded by spatial meshes on time planes at times t^n and t^{n+1}. Time slabs allow for causal solution algorithms; i.e., the solution in a particular time slab is decoupled from the solutions in later time slabs. Note that the space-time domain boundaries may be curved, reflecting the fact that the boundary positions may change with time.

In principle, the mesh within a time slab may be quite general. For simplicity, we require all cells to span the distance between times t^n and t^{n+1}. The cells have a dimensionality of one larger than the spatial dimensionality, so one-dimensional problems have two-dimensional cells. The cell boundaries are called *faces*, even though geometrically they are edges. This terminology has developed to emphasize the fact that the finite volume method represents a balance of discrete fluxes through the faces of a control volume. It is clear that space-time cells are bounded by two distinctly different types of faces: those which lie on a time plane, which we call *time faces*, and those which span the distance between time planes, which we call *space-time faces*.

When tesselating a time slab, we use an existing spatial mesh on the lower time plane t^n, to simultaneously generate the spatial mesh on t^{n+1} and the space-time mesh between the time planes. For the first time slab, the spatial mesh at time t^0 must be given. For one-dimensional problems, this initial mesh consists simply of a predefined number of vertices evenly spaced over the length of the spatial domain, as well as the time faces of length $\overline{L}$ which which join the vertices.

The time slab meshing algorithm uses a four step-process, as illustrated in Figure 2. In step (a), the lower spatial mesh is extruded in time to generate a space-time face from each vertex and a quadrilateral cell from each time face. In step (b), the boundary vertices on the new time plane are moved to their new prescribed locations. This step may produce time faces which are too long or too short or even tangled. Step (c) therefore modifies the mesh topology next to the boundary by adding or removing vertices. A vertex is

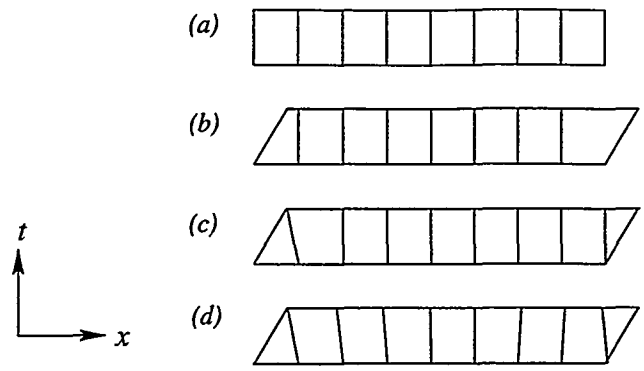


Figure 2: Generating a mesh for a time slab: (*a*) extrude in time; (*b*) move boundaries; (*c*) add/remove vertices; (*d*) smooth.

added next to the boundary if the time face length is too long,

$$L > \beta_{\max}\overline{L}, \tag{1}$$

which generates a triangular cell having a vertex on the lower time plane and an edge on the upper time plane. A vertex is removed if the time face length is too short,

$$L < \beta_{\min}\overline{L}, \tag{2}$$

which generates a triangular cell having an edge on the lower time plane and a vertex on the upper time plane. $\beta_{\max}$ and $\beta_{\min}$ are defined parameters. Finally, in step (d) the vertex locations on the new time plane are smoothed.

The algorithm as described places a restriction on the allowable time step: no more than one vertex may be added or removed next to a boundary in a time step. However, for one-dimensional problems, it is not hard to modify the algorithm to allow additional vertices to be added or removed. An example of a space-time mesh generated with this algorithm is shown in Figure 3, where a domain has its left boundary fixed at $x = 0$ and its right boundary vary according to $x = .5\sin(\pi t)$. The space-time mesh is shown for $0 < t < 3$ using time steps of $\Delta t = 0.1$ and ten cells on the initial time plane.

3 Two-dimensional problems

3.1 Overview

Space-time meshing for two-dimensional problems follows the same general approach as with one dimension, but the steps are more involved. As in the one-dimensional case, time slabs are used, with space-time cells spanning the distance between the time planes. Generating the space-time mesh requires tracking two types of topologies: the two-dimensional spatial mesh on each time plane and the space-time mesh which joins the time planes. A good space-time mesh requires having quality meshes for both of these. The initial spatial mesh for the t^0 time plane is generated by a publicly-available triangulator called EasyMesh [7]. For simplicity, we require time faces to be triangular.

For one-dimensional problems, we outlined a four-step process to tesselate a time slab. In two dimensions, three of these four steps are relatively straightforward. First, the spatial mesh on the lower time plane is extruded. In so doing, the edges generate quadrilateral space-time faces, while the triangular time faces generate triangular prisms. Next, vertices which lie on moving boundaries are moved to their new locations. Third, the mesh next to the boundary is modified as required. This third step is the most involved, and is

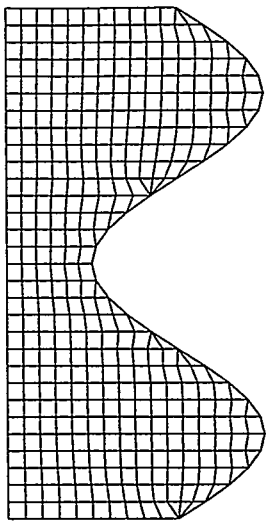


Figure 3: Space-time mesh for a one-dimensional problem having a fixed left boundary and oscillating right boundary.

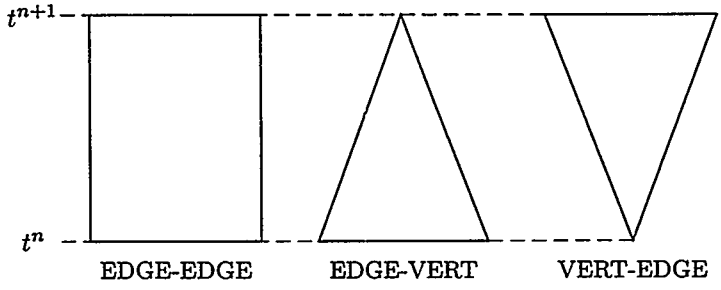
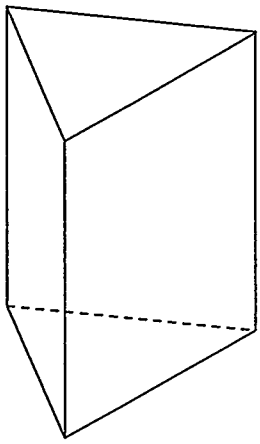


Figure 4: Space-time face topologies for two-dimensional problems.

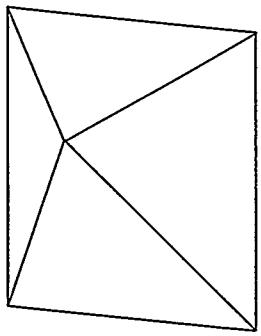
discussed in further detail below. The fourth step is a smoothing step, where one or two layers of vertices next to moving boundaries are smoothed.

Modifying the extruded mesh may change the topologies of space-time faces and cells. For the space-time faces, there are three possible topologies, shown in Figure 4. The unmodified topology is a quadrilateral, having an edge on both the lower and upper time planes. In the course of the modifications, two triangular topologies, having an edge on one time plane and a vertex on the other, may be encountered. The face types are labelled according to how they appear on the lower and upper time plane; eg., a VERT-EDGE face has a vertex on the lower time plane and an edge on the upper time plane.

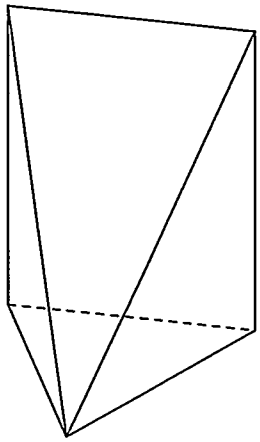
For the cells, there are six possible topologies, shown in Figure 5. The unmodified topology is a triangular prism, having a triangular time face on both the lower and upper time planes. In the course of the modifications, two pyramid topologies may be encountered, having a triangular time face on one time plane and an edge on the other. Three tetrahedral topologies are also possible: two have a triangular time face on one time plane and a vertex on the other, and the third has an edge on both time planes.



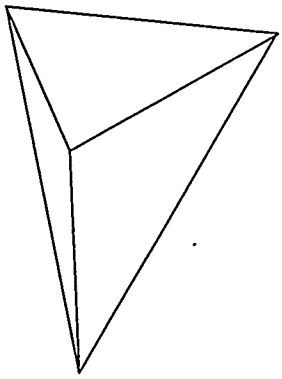
(a) TRI-TRI



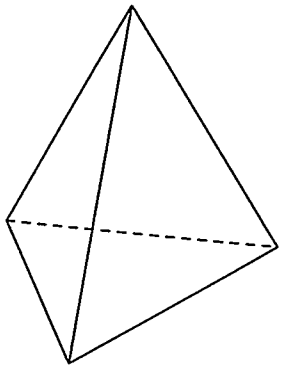
(b) EDGE-TRI



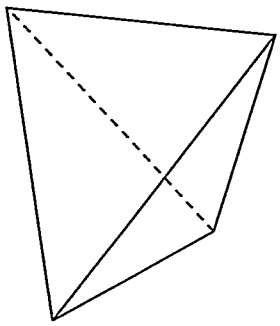
(c) TRI-EDGE



(d) VERT-TRI

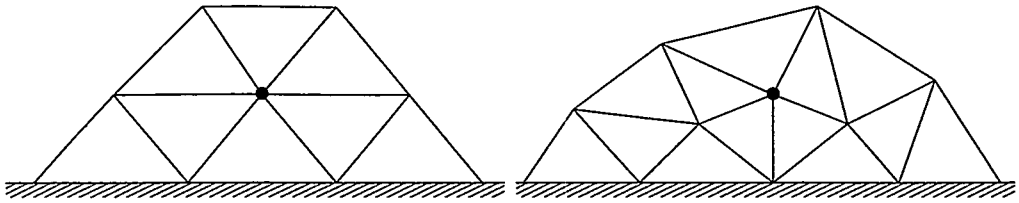


(e) TRI-VERT



(f) EDGE-EDGE

Figure 5: Cell topologies for two-dimensional problems.



(a) Face vertex (b) Edge vertex

Figure 6: Types of vertices which may be connected to a moving boundary. The mesh lies on a time plane.

Deciding where to add and remove vertices in two dimensions follows the method for one-dimensional problems. Define $\overline{L}$ to be the average edge length on the initial time plane. Suppose the distance of a vertex connected to a moving boundary to that boundary is L. Another vertex must be added if the distance is too long,

$$L > \beta_{\max}\overline{L}, \tag{3}$$

and the vertex must be removed if the distance is too short,

$$L < \beta_{\min}\overline{L}. \tag{4}$$

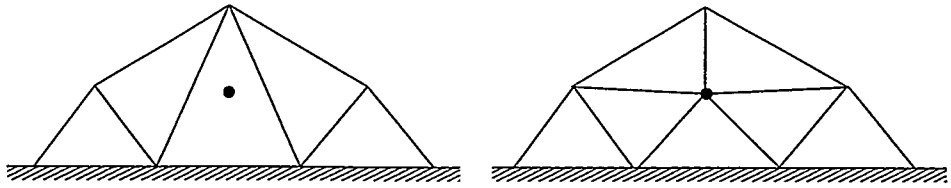
In order to maintain an acceptable mesh quality when these operations are carried out, it is essential that the spatial mesh on the new time plane retain its original structure. We therefore distinguish two types of vertices which may be connected to the boundary, as illustrated in Figure 6. The first, labelled a *face vertex*, is connected to the boundary by two edges, whereas an *edge vertex* is connected by a single edge. Removing these vertices results in different topological changes, as does adding new vertices when they are too far from the boundary. Vertices may also be added to the boundary itself, for cases where the boundary stretches or shrinks. These operations are similar to adding and removing face and edge vertices and will not be discussed in detail.

3.2 Adding a vertex

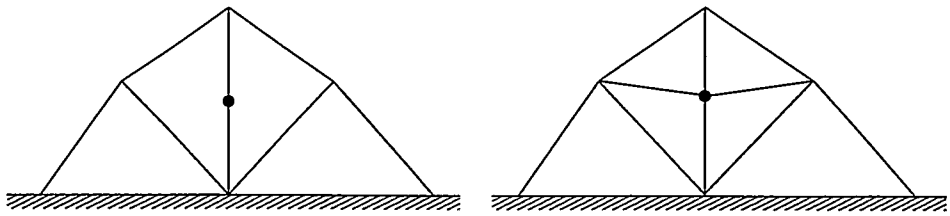
When a vertex is added, the topology both of the spatial mesh on the new time plane and of the space-time mesh below it are modified. The effect on the new time plane is relatively straightforward, as shown in Figure 7.

Consider now the effect of these operations on the topology of the space-time mesh, as shown in Figure 8. When adding an edge vertex, the topological changes can be separated into a left and a right section, each consisting of a TRI-VERT cell and two EDGE-TRI cells. When adding a face vertex, the left and right sections again consist of a TRI-VERT cell and two EDGE-TRI cells, but they no longer meet at a common plane. Instead, a central piece consisting of a TRI-VERT cell and an EDGE-TRI cell is introduced.

These operations have assumed that only a single vertex is added in isolation. In some cases, however, two adjacent vertices must be added together. For instance, suppose two adjacent time faces must be split, as shown in Figure 9. Notice the appearance of the shaded quadrilateral shape joining the two new vertices and the two old face vertices. This quadrilateral must be split in order to retain a triangulation on the new time plane. One consequence of this is that if the mesh on the old time plane is composed of equilateral triangles, the mesh on the new time plane will take on a squarish look as one of the boundaries moves out. The space-time mesh for the same operation has been split into six sections. The left and right sections of Figure 8 reappear, and the central section reappears twice. In the very center, two additional pieces appear, which arise from the interaction between the two vertices. The front piece is composed of a TRI-VERT cell

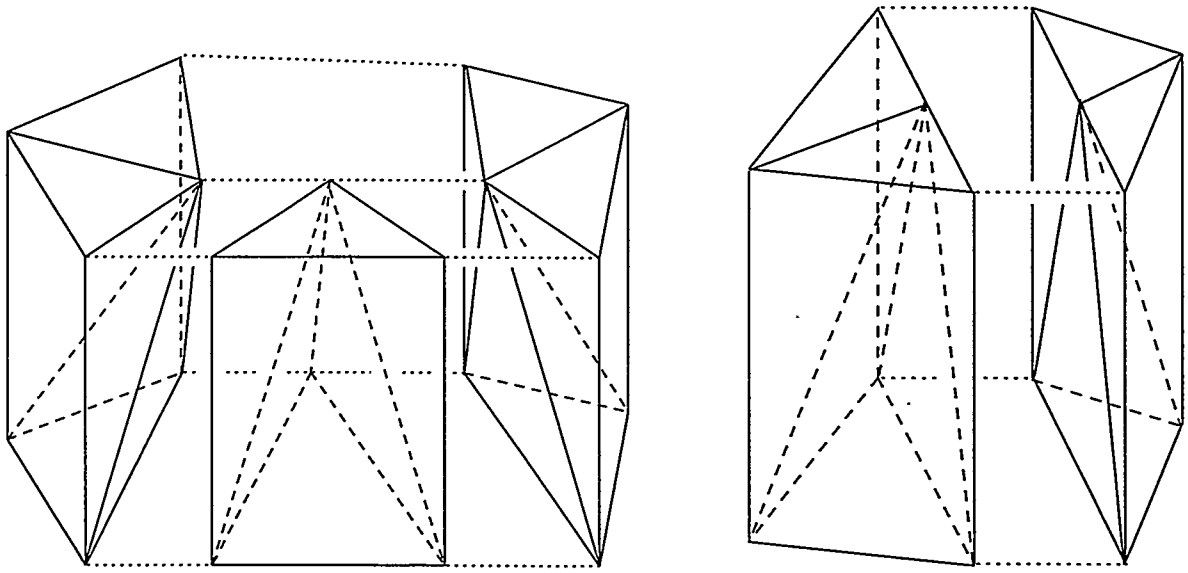


(a) Adding a face vertex



(b) Adding an edge vertex

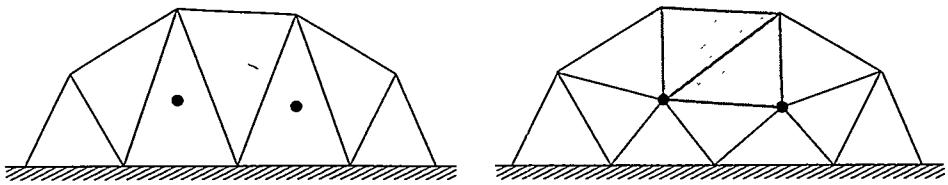
Figure 7: Topological changes to time plane when a vertex is added.



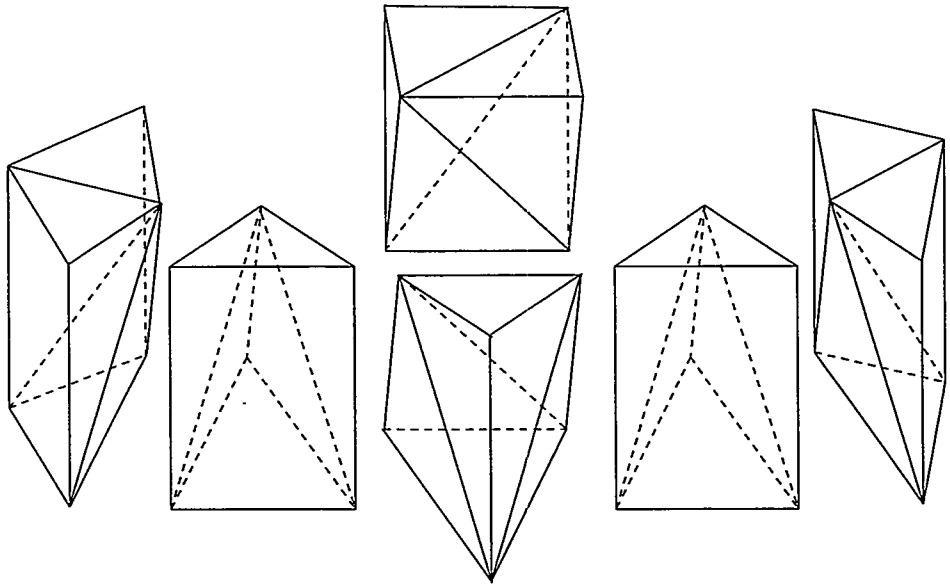
(a) Adding a face vertex

(b) Adding an edge vertex

Figure 8: Topological changes to time plane when a vertex is added. The dotted lines illustrate how the pieces glue together.



(a) Changes to time plane. The shaded quadrilateral illustrates how the mesh takes on a squarish look.



(b) Structure of time slab

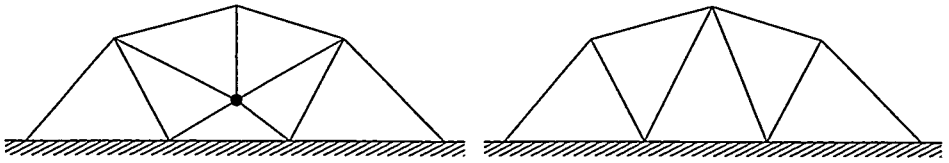
Figure 9: Topological changes when two vertices are added.

on the left and an EDGE-TRI cell on the right. The back piece has an EDGE-EDGE cell sandwiched by two VERT-TRI cells.

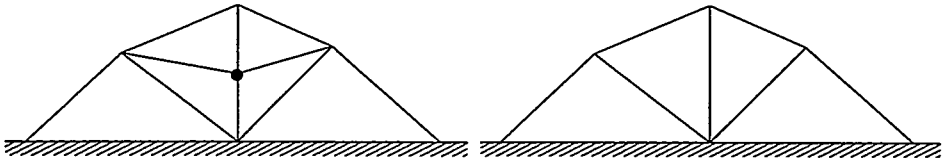
3.3 Removing a vertex

Removing a vertex is roughly the reverse procedure of adding a vertex. The topological changes to the time plane are shown in Figure 10. The operation consists of removing all edges on the time plane which touch the vertex and then retriangulating the resulting polygon. The examples shown are unambiguous, but in other cases, where the vertex to be removed has more neighbours, the retriangulation must be done in such a way that the mesh quality does not deteriorate.

The topological effects of removing a vertex are illustrated in Figure 11. The space-time cells are inverted versions of those which appear when a vertex is added (Figure 8). For every time face which shares the vertex on the old time plane, a TRI-EDGE cell appears which causes the time face to collapse into the edge opposite the vertex. In addition, for each new time face which appears on the new time plane, a VERT-TRI cell appears. When the vertex being removed has more neighbours than the examples shown here, the central section becomes more complex. Removing adjacent vertices is also more involved than removing a single



(a) Removing a face vertex



(b) Removing an edge vertex

Figure 10: Topological changes to time plane when a vertex is removed.

vertex; however, the situation is quite similar to adding adjacent vertices, and the time slab again features the EDGE-EDGE cell.

3.4 Geometry Calculations

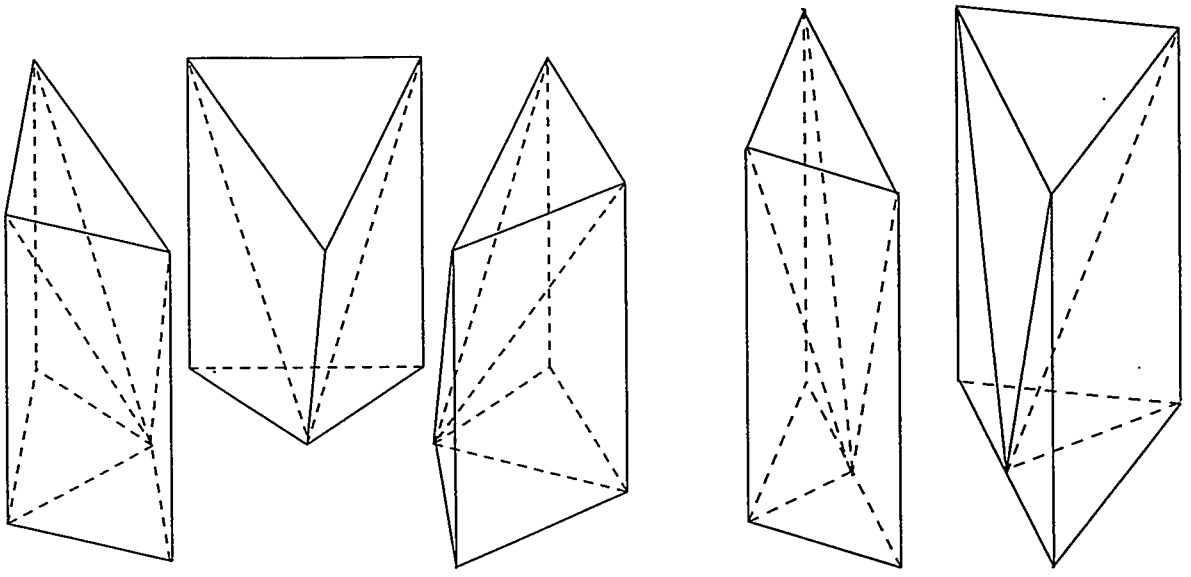
The cell-centered flow solver requires the calculation of several geometrical entities. Areas and centroids are required for time faces; areas, centroids, and normal vectors for space-time faces; and volumes and centroids for cells.

The faces and cells through most of the time slab are extruded from the lower time plane, and their geometries are easily calculated. Only near moving boundaries, where vertices have been added, removed, or moved are more extensive calculations required. These calculations are explained below.

The geometries of time faces, EDGE-VERT faces, and VERT-EDGE faces are straightforward, as they are triangular. The geometries of EDGE-EDGE faces are calculated by introducing an auxiliary point as the average of the four bounding vertices. The auxiliary point is used to decompose the face into four triangles, whose areas are summed to obtain the face area and whose centroids are weighted by area and averaged to obtain the face centroid.

A similar process is used for the cell calculations. The geometries of VERT-TRI, TRI-VERT, and EDGE-EDGE cells are easily calculated, as they are tetrahedra. For the remaining cell types, an auxiliary point is used to decompose the cell into tetrahedra: 14 for TRI-TRI cells and 8 for EDGE-TRI and TRI-EDGE cells. Cell volumes are obtained by summing the tetrahedral volumes, and cell centroids are obtained from the volume-weighted average of the tetrahedral centroids.

It should be noted that, although a Green-Gauss theorem could be used to calculate cell volumes, it is still necessary to decompose the cell into tetrahedra for the centroid calculation. This is because the Green-Gauss surface integral the centroid position has quadratic terms, which cannot be solved exactly using a single-point quadrature.



(a) Removing a face vertex

(b) Removing an edge vertex

Figure 11: Topological changes to time plane when a vertex is removed.

3.5 Test Cases

Too illustrate how the algorithm works in practice, two problems are considered. The first consists of an initially-square cavity having a dimension L. The right boundary of the square moves back and forth in a sinusoidal manner. The amplitude of the oscillation is $0.75L$ and the period of oscillation is T. The initial geometry is meshed using 670 triangles and 80 time slabs are used per period. Figure 12 shows the resulting mesh on time planes $0 < t < 1.125T$ in intervals of $0.125T$. The results clearly show that good mesh quality is maintained throughout the period.

The second test case involves fluid flow in a channel. A portion of the channel's lower boundary moves into the channel in a periodic manner. Demirdžič and Perič [2] have solved the same problem using a conventional finite volume method. The geometry is shown in Figure 13. Defining T to be the oscillation period, the normalized time is $t^* = t/T$. The height of the indentation at a particular time is defined by

$$h = .19(1 - \cos(2\pi t^*)) \tag{5}$$

and the curved portion of the lower wall is defined by

$$y = \begin{cases} 0.5h(1 - \tanh(4.14(x - 5.25))) & \text{if} \quad 4b < x < 6.5b, \\ 0.5h(1 + \tanh(4.14(x + 5.25))) & \text{if} \quad -6.5b < x < -4b. \end{cases} \tag{6}$$

The meshing algorithm performs well on this mesh. The spatial mesh at several time levels for the region around the indentation's downstream end is shown in Figure 14.

3.6 Discussion

The results for this test case and others indicates that the method works well on a variety of problems. There are some limitations, however. First, unlike the one-dimensional case, there is a CFL-type restriction

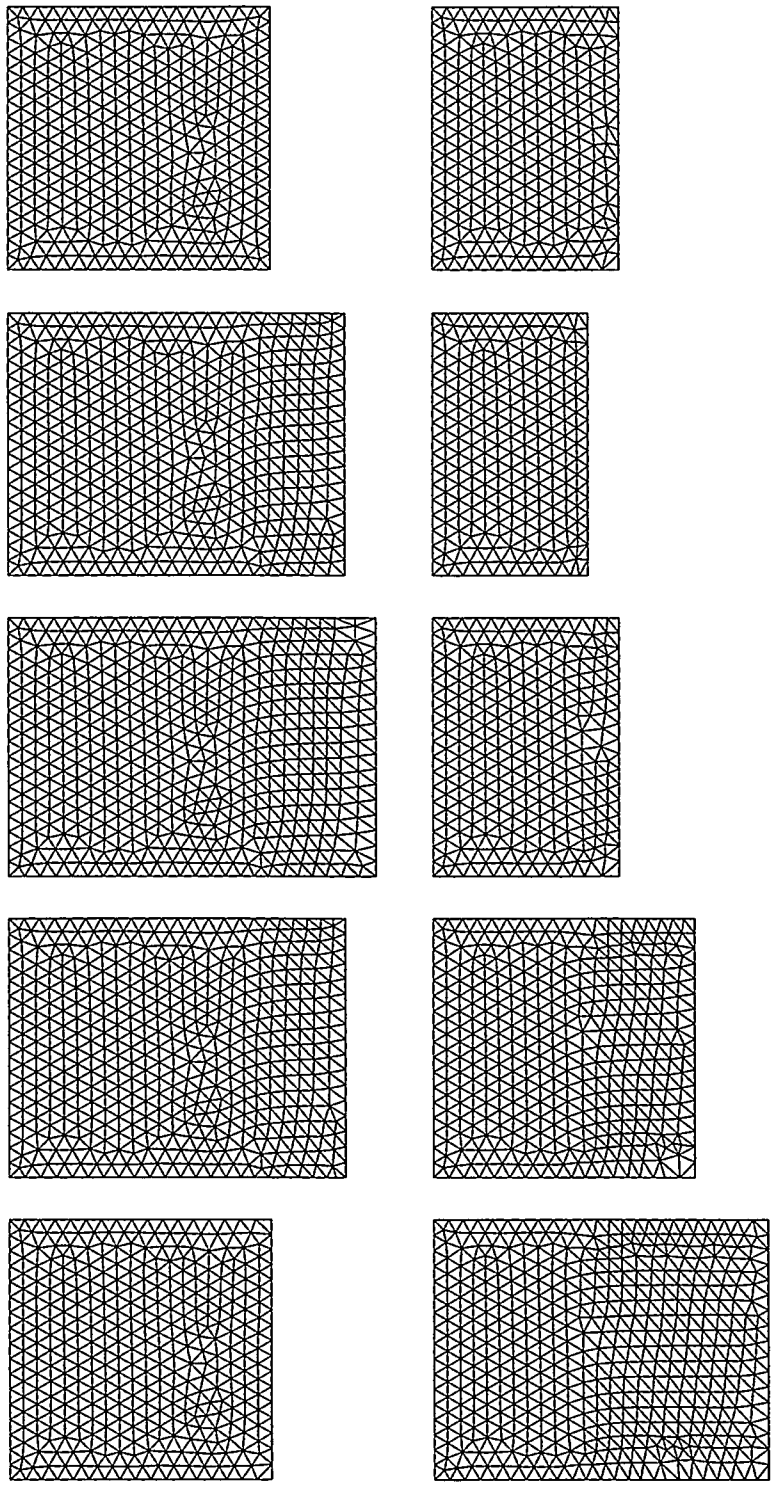


Figure 12: Spatial mesh of cavity test case on time planes $0 < t < 1.125T$ in intervals of $0.125T$. The sequence runs by column.

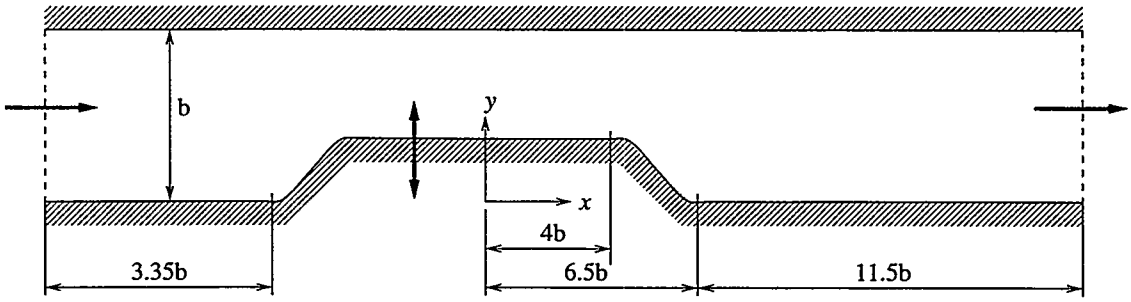


Figure 13: Geometry of test case (not to scale), with $b = 1$.

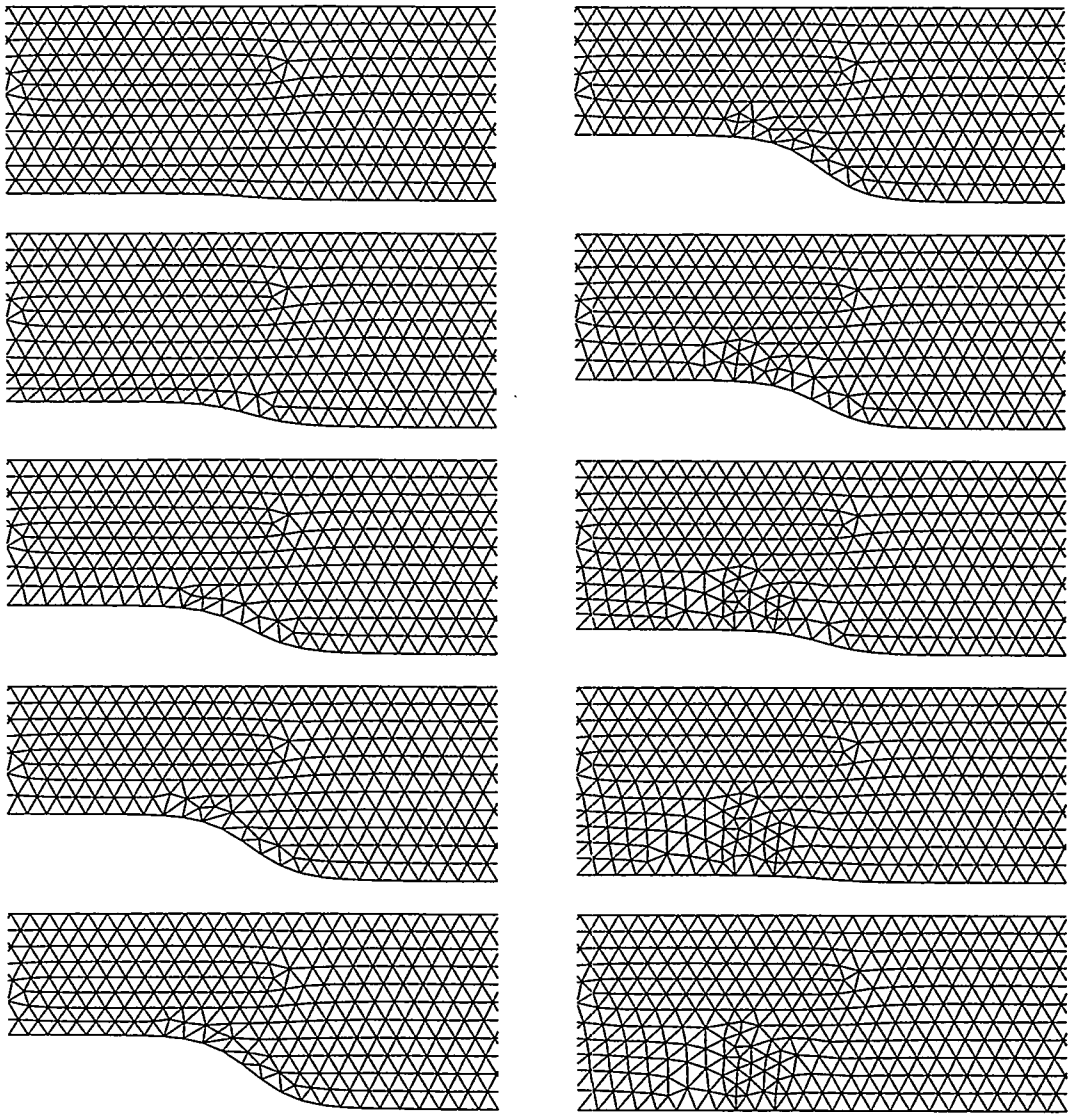


Figure 14: Spatial mesh in the downstream vicinity of the indentation on time planes $t^* = 0.1, 0.2, \ldots, 1.0$. The sequence runs by column.

on the thickness of the time slab. This is due to the excessive complexity which would be required to allow multiple layers of vertices to be added or removed in a time slab. A second limitation is that the method degenerates when the boundary vertices move in a direction significantly different than the normal vectors to the boundary time edges. Empirical observations suggest a maximum difference of about 45°. A further limitation is the assumption of a uniform isotropic spatial mesh on the time planes; if necessary, however, it should be possible to extend the algorithm to other cases as well. In addition to these limitations, the algorithm is relatively complex to code: there are several special cases, particularly near corners, which must be explicitly considered.

Despite these limitations, the method has some significant advantages. The main advantage is its computational efficiency. By using a four-step procedure of extruding the mesh on the old time plane, moving the boundary vertices, adding and removing vertices near the boundary, and smoothing on the new time plane, mesh quality is maintained while avoiding global remeshes. Future work will involve coupling the algorithm with a free-surface flow solver.

Acknowledgements

The authors are grateful to the Natural Sciences and Engineering Research Council of Canada (NSERC) and to AEA Technology for financial support.

References

[1] I. Demirdžić and M. Perić. Space conservation law in finite volume calculations of fluid flow. *International Journal for Numerical Methods in Fluids*, 8:1037–1050, 1988.

[2] I. Demirdžić and M. Perić. Finite volume method for prediction of fluid flow in arbitrarily shaped domains with moving boundaries. *International Journal for Numerical Methods in Fluids*, 10:771–790, 1990.

[3] A. M. Froncioni, A. Garon, and R. Camarero. *h*-adaptive strategies for space-time finite elements. In *Proceedings of the 3rd Annual Conference of the CFD Society of Canada*, pages 47–54, 1995.

[4] P. Hansbo. The characteristic streamline diffusion method for convection-diffusion problems. *Computer Methods in Applied Mechanics and Engineering*, 96:239–253, 1992.

[5] P. Hansbo. The characteristic streamline diffusion method for the time-dependent Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 99:171–186, 1992.

[6] T. J. R. Hughes and G. M. Hulbert. Space-time finite element methods for elastodynamics: formulations and error estimates. *Computer Methods in Applied Mechanics and Engineering*, 66:339–363, 1988.

[7] B. Niceno. *EasyMesh 1.4*. World Wide Web, http: //www-dinma.univ.trieste.it /~nirftc /research /easymesh.

[8] T. E. Tezduyar, M. Behr, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces — The deforming-spatial-domain / space-time procedure: I. the concept and the preliminary numerical tests. *Computer Methods in Applied Mechanics and Engineering*, 94:339–351, 1992.

[9] T. E. Tezduyar, M. Behr, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces — The deforming-spatial-domain / space-time procedure: II. computation of free-surface flows, two-liquid flows, and flows with drifting cylinders. *Computer Methods in Applied Mechanics and Engineering*, 94:353–371, 1992.

[10] P. D. Thomas and C. K. Lombard. Geometric conservation law and its application to flow computations on moving grids. *AIAA Journal*, 17:1030–1037, 1979.

[11] H. Zhang, M. Reggio, J. Y. Trépanier, and R. Camarero. Discrete form of the GCL for moving meshes and its implementation in CFD schemes. *Computers & Fluids*, 22:9–23, 1993.

[12] P. J. Zwart, G. D. Raithby, and M. R. Raw. An integrated space-time finite volume method for moving boundary problems. *AIAA Paper 98-0518*, 1998.

Simultaneous Refinement and Coarsening: Adaptive Meshing with Moving Boundaries *

Xiang-Yang Li Shang-Hua Teng Alper Üngör

Abstract. *In the numerical simulation of the combustion process and microstructural evolution, we need to consider the adaptive meshing problem for a domain with a moving boundary, in which, the submesh in the region behind the moving boundary needs to be coarsened while the submesh in the region ahead of the moving boundary needs to be refined. In this paper, we present a unified scheme for simultaneously refining and coarsening a mesh. Our method guarantees that the resulting mesh is well-shaped and is of a size that is within a constant factor of the optimal possible. We also present several practical variations of our provably good algorithm.*

keywords. adaptive meshing, coarsening, refinement, mesh generation, moving boundary, sphere-packing, Delaunay triangulation.

1 Introduction

In the numerical simulation of many problems, we need to handle evolving meshes which change as a function of time or the number of iterations of a numerical procedure. There are two basic scenarios where we need to adaptively and dynamically generate proper evolving meshes:

- **Adaptive refinement based on posterior error analysis:** In the numerical simulation of time-independent problems, we apply an iterative procedure which first generates a mesh based on *a priori* estimates of the local mesh density, solves the numerical system defined on the initial mesh, and then based on the posterior error analysis, adaptively refines the mesh and repeats the steps for the numerical solution and adaptive refinement.
- **Dynamic meshing with a moving boundary:** In the numerical simulation of time-dependent problems such as the combustion process and microstructural evolution, we need to consider the adaptive meshing problem for a domain with a moving boundary, in which, as a function of time, the mesh need to be dynamically changed to be effective for the next step simulation.

In both cases, submeshes in some parts of the domain need to be refined, while submeshes in some other parts need to be coarsened. For example, the moving boundary of a time-dependent problem could divide the domain into two regions: the front region and the back region. See Figure 1. During the simulation, numerical conditions in the front region become stronger, requiring the submesh in the front region to be refined. In contrast, the submesh in the back region needs to be coarsened. Therefore, we need to develop a unified framework for simultaneous mesh refinement and coarsening.

In this paper, we present a sphere-packing based scheme that simultaneously refines and coarsens a mesh M. It constructs the new mesh M' as the following.

1. Based on a dynamic mesh density estimation procedure, compute the new spacing at each mesh point in M;
2. Determine the coarsening factor of each mesh point referred as a C-point whose new spacing is larger than the previous one (such as for mesh points in the back region), and the refining factor of each mesh point referred as an R-point whose new spacing is smaller than the previous one (such as for mesh points in the front region);
3. Properly scale up the spheres of all C-points and scale down the spheres of all R-points, and fill the gaps among the shrunk spheres with new spheres of proper sizes;
4. From the sphere system, construct the point set of the new mesh;
5. Use Delaunay triangulation to generate the new mesh M'.

We will show that our method guarantees that the resulting mesh is well-shaped and is of a size that is within a constant factor of the optimal possible. We also present some practical variations of the algorithms in section 5.

*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. Supported in part by an NSF CAREER award (CCR-9502540), an Alfred P. Sloan Research Fellowship, and the U.S. Department of Energy through the University of California under Subcontract number B341494 (DOE ASCI).

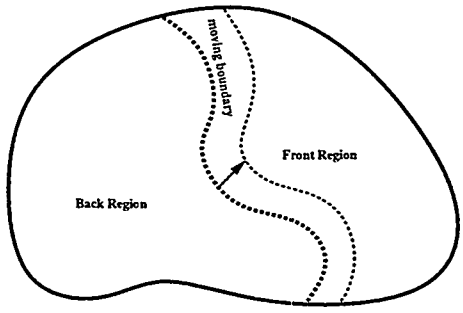


Figure 1: Domain with a moving boundary.

2 The Evolving Mesh Problem

In this section, we define the Evolving Meshing Problem which is more general than the dynamic meshing problem with a moving boundary. We will also introduce notion that will be needed in this paper.

A *mesh* M is a discretization of a domain Ω into a collection of simple elements. We consider unstructured triangular meshes which have varying local topology and spacing, and in which each element is a simplex, i.e., a triangle in 2D or a tetrahedron in 3D. The use of unstructured meshes is necessary for simulating irregular engineering problems, such as the problems that are considered in this proposed project, with fewer mesh elements [2, 9, 11].

Numerical approximation errors depend on the *quality* of the mesh, while the time and the space requirements of numerical algorithms are a function of the number of mesh elements. To properly approximate a continuous function, in addition to the conditions that a mesh must conform to the boundaries of the region and be fine enough, each individual element of the mesh must be *well-shaped*. A common shape criterion for elements is the condition that the angles of each element are not too small, or the aspect ratio of each element is bounded [1, 2, 14]. In this paper, we measure the quality of a triangular mesh by the *radius-edge aspect ratio* defined by Miller, Talmor, Teng, and Walkington [9, 10]. The radius-edge aspect-ratio of a simplex is the ratio of the circum-radius to the length of the shortest edge to of the simplex. A mesh M is *α-well-shaped* for a constant $\alpha > 1$ if the radius-edge aspect-ratio is bounded from above by α. In two dimensions, these definitions are equivalent in the sense that if a triangle is bounded away from being an ill-shaped triangle under one aspect-ratio, it is bounded away under the others as well.

A spacing function specifies how fine a mesh should be at a particular region. Given a well-shaped mesh M over a domain Ω, there are several ways to describe its spacing function:

- **[Edge-length function, el_M]** for each point $x \in \Omega$, $el_M(x)$ is equal to the length of the longest edges of all mesh simplex elements that contain x (note that points on the lower dimensional faces of a simplex are contained in more than one element).
- **[Nearest-neighbor function, nn_M]** Let x be a point in Ω, there are two cases. (1) if x is a mesh point, then $nn_M(x)$ is equal to the distance of x to the nearest mesh point in M. (2) if x is not a mesh point, then $nn_M(x)$ is equal to the distance to the second closest mesh point in M.

Lemma 1 ([9]) *If M is an α-well-shaped, then there exists constants c_1 and c_2 depending only on α such that for all point $x \in \Omega$,*

$$c_1 el_M(x) \leq nn_M(x) \leq c_2 el_M(x).$$

As shown in [9, 12], the spacing function for a well-shaped mesh should be smooth in the sense that it changes slowly as a function of distance. Formally, a function f is *Lipschitz with a constant α* if for any two points x, y in the domain, $|f(x) - f(y)| \leq \alpha ||x - y||$.

We now define the Evolving Mesh Problem.

Definition 1 (The Evolving Mesh Problem (EMP)) *The input to the problem has two parts: (1) a well-shaped mesh M and (2) a list of positive reals δ, one for each mesh point, i.e., associated with each mesh point p is a real number $\delta(p)$, such that $l \leq \delta(p) \leq L$ for constants $0 < l < 1$ and $L > 1$.* [1]

We would like to construct a new mesh M' with the following properties:

- *For each mesh point p in M, $nn_{M'}(p) \leq \delta(p) nn_M(p)$;*
- *M' is well-shaped; and*
- *the size of M' is as small as possible.*

For each mesh point $p \in M$, if $\delta(p) > 1$, then it is a C-point (where C stands for coarsening); if $\delta(p) < 1$, then it is a R-point (where R stands for refinement). Our definition of the Evolving Mesh Problem allows some part of the mesh to be coarsened while some other part to be refined.

To model the dynamic meshing problem with a moving boundary by EMP, we can define δ as a function of the moving boundary. For example, we can define the new spacing of a mesh point by applying a proper monotonic function to the distance from it to the closest point on the moving boundary. EMP is more general in the sense that it does not require any correlation in the change of δ among mesh points.

EMP is very closely related with adaptive mesh generation. In the literature, adaptive mesh generation is rather a general term. It has been used as adapting the mesh to the domain geometry or to the error analysis in the mesh generation. Most often, it has been used to refer the problem for adaptive refinement based on new error bound. While we would like to use EMP to emphasize the problem of simultaneously refinement and coarsening of a mesh. The importance of our scheme is being able to handle these both cases that the domain geometry or the dynamic error analysis might bring up.

3 An Adaptive Scheme for Evolving Meshes

The objective of our algorithm for the Evolving Mesh Problem is to use the structure of the current mesh M as much as possible and as efficient as possible. First, for each mesh point p in M, we compute the value of $nn_M(p)$. Because M is well-shaped, it has a linear number of elements and edges in terms of the number of mesh points $|M|$ [9]. Therefore, nn_M can be evaluated in $O(|M|)$ time.

We now extend the spacing-function-based coarsening technique of Miller, Talmor, and Teng [7] to simultaneously refine and coarsen a mesh. The algorithm of [7] does not directly apply to EMP. See the end of Section 4 for a detailed discussion.

For each mesh point p in M, we define a local spacing function $f_p(x)$ as

$$f_p(x) = \delta(p) nn_M(p) + ||x - p||.$$

This spacing function increases with the distance, and has Lipschitz constant 1. The global spacing $f(x)$ is then given as

$$f(x) = \min_{p \in M} f_p(x).$$

In other words, f is the lower envelope of all local spacing functions. It is easy to show that f is 1-Lipschitz [7].

Lemma 2 *For any mesh point p, if $\delta(p) \leq 1$, then $f(p) = \delta(p) nn_M(p)$; if $\delta(p) \geq 1$, then $nn_M(p) \leq f(p) \leq \delta(p) nn_M(p)$.*

Proof. By definition, $f(p) = min(min_{q \neq p} f_q(p), f_p(p))$, and $f_p(p) = \delta(p) nn_M(p)$. It suffices to show $min_{q \neq p} f_q(p) \geq nn_M(p)$. For all $q \neq p$, $f_q(p) = \delta(q) nn_M(q) + ||q - p|| \geq ||q - p|| \geq nn_M(p)$. The last inequality comes from the definition of nn_M. Hence, $min_{q \neq p} f_q(p) \geq nn_M(p)$. If $\delta(p) \leq 1$, then $f(p) = min(min_{q \neq p} f_q(p), f_p(p)) = \delta(p) nn_M(p)$. Otherwise, $f(p) \leq f_p(p) = \delta(p) nn_M(p)$, and $\delta(p) * nn_M(p) \geq nn_M(p)$ implies $f(p) \geq nn_M(p)$. ∎

So for both cases, we have $f(p) \geq l * nn_M(p)$, and hence $nn_M(p) \leq f(p)/l$.

[1] The constant l defines the maximum degree of the refinement. The smaller the value l, the more a certain region of the mesh can be refined. In practice, l is a reasonably large constant, such as 1/4.

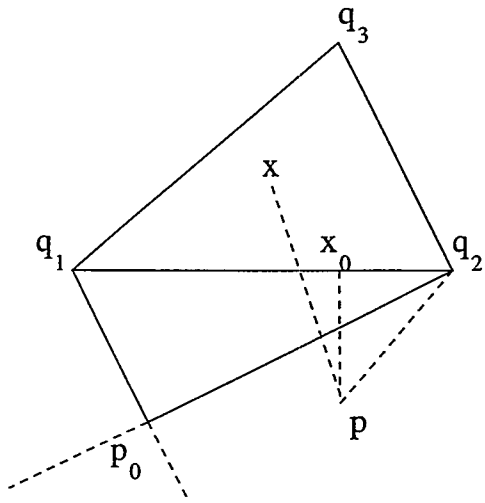


Figure 2: The distance ratio $||p-x||/min_{1\le i\le 3}||p-q_i||$ is at least $\tan(\theta)$, where θ is the lower bound on the angle of mesh elements.

Lemma 3 (Local Similarity) *For each edge $(p,q) \in M$, there exists a constant c_3 which depends only on the aspect ratio α of M and the lower bound l of δ, such that*

$$f(p) \le c_3 * f(q).$$

Proof. $f(p) \le f(q) + ||p-q|| \le f(q) + 1/c_1 * nn_M(q) \le f(q) + f(q)/(l * c_1) = f(q) * (1 + 1/(l * c_1))$. The second inequality follows from Lemma 1 that $||p-q|| \le el(q) \le nn_M(q)/c_1$. Hence $f(p) \le c_3 * f(q)$, where $c_3 = 1+1/(l*c_1)$. ∎

Similarly, for any point x in a triangle element $q_1q_2q_3$, we have $f(x) \le c_3 f(q_i)$, for $i = 1,2,3$.

Lemma 4 *Let $q_1q_2q_3$ be a triangle element of M. Let p be a mesh point other than q_1, q_2, q_3. Let x be a point inside the triangle. Then there exists a constant c_4, depending only on the smallest angle θ of M, such that*

$$||p-x||/min_{1\le i\le 3}||p-q_i|| \ge c_4.$$

Proof. There are two cases for the nearest point in the triangle to p:

- one of $\{q_1, q_2, q_3\}$; In this case, we have $||p-x||/min_{1\le i\le 3}||p-q_i|| \ge 1$.
- a boundary point other than q_1, q_2 or q_3. W.l.o.g., assume q_1q_2 separates p from q_3. Let x_0 be the closest point on the segment q_1q_2 to p. See Figure 2. If p is directly connected to q_1 and q_2 in the mesh, then $||p-x_0||/||p-q_i|| \ge \tan(\theta)$, where $i = 1,2$, and θ is the lower bound on the element angle. Otherwise, assume p_0 is the mesh point other than q_3 directly connected to q_1 and q_2 in the mesh. Either p_0q_1 separates p from q_2 or p_0q_2 separates p from q_1 or both. W.l.o.g., assume p_0q_2 separates p from q_1. Then we have $||p-x_0||/||p-q_2|| \ge \tan(\theta)$, which implies that $||p-x_0||/min_{1\le i\le 3}||p-q_i||$ is at least $\tan(\theta)$. The lemma follows from the fact that $||p-x|| \ge ||p-x_0||$.

In both cases, we have

$$||p-x||/||min_{1\le i\le 3}||p-q_i|| \ge min(1, \tan(\theta)).$$

∎

Lemma 5 *Let x be a point in a triangle element $q_1q_2q_3$ of a well-shaped M. The following is true for the global spacing function: there exists a constant c_5, depending only on the smallest angle θ of M, and the lower bound l on δ, such that*

$$f(x)/min_{1\le i\le 3}f(q_i) \ge c_5.$$

Proof. From the definition of f, there exists a mesh point p such that $f(x) = \delta(p) * nn_M(p) + ||x - p||$. If p is one of q_1, q_2, q_3, then

$$f(x) = \delta(p)nn_M(p) + ||x - p|| \geq \delta(p)nn_M(p) \geq min_{1 \leq i \leq 3} f(q_i).$$

Otherwise, let q be the q_i with the minimum distance to p. We have $f(q) \leq \delta(p) * nn_M(p) + ||q - p||$. If $||x - p|| \geq ||q - p||$, then

$$f(x) = \delta(p) * nn_M(p) + ||x - p|| \geq f(q).$$

Otherwise, $f(x)/f(q) \geq (\delta(p) * nn_M(p) + ||x - p||)/(\delta(p) * nn_M(p) + ||q - p||) \geq ||x - p||/||q - p|| \geq c_4$. The last inequality is given in Lemma 4. In both cases, we have $f(x)/f(q) \geq min(1, c_4)$. From Lemma 3, we have $f(q)/min_{1 \leq i \leq 3} f(q_i) \geq 1/c_3$. Hence

$$f(x)/min_{1 \leq i \leq 3} f(q_i) \geq min(1, c_4)/c_3 = min(1, \tan(\theta))/c_3.$$

∎

Let $B(x, r)$ be the sphere of radius r centered at point x. We will use the following notion of sphere packing [7, 15] in our algorithm.

Definition 2 (β-Packing) *Let β a positive real constant. A set S of spheres is a β-packing with centers P of Ω with respect to a spacing function f if*

- *For each point p of P, $B(p, f(p)/2) \in S$;*
- *The interiors of any two spheres s_1 and s_2 in S do not overlap; and*
- *For each point $q \in \Omega$, there is a sphere in S that overlaps with $B(q, \beta * f(q)/2)$.*

To construct the mesh points for M', we first use the following procedures to generate a β-packing of Ω with respect to f by using as many mesh points from M as possible. Here β is a constant to be given later. The mesh M' is the Delaunay triangulation of the centers of the resulting β-packing.

Algorithm `Functional-Refining-Functional-Coarsening`

1. Let $S_1 = \{B(p, f(p)/2) | p \in M\}$;
2. For each triangle element $t = (q_1 q_2 q_3)$ in M, let q be the mesh point q_i with the smallest $f(q_i)$. Let b_t be the smallest box that contains t. We divide b_t into a set of uniform cells with the side length $c_5 * f(q)/(2\sqrt{2})$, where c_5 is a constant given in Lemma 5. See Figure 3. Choose a random point in every cell that intersects t for a nonempty area, and for each such a point x, define a sphere with center x and radius $f(x)/2$. [2] Let S_2 be the set of these spheres;
3. Let $S' = S_1 \cup S_2$;
4. Order the sphere in S' as the following: all spheres whose centers are on the boundary come first, followed by all other spheres in S_1 in the order of increasing radii, followed by all spheres in S_2 in the order of increasing radii;
5. We say two spheres s_1 and s_2 in S' are *conflicting* if their interiors overlap. The conflicting relation defines a *Conflict Graph* (CG) over S'. Let S be the set of spheres which form the *Lexical-First Maximal Independent Set* (MIS) of CG;
6. Let M' be the mesh defined by the constraint Delaunay Triangulation of centers of S.

The lexical-first MIS is defined as the following. The initial MIS is empty. Then we add a sphere with the smallest index that does not conflict with any spheres of the existed MIS until no sphere can be added. The intuition is that we try to conform the boundary, and use as many spheres as possible. In addition the smaller sphere has higher priority to be chosen.

The basic idea of `Functional-Refining-Functional-Coarsening` *FRFC* is to first compute a maximum spacing function that satisfies the new spacing requirement of the Evolving Mesh Problem. We then make use of the point set of M to construct a sphere packing with respect to the spacing function, and hence the point set of the new mesh. Then the new mesh is obtained by using Delaunay triangulation. The maximality of the spacing function f is given in the following lemma.

[2] In practice, we can use $(f(q) + ||x - q||)/2$ to approximate the radius $f(x)/2$. Note that $c_5 f(q) \leq f(x) \leq f(q) + ||x - q|| \leq c_3 f(q) \leq c_3/c_5 f(x)$. In other words, $(f(q) + ||x - q||)/f(x)$ has constant lower and upper bound. Hence, it is reasonable to use $f(q) + ||x - q||$ to approximate $f(x)$.

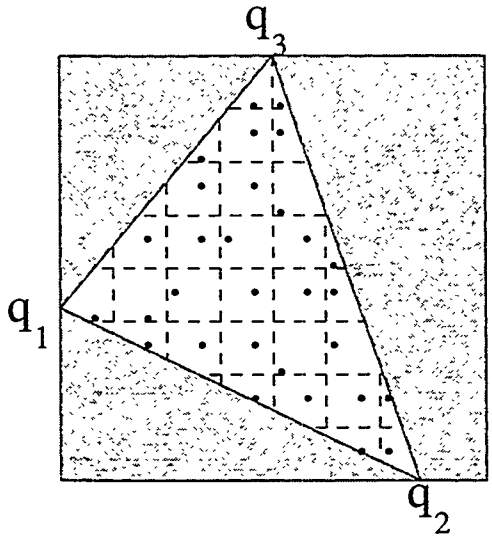


Figure 3: Sampling points in a triangle element.

Lemma 6 (Maximality) *Let g be any spacing function of Lipschitz constant 1 over the domain Ω that satisfies the condition $g(p) \leq \delta(p) nn_M(p)$ for all mesh point p in M. Then for any point q in Ω, not necessarily a mesh point of M, $g(q) \leq f(q)$.*

Proof. Let $g_p(x) = g(p) + ||x - p||, g'(x) = min(g_p(x))$. Then $g'(x) \leq f(x)$, for all x. Now assume P_x is the point that drives x to get the smallest value for g', i.e., $g'(x) = g_{P_x}(x) = g(P_x) + ||P_x - x||$. Note that $g(x) \leq g(P_x) + ||P_x - x||$ because g is 1-Lipschitz function. Then we have $\forall x, g(x) \leq g'(x) \leq f(x)$. ∎

Therefore, let M'' be any mesh that satisfies the condition of the Evolving Mesh Problem. We have for any point q in Ω, $nn_{M''}(q) \leq f(q)$, because $nn_{M''}$ is 1-Lipschitz function, and $nn_{M''}(p) \leq \delta(p) nn_M(p)$.

Lemma 7 (Number of Sample Points) *The number of sampled points in a triangle element $q_1 q_2 q_3$ is bounded by $2\sqrt{2}/(c_5 * l)$.*

Proof. Note that the number of cells generated in the triangle element is no more than $nn_M(q_i) * 2\sqrt{2}/(c_5 * f(q_i))$. The lemma follows from Lemma 2. ∎

Lemma 8 (Dense Sample) *For any point x in the domain, the sphere $B(x, f(x)/2)$ contains at least one point from $S_1 \cup S_2$.*

Proof. It is sufficient to show a stronger statement that is $B(x, f(x)/2)$ contains at least one cell generated during the sampling procedure. Let $t = q_1 q_2 q_3$ be the triangle element that contains x. Let q be a mesh point q_i with the smallest $f(q_i)$, $i = 1, 2, 3$. The side length of the cell generated during the sampling procedure is $c_5 * f(q)/(2\sqrt{2})$, where c_5 is given in Lemma 5. If the radius of the sphere is at least the diagonal of the cell, i.e., $f(x)/2 \geq c_5 * f(q)/2$, then the sphere will contain a cell. This is true by Lemma 5 that $f(x) \geq c_5 * f(q)$. ∎

The following lemma is from Miller *et al* [5].

Lemma 9 *Let P be a set of points in domain Ω. Let g be an α-Lipschitz function defined on Ω. Let $S = \{B(p, g(p)) | p \in P\}$. If for any point $x \in \Omega$, $B(x, g(x))$ contains at least one point from P, then the maximal-independent-set of the conflict graph of S is $(3 + \alpha)/(1 - \alpha)$-packing.*

Theorem 1 *The S returned by the* FRFC *algorithm is a 7-packing.*

Proof. Note that the spacing function $f/2$ used for spheres is 1/2-Lipschitz. The lemma follows from Lemma 9. ∎

Let Γ be a collection of spheres. The *ply* of a point x, denoted by $ply(x, \Gamma)$, is the number of the spheres in Γ that contains x.

Lemma 10 (Constant Ply) *For $S' = S_1 \cup S_2$, there exists a constant c_6 such that for any point $x \in \Omega$,*

$$ply(x, S') \leq c_6.$$

Proof. Let $C(x)$ be the set of the centers of the spheres in S' that contain x. For any point $c \in C(x)$, we have $2||x - c|| \leq f(c)$. Note that $f(c) \leq f(x) + ||x - c||$ because f is 1-Lipschitz function. Hence $||x - c|| \leq f(x)$, i.e., point c is in $B(x, f(x))$. Let $t = (q_1 q_2 q_3)$ be a triangle element that contains x. Consider the set of triangles, $N(t)$ (stands for neighbors), that are incident to t. The size of $N(t)$ is at most $4\pi/\theta$, where θ is the minimum angle of the mesh. Let $E(t)$ (stands for exteriors) be $\{t' | t' \cap B(x, f(x)) \neq \emptyset$ and $t' \notin N(t) \cup \{t\}\}$. It is sufficient to show that the size of $E(t)$ is bounded by a constant. Let $p_1 p_2 p_3$ be a triangle in $E(t)$ that contains a point $c \in C(x)$. We have $c_3 * f(p_i) \geq f(c) \geq 2 * ||x - c||$. The first inequality is from Lemma 3. Note that $Lnn_M(p_i) \geq \delta(p_i) * nn_M(p_i) \geq f(p_i)$. From Lemma 4, we have $||x - c|| \geq c_4 * min_{q_j}(||c - q_j||)$. Let q be the point with the minimum $nn_M(q_j)$, j=1,2,3. Hence, $||x - c|| \geq c_4 * nn_M(q)$. So $nn_M(p_i) \geq f(p_i)/L \geq f(c)/(L * c_3) \geq (2c_4/(L * c_3)) * nn_M(q)$. By a volumn argument, $f(x) \leq c_3 * f(q) \leq c_3 * L * nn_M(q)$ implies that the number of triangles in $E(t)$ is bounded by a constant. Lemma follows from the fact that the number of points in each triangle is bounded by a constant. ∎

We now analyze the time complexity of the algorithm. The time to compute the global spacing function f is $O(|M| \log(|M|))$. Notice that the mesh point p_j that defines $f(p)$ has the property that for all mesh point $p_k \in M$:

$$\delta(p_j) * nn_M(p_j) + ||p_j - p|| \leq \delta(p_k) * nn_M(p_k) + ||p_k - p||.$$

That is, p_i is contained in the additively weighted Voronoi cell of p_j. Fortune [3] shows how to apply the sweep-line technique to compute the additively weighted Voronoi diagram in $O(|M| \log(|M|))$ time.

The time complexity of step 1 of the algorithm FRFC is $O(|M| \log(|M|))$. During step 2, the number of the points sampled at any triangle element $q_1 q_2 q_3$ is bounded by $nn_M(q_i) * 2\sqrt{2}/(c_5 * f(q_i))$. Lemma 2 implies that the number of sampled points is at most $2\sqrt{2}/(c_5 * l)$. Hence, the time complexity of step 2 is also $O(|M| \log(|M|))$. Note that the time complexity is $O(|M|)$, if we use $(f(q) + ||x - q||)/2$ to approximate $f(x)$. The time to sort all the spheres during step 4 is $O(|M| \log(|M|))$. Because the maximum ply of any point in Ω with respect to $S_1 \cup S_2$ is bounded by a constant (Lemma 10), we can apply the sphere-separator based divide-and-conquer algorithm [6] to construct the conflict graph in $O(|M| \log |M|)$ time. In addition, we know that the conflict graph has at most $O(|M|)$ number of edges. Computing the maximal-independent-set of the conflict graph then takes $\Omega(|M|)$ time. The Delaunay triangulation takes $O(|M'| \log(|M'|))$ time, where $|M'|$ is linear in $|M|$, because the total number of the sampled points is linear in $|M|$. Therefore, the time complexity of FRFC is $O(|M| \log(|M|))$.

4 Size and Quality

We now show that FRFC returns a mesh M' that is well-shaped, and is of a size that is within a constant factor of the optimal possible. We will use the following structure theorem of Miller, Talmor, and Teng [7] which states an equivalent relationship between β-sphere packing and well-shaped meshes.

Theorem 2 (Sphere Packing and Well-Shaped Meshes) *1. For any positive constant β, there exists a constant α depending only on β such that if f is a spacing function of Lipschitz constant 1 over a domain Ω and S is a β-sphere packing with respect to f, then the Delaunay triangulation M of the centers of S is an α well-shaped mesh; in addition, for each point p in Ω, $nn_M(p) = \Theta(f(p))$, where the constant in Θ depends only on β.*

2. For any positive constant α, there exists a constant β depending only on α such that if M is an α well-shaped mesh, then the set of spheres

$$S = \{B(p, nn_M(p)/2) : \text{for all mesh point } p \in M\},$$

is a β-packing with respect to $nn_M/2$.

Therefore, there exists a constant α such that the mesh M returned by the algorithm is α-well-shaped. Note that the point set S returned by the algorithm is 7-packing with respect to $f/2$. The size optimality follows from the fact that f is a maximum spacing function that satisfies the condition of the Evolving Mesh Problem, (see Lemma 6), and the following lemma of [7].

Lemma 11 (Size of a Well-shaped Mesh) *If M is an α-well-shaped mesh of n elements, then*

$$n = \Theta(\int_{\Omega} \frac{dA}{nn_M^2}).$$

First, notice the number of the spheres in S is bounded by

$$\Theta(\int_{\Omega} \frac{dA}{f^2}),$$

by a simple volume argument. Because f is point-wise larger than the nn function of any well-shaped mesh that satisfies the Evolving Mesh Problem. It follows that size M is within a constant factor from the best possible. Therefore,

Theorem 3 (Main) FRFC *constructs a well-shaped mesh that satisfies the spacing condition given by δ. In addition, its size is optimal within a constant factor.*

The key to our algorithms in maintaining the well-shaped condition is to make sure that the shape condition does not deteriorate from M to M'. This is why we add new sampling points to regions near C-points to ensure the constant β in β-packing is maintained. Miller *et al* [7, 8] showed that in their coarsening algorithm that no new point is needed for coarsening. However, to do so, they need to use the original finest mesh directly to generate the coarsening mesh at each level. In other words, if the original mesh is M_0, then mesh point of M_0 are used to build the conflict graph to generate the mesh points for M_i. If they simply use mesh points of M_{i-1}, then mathematically, they can not guarantee the mesh points of M_{i-1} are dense enough for the well-shaped condition through the quality of the packing for M_i. For EMP, because of the mixed refinement, the original mesh does no longer provide fine enough sample points to guarantee the packing condition. Hence, new points has to be added. Our objective here is to add as small number of new points as possible, and meanwhile, by using as simple procedures as possible. In practice, for the moving boundary problem, there is no need to add new sample points to the back region of the moving boundary. We can use the algorithm of Miller *et al* [7, 8] to coarsen the back region.

5 Practical Variations

The δ values decompose the mesh M into a collection of components of maximal submeshes where the δ values of all mesh points in each submesh are either larger than 1 (type C-submeshes), or smaller than 1 (type R-submeshes). In practice the number of such submeshes is bounded by a small constant. For example, this number in most problems with a moving boundary is 2 (one for the front-end of the moving boundary, one for the back-end). As observed in Section 3, we need to insert Steiner points in the submeshes that are required for refinement. From submeshes to be coarsened, we often need to remove some original mesh points. A practical variation of our scheme is to first refine the R-submesh by any adaptive refinement algorithm, such as quad/octree refinement and Delaunay refinement. Then we apply the one-level coarsening algorithm of Miller, Talmor, and Teng [7]. We now present a detailed procedure for the case where Delaunay refinement is used. Recall that the standard Delaunay refinement procedure contains three rules [12, 13]:

1. splitting boundary subsegment whose diametral sphere contains a mesh point other than its end-points in its interior by adding a Steiner point at its midpoint;
2. splitting a boundary subfacet whose equatorial sphere contains a non-coplanar mesh point by adding a Steiner point at its circum-center. However, if the new point would cause any subsegment of the subfacets to split, apply rule one to these subsegments instead.
3. splitting any simplex that does not satisfies the well-shape condition by adding a Steiner point at its circum-center. However, if the addition of this circum-center would cause any subsegment or subfacet to split, then apply rules 1 and/or 2 instead.

We add a fourth rule, which states as: splitting any simplex in which the nn-spacing of any one of its mesh points is more than its delta-value times its nn-spacing by adding a Steiner point at its circum-center. However, if the addition of this circum-center would cause any subsegment or subfacet to split, then apply rules 1 and/or 2 instead.

Algorithm `Delaunay-Refining-Functional-Coarsening`
Input: A well-shaped mesh M and a list of positive reals δ.

1. Apply rules 1, 2, 3, 4 until all constraints on the spacing and shape at each mesh point are satisfied. Call the resulting mesh M_I.
2. Apply the one-level coarsening method of Miller, Talmor, and Teng [7] to M_I with coarsening factors given in δ to M_I to construct M'.

The following theorem follows directly from the main theorem of Ruppert [12] for 2D and of Shewchuk [13] for 3D and the coarsening result of Miller *et al* [7].

Theorem 4 `Delaunay-Refining-Function-Coarsening` *(DRFC) constructs a well-shaped mesh that satisfies the spacing condition given by* δ.

One of the shortcomings of DRFC is that it may construct a mesh that is larger than necessary. The reason is that in the refinement, we did not remove any original mesh point. In FRFC, we may replace some original mesh points in the R-submeshes by new Steiner points, which potentially reduce the mesh size. However, DRFC is in general more efficient.

When the lower bound l on δ is very small, the number of points introduced in each triangle could be very large, although it is a constant. Especially for the coarsening regions, (e.g., backend of a moving boundary), this is undesirable. In practice, we have a few alternatives:

- Do not add any points to triangles all of whose mesh points are C-points.
- Add only the barrycenter and/or midpoints of the edges rather than generating random quasi-uniform points based on the local grid.

Talmor [15] showed in her thesis that in practice no new point is needed for the region to be coarsened repeatly. We will conduct more experiments to verify this point in the context of EMP.

6 Conclusion

In this paper, we present a unified approach for coarsening and refining evolving meshes. One application and motivation of our work is for solving time-dependent problems with a moving boundary. In our future work, we will explore the structure of the moving boundary and level sets to speed up the coarsening and refinement procedure. We will also work on incorporate our algorithm into some standard mesh generation software. We will present some experimental results during the conference to show the effectiveness of our algorithm and its practical variations. In addition, all of the lemmas and theorems are applied to three dimensions if the aspect-ratio is used as shape criterion of the well-shaped mesh. However, this does not prohibit the existence of slivers.

In the context of parallel implementation of the Evolving Mesh Problem, the need of mesh evolution could introduce load imbalance among processors, where the load measures the amount of work required by solving the Evolving Mesh Problem itself as well as by numerical calculations thereafter. We need to develop a mesh distribution estimation algorithm to incorporate with the dynamic load balancing scheme developed in [4].

References

[1] I. Babuška and A.K. Aziz. On the angle condition in the finite element method. *SIAM J. Numer. Anal.*, 13(2):214–226, 1976.

[2] M. Bern, D. Eppstein and J. R. Gilbert. Provably good mesh generation. In *31st Annual Symposium on Foundations of Computer Science, IEEE*, 231–241, 1990.

[3] S. Fortune. A sweep line algorithm for Voronoi diagrams. *Algorithmica*, 2:153-174, 1987.

[4] X.-Y. Li and S.-H. Teng. Dynamic load balancing for parallel adaptive mesh refinement. In *5th International Symposium on Solving Irregularly Structured Problems in Parallel*, Berkeley, 144-155, 1998.

[5] G. L. Miller, D. Talmor, S.-H. Teng. Data Generation for Geometric Algorithms on Non-Uniform Distributions. *International Journal of Computational Geometry and Applications*, accepted and to appear, 1998.

[6] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, IMA Volumes in Mathematics and its Applications. Springer-Verlag, pp57–84, 1993.

[7] G. L. Miller, D. Talmor, S.-H. Teng. Optimal coarsening of unstructured meshes. In *Journal of Algorithms*, invited and accepted to a special issue for SODA 97.

[8] G. L. Miller, D. Talmor, S.-H. Teng. Optimal Good Aspect Ratio Coarsening for Unstructured Meshes. In *8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp 538–547, January 1997. ACM-SIAM.

[9] G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 683–692, 1995.

[10] G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. On the radius–edge condition in the control volume method. *SIAM J. on Numerical Analysis*, accepted and to appear, 1998.

[11] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in three dimensions. Proc. ACM Symposium on Computational Geometry, pp 212-221, 1992.

[12] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Third Annual ACM-SIAM Symposium on Discrete Algorithms*, 83–92, 1992.

[13] J. R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *14th Annual ACM Symposium on Computational Geometry*, 86–95, 1998.

[14] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method*, Prentice-Hall, 1973.

[15] D. Talmor. Well-Spaced Points for Numerical Methods. Ph.D thesis, Carnegie Mellon, 1997.

Fast Adaptive Quadtree Mesh Generation

Pascal J. FREY and Loïc MARECHAL
INRIA, Gamma Project,
Domaine de Voluceau, Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France.

Abstract. *A size-governed quadtree mesh generation method is presented in this paper to deal with planar domains of arbitrary shape. The tree decomposition provides a convenient control space, which can be used to determine the element sizes, as well as a neighboring space, which allows for the quick searching of mesh items. The sizes of the tree cells are adjusted to match the size specifications (defined as a continuous element-size distribution function in $\mathbb{R}^2$). Hence, the proposed method can be used in the context of mesh adaption in numerical simulations based on the finite element method. Several application examples are provided to emphasize the main features of this approach.*

Keywords. Quadtree, Spatial decomposition, Mesh generation, Mesh adaption.

Introduction

The application of spatial decomposition methods (so-called because they combine quadtree decomposition techniques with quadrant-level meshing procedures) to finite element mesh generation have been pionneered about two decades ago. A survey of the literature devoted to spatial decomposition methods for mesh generation purposes can be found in [Thacker-1980] and [Shephard-1988]. Some of these approaches have yet proved to be robust and reliable and are stilll commonly used in a wide range of engineering applications and even in some commercial packages. One of the main feature of such an approach is its ability to deal with a domain represented by a boundary discretization or to interact directly with a CAD modelling system (to generate both the boundary discretization and the mesh of the domain). In some sense, this type of method have contributed to the avent of generic mesh generation techniques (capable of handling arbitrarily shaped domains), in the same way as advancing-front or Delaunay-based methods [George-1991].

The classical approach consists of two main stages, the domain $\Omega \in \mathbb{R}^2$ is first recursively decomposed into a set of variably sized disjoint cells (*i.e.*, the *quadrants*) representing a partition of a bounding box $\mathcal{B}(\Omega)$ of Ω (cf. Figure 1.(ii)). A second stage consists in subdividing each terminal cell into finite elements (triangles and/or quadrilaterals) so as to complete the mesh of the domain. The elements esternal to the domain are then discarded from the resulting mesh. Usually an optimization stage based on node smoothing and mesh modifications (by means of geometric or topological operations) is used to improve the shape quality of the mesh elements.

Related work. The use of a quadtree decomposition for meshing purposes was pioneered fifteen years ago by [Yerry,Shephard-1983] and several variants have been proposed (see for instance [Kela *et al.* 1986] or [Perucchio *et al.* 1989]). The element creation stage commonly involves pre-defined patterns (the so-called *templates*) to meet the requirements of conformity (between adjacent quadrants) as well as of efficiency. On the other hand, during the spatial decomposition stage, a filtering operation can be introduced to control the element shape quality and to avoid creating badly-shaped elements. Recent papers have investigated this *a priori* evaluation of the mesh element quality, an upper bound for the angles being eventually exhibited [Mitchell,Vavasis-1992].

To our knowledge, in most of the publications related to quadtree mesh generation, the hierarchical structure is mainly used as a *neighboring space* (used to locate the cells adjacent to a given cell). However, the possibility of using the tree as a *control space* (used to prescribe the desired elements sizes) has not been clearly devised so far (except perhaps for specific Euler computations in CFD [Shephard *et al.* 1988]). If size specifications are supplied, the tree decomposition can be adjusted so that the cell sizes locally match the desired element sizes [1]. In the context of mesh adaption (in which the size specification is usually provided by an error estimate), this feature allows to increase the convergence of the computational scheme and the accuracy of the numerical results [Ciarlet-1978].

[1] as the length of the mesh edges are closely related to the cell sizes.

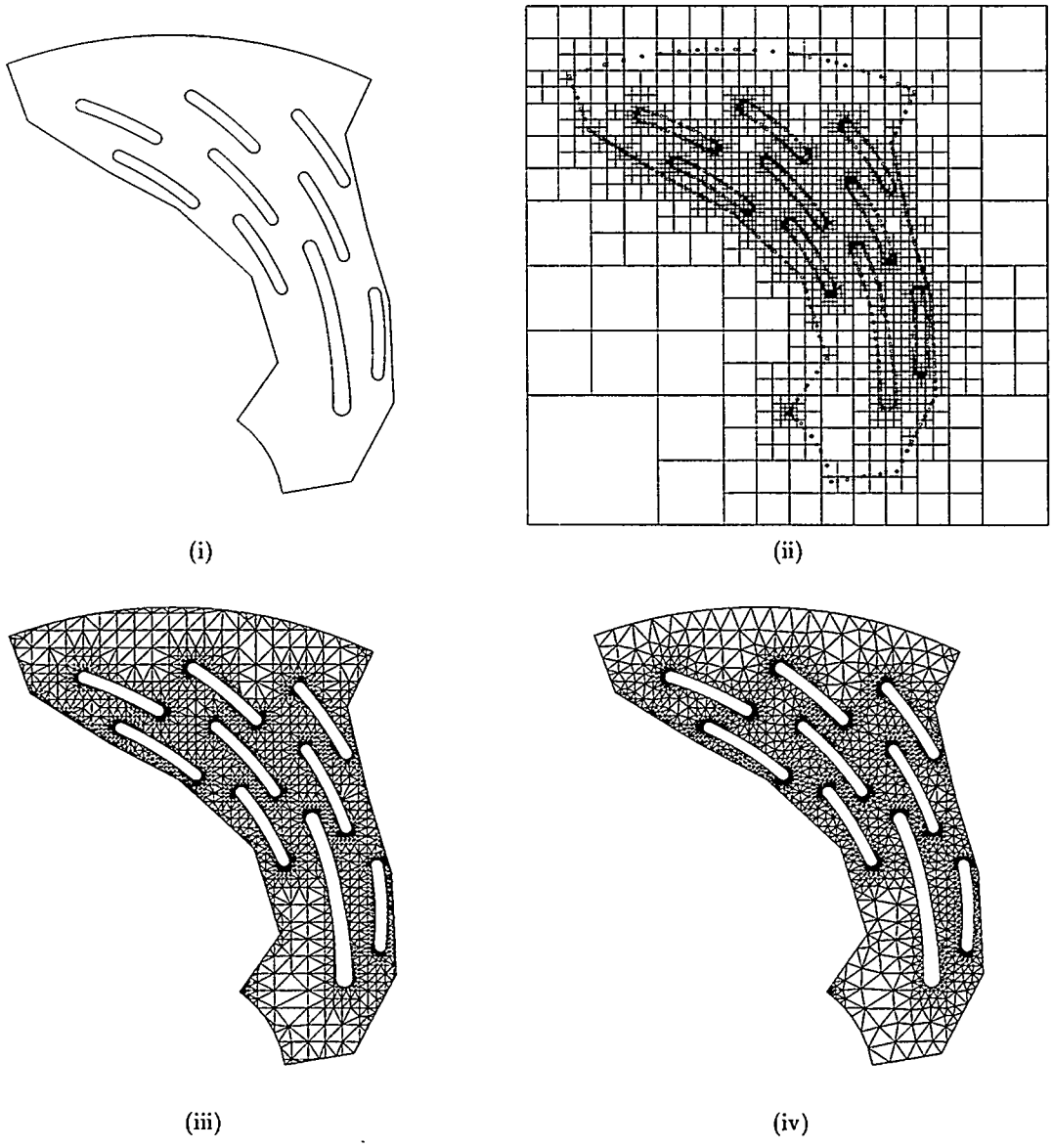


Figure 1: Two dimensional domain Ω, a rotor : domain boundaries (polygonal contour $\Gamma(\Omega)$) (i), spatial decomposition of a bounding box $\mathcal{B}(\Omega)$ (ii), resulting *raw* mesh before optimization (no filtering) (iii) and final mesh of Ω after optimization (iv).

Scope. In this paper, we would like to discuss the tree decomposition stage and propose some improvements of the classical scheme to cope with a given size map. The motivation of this work has been the will to design a robust and efficient algorithm capable of handling two-dimensional domains of arbitrary shape (e.g. non-manifold models as well as domains with features of widely different scales).

A common idea relates the efficiency of the tree decomposition meshing algorithm to the data structures used to represent the tree and to store the mesh. Although these structures contribute obviously to the overall performance of the meshing algorithm (for instance by allowing to reduce the number of tree searching and traversal operations), the most time consuming operation is the filtering step. This treatment attempts to remove closely spaced entities in a terminal quadrant and therefore involves extensive topological checks to maintain the topological consistency of the decomposition. Thus, for efficiency purposes, the proposed approach does not include a filtering stage, which is then replaced by an adequate mesh optimization procedure.

Outline. This paper is divided into five sections. Section 1 recalls the terminology used with quadtree decompositions and summarizes the general scheme of the classical quadtree mesh generation. Section 2 proposes a governed quadtree mesh generation algorithm and eventually discusses the boundary discretization problem. Section 3 presents the extension of the general scheme of the size-governed quadtree algorithm in the context of mesh adaption. Several application examples are proposed Section 4 to emphasize the main features of the proposed approach. A brief section concludes the paper by mentioning the possible extensions of this work.

1 Quadtree decomposition

In this section, we recall some terminology and the basic definitions of the spatial decomposition structures in two dimensions [Samet-1984]. The input of the problem is an arbitrarily shaped domain $\Omega \in I\!R^2$, represented by a boundary representation (*i.e.*, a polygonal contour $\Gamma(\Omega)$ described by a set of vertices $\mathcal{V}$ and a list of edges $\mathcal{E}$).

Some terminology. The basic concept of any spatial representation consists of enclosing the domain Ω into a bounding box (usually a square), denoted $\mathcal{B}(\Omega)$, corresponding to the *root* of the spatial decomposition tree. This box is subdivided into four equally-sized *cells* (the *size* of a cell c is the length of a side of c), each of which being recursively subdivided several times. The *stopping criterion* used to subdivide a cell can be based on the local geometry of the domain (e.g., the local curvature of the boundary) or user-defined (e.g., the maximum level of refinement).

The four vertices at the corners of a cell are called *corners*. The edges connecting each consecutive corners are the *sides* of the cell. The edges of the decomposition that belong to the boundary of the cell are called the *edges* of the cell. Hence, each side of a cell contains at least one edge. Two cells are *adjacent* if they share an edge. Neighbors are identified by the four cardinal directions, $\{N, S, E, W\}$ and the quadrants obtained by subdividing a cell are identified by their relative position within the parent cell : $\{NW, NE, SW, SE\}$. The *level* of a cell corresponds to its depth in the related tree, *i.e.*, the number of subdivisions required to obtain the cell. The bounding box is at level 0. The *depth* of the tree corresponds the maximum level of subdivision. Any cell that is not subdivided is a *terminal* cell or a *leaf*.

At each stage, any cell can be subdivided into four sub-cells or not. Hence, the resulting decomposition tree may be quite unbalanced (the number of subdivision levels being not constant in each cell). This remark leads to introduce the following rule[2] : any side of a terminal cell must contain at most one corner[3] This condition is known as the [2:1] *rule* that was first suggested by [Yerry,Shephard-1983].

Operations on quadtrees. Two types of operations are frequently applied with trees : topological operations (e.g. quadrant creation, finding the cells adjacent to a given cell in a given direction) and geometric operations (e.g. finding the cell that contains a given point, finding the intersections between an edge and the current cell). These operations have not all the same frequency and not the same computational cost. The *neighbor finding* operation concerns the search of the cells adjacent to a given cell and is frequently called during the quadtree decomposition, for instance by a post-processing algorithm used to enforce the [2:1] rule.

[2] which is also used to control the mesh gradation (*i.e.*, the size variation between neighboring elements.)
[3] This condition is equivalent of writing that the sizes of any two adjacent cells differ by at most a factor two.

Data structures. Three approaches are possible to represent trees [Knuth-1975] :

- A tree structure using pointers. In this obvious approach, each internal cell requires four pointers, one for each of the subtrees and a bit of information to indicate the cell classification (internal or terminal). In addition, extra pointers can be added to improve the efficiency, such as links to parent cell, links to neighboring cells, ...
- A list of the nodes encountered by a traversal of the structure. With this implementation, intersection algorithms can be efficiently performed, although other algorithms may be less efficient. For instance, visiting the second subtree of a cell requires to visit each node of the initial tree to locate the root of the subtree.
- A system of locational codes (e.g. binary encoding), for instance with a *linear quadtree.*

The information requirements almost dictate the choice of a representation : for instance the need to quickly identify neighbors leads to favor a linear structure or a pointer-based structure containing pointers to the parent cell and to the adjacent cells. We have retained this last representation as most of the treatments are localized and involve mainly the adjacent cells of a given cell (for instance the balance condition).

General scheme. The aim of the quadtree-based mesh generation approach is to obtain a valid and accurate representation of a planar domain by a set of triangles suitable for finite element analysis. More specifically, the domain is decomposed into a set of cells that have a size distribution compatible with the desired mesh gradation and store all the information required by the meshing step to generate a finite element mesh of the domain. Several features of the method are common to all quadtree-based meshing techniques :

- the data structure is used for localization and searching purposes,
- the mesh generation is twofold, first the tree is generated, then the mesh is created,
- the cell entities (corners, edges) are mesh entities (vertices, edges),
- the mesh gradation is controlled by the level of refinement of the cells (for instance, using the [2:1] rule).

Schematically, the classical quadtree-based meshing approach consists of three successive steps and can be summarized as follows :

1. the *quadtree construction* : decomposition (insertion of the entities of $\mathcal{V}$ and $\mathcal{E}$, balance enforcement ([2:1] rule), filtering and warping of the quadrant entities,
2. the *mesh generation* : point and element creation (based on templates),
3. the *mesh optimization* : node smoothing, topological and geometric mesh modifications.

The next section explains how this approach differs from this general scheme to account for a specified size map.

2 Governed quadtree mesh generation

Let consider a planar domain $\Omega \in \mathbb{R}^2$ represented by a polygonal contour $\Gamma(\Omega)$ described by a list $\mathcal{V}$ of points and a list $\mathcal{E}$ of edges (*curved sections* in the boundary description are allowed). In addition to this input data, a size map can be supplied to prescribe the sizes of the mesh elements. In this section, we will explain how the size specifications can be taken into account during the tree construction stage.

2.1 Control space

To govern the internal point and element creation, it is convenient to use a control space. Formally speaking, this control space is defined as follows [George-1991]:

Definition 2.1 *(Δ, H) is a control space associated with a mesh $\mathcal{T}$ of a domain Ω if*

- *$\Omega \subseteq \Delta$ where Δ is a covering up of Ω,*
- *$\forall P \in \Delta, \quad \exists H(P, \vec{d})$, where $\vec{d}$ is a direction of the disk S^1 : $H(P, \vec{d}) : \Delta \times S^1 \longrightarrow \mathbb{R}$.*

From the geometric point of view, it is obviously convenient to consider the tree decomposition as the desired covering up Δ. To construct the function H, we will use a Q^1 interpolation, from the sizes h_{P_i} associated with the vertices of the tree cells. If no size map is provided, an *intrinsic* size map is computed. A discrete sizing function is obtained from the tree decomposition, at each quadrant vertex P the size h_P can be defined by averaging the Euclidean lengths of the cell sides incident to this vertex. Using a generalized interpolation scheme, a continuous size function H can be defined at any point of the computational domain Ω. Note that this sizing function accounts for the mesh gradation as controlled by the [2:1] rule.

On the other hand, if a size function is supplied, it can be used to govern the tree decomposition. The element sizes are commonly specified by :

- parameters associated with domain entities,
- values associated with the vertices of the previous mesh (in an adaption scheme),
- parameters related to the domain geometry (e.g. the local curvature).

As the elements created in a quadrant almost span the cell, the length of a mesh edge is then approximately equal to the cell size containing it. Let consider a point P in a tree cell c. If the desired size h'_P differs from the intrinsic size h_P at P (based on the current decomposition) and is such that $\frac{h_P}{h'_P} > \sqrt{2}$, the cell is refined into four sub cells and the new cell containing P is recursively analyzed. At the same time the cell is divided into four sub-cells, the intrinsic sizes are updated at the tree vertices and the tree is balanced. At completion of this procedure, the intrinsic size function and the specified size function are close (the ratio between h_P and h'_P is bounded by the value $\sqrt{2}$).

2.2 Boundary discretization

The quadtree method can either generate the boundary representation of the domain along with the domain covering-up or start fom a given polygonal representation and create internal elements only. The first approach has been commonly adopted by several authors (the mesh generation algorithm being interfaced with a geometric modelling system [Grice *et al.* 1988], [Shephard-1988]). In the proposed approach, the boundary discretization is clearly disconnected from the tree decomposition stage. This choice is performed to disconnect the quadtree approach from any modelling system and to allow a better control on the geometric approximation of the domain boundaries.

Curvature-based refinement (*i.e.*, finer meshes in highly curved regions and coarser meshes in regions of low curvature) provides a convenient means of controlling the geometric approximation of the mesh elements. Basically, the goal is to obtain a polygonal approximation of the curve $\Gamma(\Omega)$ that deviates from the underlying true geometry by no more than a (user) given tolerance value ϵ.

Geometric support construction. The set of polygonal segments of $\Gamma(\Omega)$ is a best approximation of the underlying continuous curve Γ and thus represents a geometric support of the curve. The construction of the polygonal support is such that the distance between any segment and the portion of the curve it represents must be bounded. Hence, refinement will be more important in highly curved regions.

Let h be the distance between a point and the supporting edge, let d be the length of the edge, we like to have $h/d < \epsilon$, ϵ being the desired tolerance. If the tolerance is not exceeded, the segment is correct. Otherwise, the segment is subdivided into two sub-segments, each of them being recursively analyzed. The resulting set of segments represents the *geometric support* of the curve (cf. Figure 2, right).

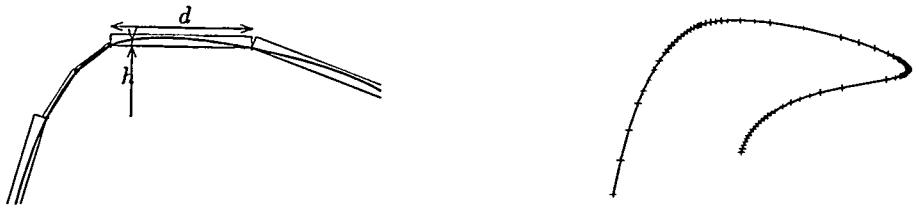


Figure 2: Approximation using a polygonal segment with respect to a user-specified tolerance value $\epsilon = 0.08$ (left), $\epsilon = 0.01$ (right).

Geometric approximation. Curvature-based mesh generation requires the ability to evaluate the local curvature of a curve or a surface[4]. In our approach, the geometric support is constructed using a third order polynomial approximation (Hermite function) from the discrete tangents. Given a polygonal segment P_iP_{i+1}, the direction of the tangent τ_{P_i} at P_i is parallel to the line $P_{i-1}P_{i+1}$, its module is given by $\|\overrightarrow{P_{i-1}P_i}\|$. The desired element size h_P at P must then be proportional to the radius of curvature r_P at P, $h_P = \alpha r_P$. The control of the geometric approximation consists in defining α so that, for a given deviation value ε,we have : $\delta/r_P \leq \varepsilon$ (the osculating circle of radius r_P being a second order approximate of the curve at P). Having two parameters ε and α, for the desired size, their relation is : $\alpha \leq 2\sqrt{\varepsilon(2-\varepsilon)}$ [Frey,Borouchaki-1998].

Boundary discretization. The boundary discretization aims at defining a polygonal approximation of the boundaries curves that conforms to a given tolerance value. To this end, once the geometric support is constructed, the initial polygonal discretization is analyzed, based on the edge lengths computation. The size map proportional to the radii of curvature provides a size function that will enable us to compute the length ℓ_{AB} of any segment AB as :

$$\ell_{AB} = (s_B - s_A) \int_0^1 \frac{1}{H(s)} dS, \tag{1}$$

where s_A, s_B are the curvilinear abcissa and $H(S)$ represents an interpolation function of the size function $h(s)$ along the segment $[s_A, s_B]$, $S = (s - s_A)/(s_B - s_A) \in [0, 1]$. The problem is then to split the segment AB into n equally sized unit sub-segments (*i.e.* having a length close to 1) with respect to the given size map [Laug *et al.* 1996]. The resulting polygonal segment will serve as input for the quadtree decomposition.

2.3 Quadtree decomposition

The objective of the tree decomposition stage is to relate the desired elements sizes to the depth of the tree. This can be achieved by mesh parameters specifications. The quadtree representation of a domain Ω is defined from a square (the bounding box $\mathcal{B}(\Omega)$) that fully encloses the points of $\mathcal{V}$. This box is then subdivided into four quadrants, each of them being then tested recursively to decide whether it needs further refinement. The process stops when each quadrant contains less than two points and when the domain is resolved to a satisfactory resolution.

Quadtree construction. The depth p of the tree can be related to the length h of a mesh edge and the size b of the bounding box $\mathcal{B}(\Omega)$ as : $p = \log_2(b/h)$. As the size h is proportional to the minimal of the principal radii of curvature (in the intrinsic size map), we obtain a lower bound for the depth of the tree : $p \geq \log_2(b/(r\alpha))$, where α is the coefficient introduced before. This result can be used to define a stopping criterion in the tree decomposition stage.

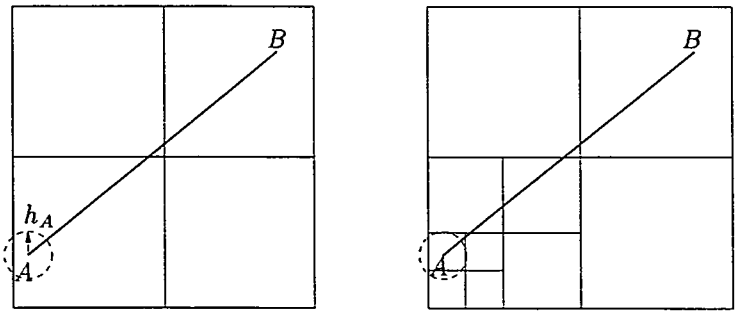


Figure 3: *Tree decomposition using a prescribed size. The size at point A forces the initial tree box containing A (left hand-side) constructed without explicit sizing prescription to be decomposed two levels deeper (right hand-side) to adjust to the size h_A represented by the dashed circle.*

Stopping criteria. Since the domain is described as a list of points and a list of edges, the stopping criterion must account for both types of entities : each leaf contains at most one connected component of $\Gamma(\Omega)$. If a leaf contains no point of $\mathcal{V}$, then it contains at most one edge or portion of an edge of $\mathcal{E}$. An additional criterion is introduced, each quadrant side must contain at most one boundary intersection point, except if the quadrant contains a point of

[4]This measure can be easily computed from the derivative information or returned by a modelling system [doCarmo-1976]

$\mathcal{V}$. Notice that special attention must then be paid to corners (vertices where two edges form an accute angle) and non-manifold points (points having more than two incident edges). The cell containing such points is not subdivided, thus violating deliberately the [2:1] rule. This will prevent the tree from degenerating when trying to separate closely spaced edges sharing a common vertex.

The tree construction consists of introducing successively all entities of $\mathcal{V}$ and all entities of $\mathcal{E}$ (cf. Figure 4). Notice that the tree decomposition may change dramatically the initial discretization as it always attempt to separate closely spaced entities that belong to different connected components (cf. Figure 9, enlargement).

Remark 2.1 *To avoid numerical roundoff problems, the vertices of $\mathcal{V}$ that coincide with a quadrant corner or lie on a quadrant side have their coordinates slightly changed. The original coordinates will be restored after the* raw *is created.*

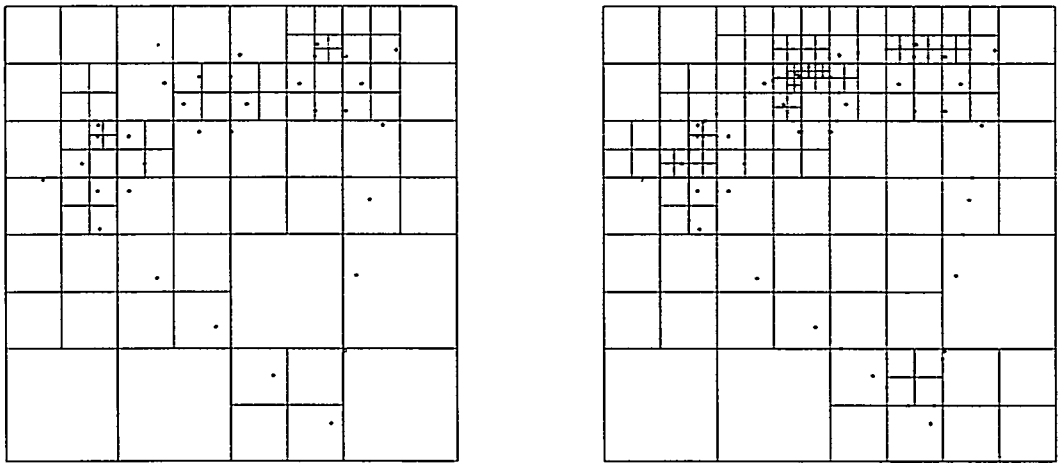


Figure 4: *Quadtree decomposition : after the insertion of the points of $\mathcal{V}$ (left) and after the insertion of the segments of $\mathcal{E}$ (right).*

Balance condition. Partly to control the mesh gradation and also to simplify the element creation step (*i.e.*, to reduce the number of templates), the levels of adjacent quadrants are checked to see if two neighboring cells differ by more than a factor two in size (according to the [2:1] rule). This procedure results in more quadrant splits to propagate accross the structure, thus increasing the total number of terminal cells. The procedure is carried out during the construction stage (without first constructing the unbalanced quadtree). At the same time a cell is subdivided, the sizes associated with the mesh vertices are updated to preserve the underlying continuous size map.

Remark 2.2 *As no check on the edge-quadrant intersections is performed, newly created vertices may be close to quadrant entities (corners or sides), thus leading to poorly-shaped elements. However, the extensive amount of topological checks required to preserve the integrity and the conformity of the domain does not seem justified, especially as a clever mesh optimzation is able to remove this undesirable elements.*

2.4 Mesh generation

The sequence of operations has concerned so far the quadtree structure. The topology of the final mesh is constructed based on the quadrant classification : all terminal cells are visited by a tree traversal procedure and the cells are triangulated accordingly. The final mesh is the union of all the quadrants triangulations. The balance procedure leads to a substantial reduction of the number of possible transition cases and allows the introduction of predefined basic patterns (the so-called *templates*). Hence, the generation of triangles in quadrants is really straightforward.

Actually, an internal quadrant can have any combination of the four neighboring quadrants, thus corresponding to $2^4 = 16$ possible triangulations (that can be reduced to only six templates using the various symmetry properties). Practically, a four bit index is created to serve as a pointer into a table that gives all quadrant triangles corresponding to a given quadrant configuration.

However, if a box contains a vertex of the boundary discretization, the treatment is slightly different. This point is star-shaped with respect to the other quadrant entities (corners or additional intersection points along the sides). Thus, this point is simply connected to all quadrants vertices so as to create a triangulation of the quadrant. On the other hand, if the quadrant contains intersection points, an element edge must connect two intersection points (or two corners) so as to result in a valid triangulation of the portion of the domain enclosed between the boundary $\Gamma(\Omega)$ and the boundary of the internal mesh (cf. Figure 5).

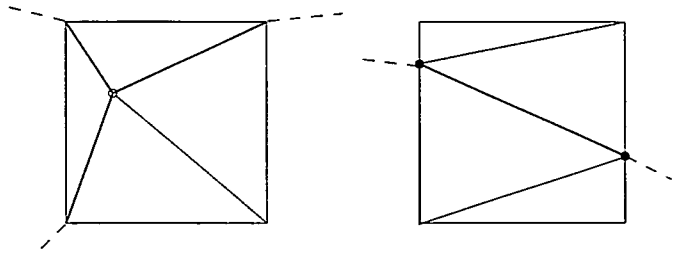


Figure 5: *Triangulation of the boundary quadrant : for a vertex of the boundary discretization (left) and for intersection points (right).*

Remark 2.3 *This meshing technique is intrinsically very simple to implement. However, one has to notice that the resulting mesh is a mesh of the bounding box $\mathcal{B}(\Omega)$ rather than a mesh of Ω only. In other words, many elements external to the domain have been created that need to be removed at this stage. Therefore, a simple coloring scheme is applied to identify the possible sub-domains and to remove the external triangles. This procedure retrieve the different connected components of the domain using adjacency relationships between the triangles.*

2.5 Mesh optimization

As no filtering stage is applied on the terminal quadrants, a (small) number of poorly-shaped elements can be created in the *raw* mesh. The first step in mesh optimization consists in removing these elements using a specific procedure. A basic observation shows that two types of porrly-shaped elements can occur, depending on whether a vertex is too close from a quadrant corner (Figure 6, left) or from a quadrant side (Figure 6, right). Any combination of these elements can potentially be created in the boundary quadrants (cf. Figure 7).

Figure 6: *Poorly-shaped element due to quadrant-model interactions, a* needle *(left) and a flat element (right).*

During the tree decomposition stage, the points that may cause the creation of bad elements are tagged as well as the elements having these points as vertices. Then an edge collapsing technique is iteratively applied to remove the badly-shaped elements, in combination with edge swapping operations to locally improve the mesh quality. The identification of the edges too small is made possible by use of the normalized edge length with respect to the current size map. At completion of the process, no unacceptable element remains left in the mesh.

Remark 2.4 *The use of the normalized edge lengths preserve the small features in the mesh, for instance in highly curved regions.*

The next step consists of pulling the newly created nodes on the boundary using the geometric support. As the geometric approximation has been controlled, this operation does not result in invalid or overlapping elements.

The triangulation of internal quadrants leads to well-shaped elements, for which the aspect-ratio is bounded *a priori*. However, the aspect ratio of triangles created in boundary quadrants can be significantly degraded as the triangulation of such quadrant is arbitrary and is, for instance, related to the relative position of the point of $\mathcal{V}$ contained in this quadrant with respect to the quadrant sides. Therefore, the resulting mesh requires some amount of mesh quality improvement.

Aspect ratio. The aspect ratio of a triangle K is defined as :

$$Q_K = \alpha \frac{h_{max}}{\rho_K} = \alpha \frac{h_{max} p_K}{S_K}, \tag{2}$$

where h_{max} is the *diameter* of K, (*i.e.*, its longest edge), ρ_K is the in-radius of K, p_K is the half-perimeter of K and S_K is the surface of K. The coefficient $\alpha = \sqrt{3}/6$ is a normalization coefficient such that the aspect ratio value of a regular triangle is equal to 1. The aspect ratio measures the degradation of the element shape quality and is a value ranging from 1 to ∞.

The objective of the mesh optimization procedure is to improve the overall shape quality of the mesh (*i.e.*, to skew the histogramm of shape quality towards the left). In addition of improving the shape quality, this procedure also accounts for the size quality of the mesh. The size quality is measured by means of the normalized lengths of the element edges. The optimal size quality is 1.

The tools used to perform mesh optimization are twofold, geometric mesh modifications (node relocation) and topological modifications (edge collapsing, edge swapping).

Remark 2.5 *As the quadtree decomposition provides a good distribution of internal mesh vertices, there is non need to introduce new nodes by means of edge splitting.*

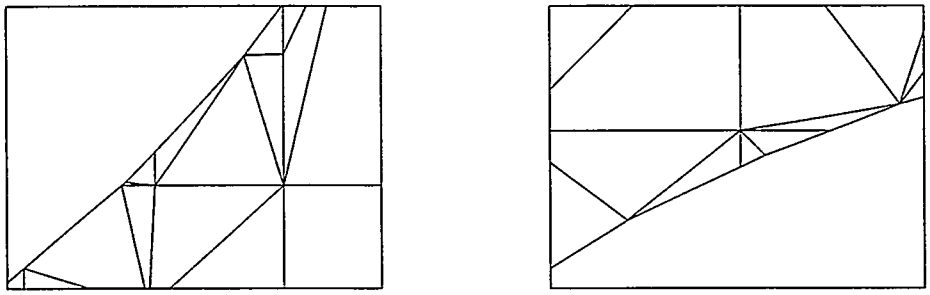


Figure 7: *Curved domain boundary, different combinations of poorly-shaped elements in the vicinity of the boundary.*

These operations are performed sequentially provided the element shape quality improves. The point relocation procedure is based on a weighted barycentrage technique that accounts for optimal edge lengths, the computation of the optimal points is based on the unit length. In the isotropic case, a simple Laplacian smoothing [Freitag-1997] or a Lagrangian relaxation procedure [George-Borouchaki-1997] can also be applied. Figure 8 shows the result of mesh optimization on a computational CFD domain.

3 Mesh adaption

The main purpose of a computational adaption scheme is to ensure the reliability of the finite element solutions. In this respect, a complete adaptive scheme usually includes a solver, an error estimate and a mesh adaption procedure. The result of the error estimation is translated in terms of sizes specifications. This sizing requirement is used to obtain a new tree decomposition and consequently a new mesh that conforms better to the desired sizes than the previous decomposition. The general scheme can be summarized as follows :

1. construction of an initial mesh $\mathcal{T}_{i=0}$,
2. computation of the solution on the current mesh,
3. solution analysis using an *a posteriori* error estimate,
   - if the solution is converged, end.
   - else
     - construction of a discrete size map,
     - construction of a mesh $\mathcal{T}_{i+1}$ governed by the above size map,
     - back in step 2.

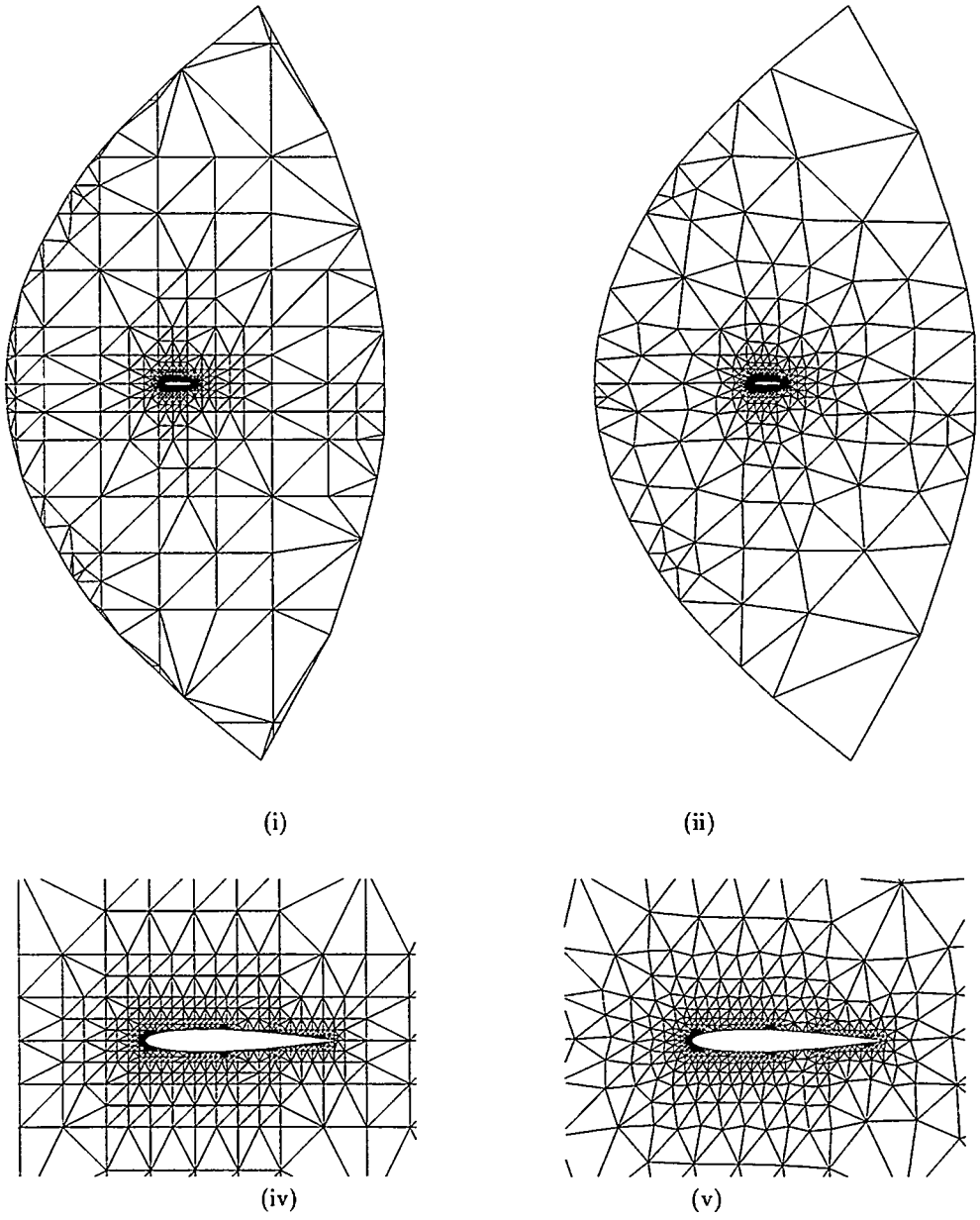


Figure 8: *Computational CFD mesh around a Naca012 airfoil : initial quadtree mesh (i) vs. final optimized mesh (ii). Local enlargments around the airfoil, original mesh (iii) and optimized mesh (iv).*

The size specifications are associated with the vertices of the current mesh and define a so-called *discrete* size map. This discrete map can be used to define a control space (cf. Section 2.1) and to govern the tree decomposition procedure.

In the adaptive context, the tree is constructed based on the boundary discretization of the domain and the specified sizing function. Usually, the current mesh is used as the covering-up Δ in the control space and the discrete size map is the function H. In our approach, there is no need of using the current mesh as control space as the tree can be used to serve this purpose. As explained previously, the tree is refined according to the desired size specified at the boundary entities. Then, the insertion of each vertex of the current mesh is simulated to see whether or not the cell containing it has the desired size. More precisely, the desired size h' at the vertex is compared with the size h obtained by a bilinear interpolation in the cell. If the ratio $h/h' > \sqrt{2}$ then the cell is refined. It is then straightforward to create a size conforming quadtree decomposition.

The element creation is exactly the same as that of the classical scheme. The mesh optimization procedure remains also unchanged as care has been taken to preserve the size of the element during the mesh modifications. The use of normalized edge lengths with respect to the metric map prevent the small elements to be removed. The aim of the optimization stages is to provide a unit mesh (*i.e.*, a mesh having all its edges of unit length w/r size map). In this context, the node smoothing procedure (moving a vertex P) is equivalent of finding an *optimal* point O_j such that each edge P_jP incident to P is of unit length :

$$O_j = P_j + \frac{\overrightarrow{P_jP}}{\ell_{\mathcal{M}}(P_j, P)}, \tag{3}$$

where $\ell_{P_j,P}$ is the length of edge P_jP with respect to the size variation function associated with the edge.

A length efficiency index is used to measure the mesh quality with respect to the metric map. Let ℓ_{AB} be the length of the edge AB with respect to the given size map. The *efficiency index*, denoted τ, of a mesh $\mathcal{T}$ can be defined as the average value of the squares of the differences to 1 of all mesh edge lengths (let na be the number of mesh edges), hence :

$$\tau = 1 - \frac{1}{na}\sum_{i=1}^{na} e_{AB}^2, \tag{4}$$

with $e_{AB} = 1 - \ell_{AB}$ if $\ell_{AB} < 1$ or $e_{AB} = 1 - \ell_{AB}^{-1}$ if $\ell_{AB} > 1$. This coefficient seems adequate to quickly estimate the mesh quality with respect to a given metric map. In particular, it indicates that the edge lengths are l times too long or too short (a value $\ell = 5$ or $\ell = 0.2$ means that all edges are 5 times too long or 5 times too short) as compared with the desired sizes. The optimal value being $\ell = 1$, any value $\ell > 0.91$ ensures a reasonable mesh quality with respect to the given metric map.

4 Application examples

Several application examples are provided to illustrate the main features of the proposed algorithm. These examples concern realistic objects as frequently encountered in numerical simulations using the finite element method. Table 1 reports statistics for the quadtree meshes. In this table, N_p, N_e represent the number of vertices and elements, Q_{min} and Q_{avg} are the worst and the average shape quality values of $\mathcal{T}$, ℓ_{min}, ℓ_{max} and ℓ_{avg}, e_i correspond to the minimum, maximum, average edge lengths and efficiency index values. The mesh elements generated using this method are

mesh	N_p	N_e	Q_{min}	Q_{avg}	ℓ_{min}	ℓ_{max}	ℓ_{avg}	τ	Figure
naca012	683	1,218	3.2	1.28	0.36	1.6	0.74	0.91	8
multinaca	1,434	2,550	2.6	1.29	0.24	3.12	0.75	0.92	10
cooler	2,204	3,794	2.9	1.29	0.25	1.92	0.75	0.91	9
rotor8	2,319	3,864	3.18	1.3	0.29	2.32	0.76	0.92	1
fourche	49,771	86,092	4.0	1.25	0.28	2.4	0.75	0.91	

Table 1: *Statistics relative to the numerical evaluation of the quadtree meshes.*

globally well-shaped and conform to the desired sizes. However, the average edge length is closer to $\sqrt{2}$ than to the unit length (this is most likely due to the discrete [2:1] transition introduced during the tree decomposition). The efficiency index confirms that the elements sizes are compatible with the desired sizes. So far, the results confirm the efficiency of the algorithm, between 1.5 and 2.5 millions elements per minute are generated on a HP 9000 PA8200,

200Mhz workstation. This cpu time correspond to the tree decomposition and the generation of the final mesh, neglecting the external elements[5].

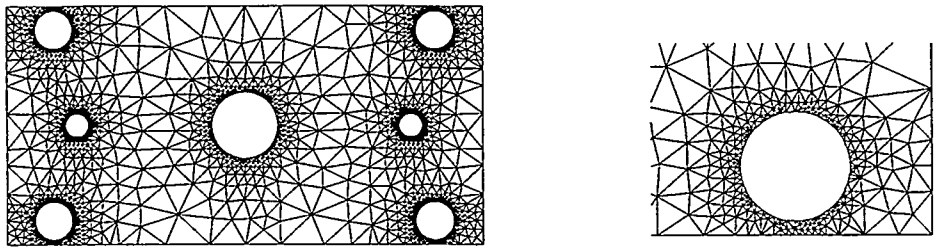


Figure 9: *Final optimized mesh of 'cooler' (left) and local enlargement (right).*

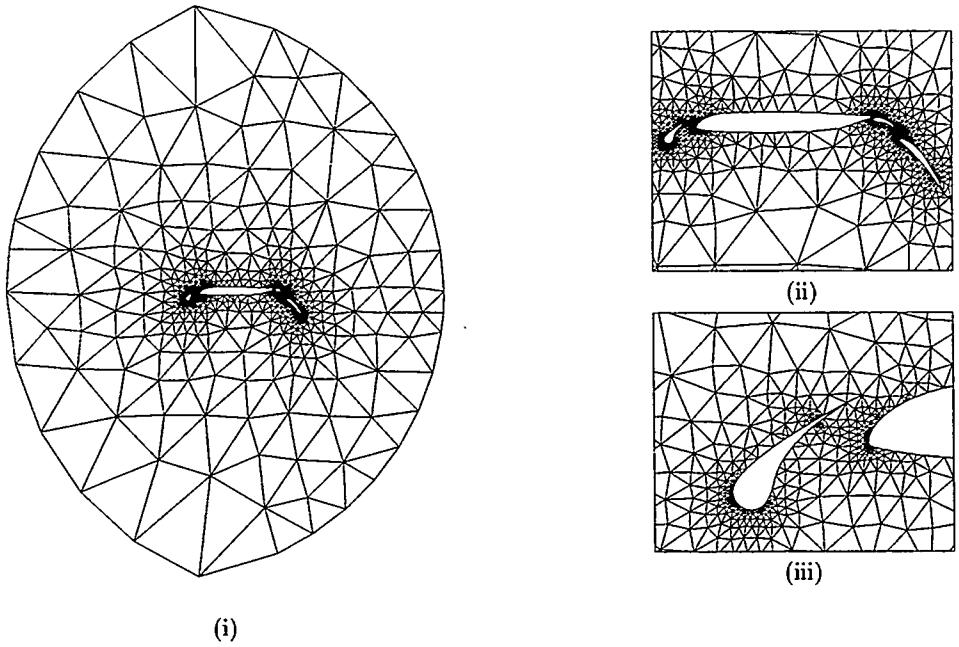


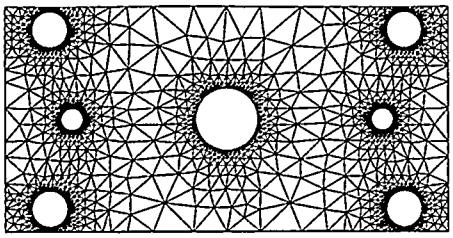
Figure 10: *Computational CFD mesh around a multi airfoil (i) and local enlargments around the airfoil (ii) and (iii).*

In the next example, the size map has been analytically defined. Two metric fields are specified inside the computational domain. The size functions have been defined at any point $P(x,y)$ as follows :

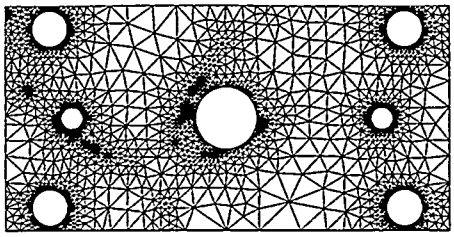
$$h1(x,y) = 4|r_1 - \|\overrightarrow{C_1P}\|| + 0.05 \quad \text{and} \quad h2(x,y) = 2|r_2 - \|\overrightarrow{C_2P}\|| + .02\,,$$

where $C_1 = (-3,5)$ and $C_2 = (10,-5)$ are the centers of two disks of radius $r_1 = 7$ and $r_2 = 13$ respectively. Ten iterations of tree and mesh adaption haven been necessary to capture the size fields. Table 2 and Figure 11 illustrate the result of quadtree adaption for this analytical example. Notice that the tree decomposition favors the creation of small edges, and more especially with this analytical function as the mesh gradation given by the sizing function is not compatible with the (almost) discrete gradation introduced by the [2:1] rule. Nevertheless, the tree decomposition is able to capture the behavior of the function.

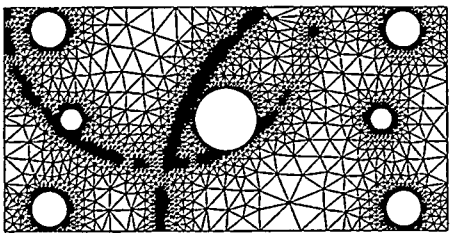
[5]Note : in most cases, almost half the number of elements are outside the computational domain.



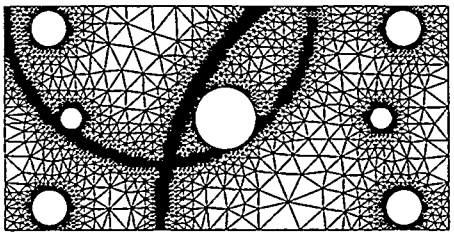
iteration 1



iteration 2



iteration 4



iteration 10

Figure 11: *Example of mesh adaption where two metric fields have been specified. The meshes illustrate iterations 1 (initial mesh), 2, 4 and 10 respectively.*

mesh - iter	N_p	N_e	Q_{min}	Q_{avg}	ℓ_{min}	ℓ_{max}	ℓ_{avg}	τ
cooler - 1	2,204	3,794	2.9	1.29	0.003	8.33	0.20	0.27
cooler - 2	2,808	4,975	2.9	1.28	0.003	21.13	0.25	0.32
cooler - 4	5,090	9,502	3.64	1.27	0.003	19.9	0.27	0.41
cooler -10	7,772	14,847	2.9	1.27	0.003	1.73	0.29	0.47

Table 2: *Statistics relative to the numerical evaluation of the adapted quadtree meshes.*

5 Conclusions

After a brief review of some elementary definitions related to quadtree-based mesh generation, we have introduced a method suitable to create meshes conforming a prescribed size map (or an intrinsic size map if no specific map was specified). The proposed method attempts to adapt the tree decomposition so that the cell sizes match the required local sizes. No filtering stage is required to remove closely spaced quadrant entities as a specific mesh optimization procedure has been designed to take care of these small features. The mesh optimization procedure is based on the normalized lengths of the mesh edges and aims at providing an optimal, unit size, mesh with respect to the given metric map. This approach can be used in an adaption scheme were the size map is provided by an a posteriori error estimate after the analysis of the numerical solution. Several application examples of planar meshes have been provided to illustrate the main features and the efficiency of the approach. The approach needs yet to be validated in the context of realistic numerical simulation examples (for instance in CFD Euler and Navier-Stokes computations).

This work can be considered as a preliminary stage toward the comprehension of tree-based decomposition methods, prior to investigate an algorithm in three dimensions.

References

[doCarmo-1976] M.P. DO CARMO (1976), Differential geometry of curves and surfaces, *Prentice Hall*, New Jersey.

[Ciarlet-1978] P.G. CIARLET (1978), *The Finite Element Method*, North Holland.

[Frey,Borouchaki-1998] P.J. FREY AND H. BOROUCHAKI (1998), Geometric evaluation of finite element surface meshes, to appear in *Finite Element in Analysis and Design.*

[Frey,Maréchal-1998] P.J. FREY ET L. MARÉCHAL (1998), Génération de maillages respectant une carte de tailles par une méthode de type 'quadtree', *RR INRIA*, to appear.

[Frey,George-1999] P.J. FREY AND P.L. GEORGE (1999), *Mesh Generation and Related Topics. Application to Finite Elements*, book to appear.

[Freitag-1997] L.A. FREITAG (1997), On combining Laplacian and optimization-based mesh smoothing techniques, *Joint ASME/ASCE/SES Summer Meeting McNu'97*, Northwestern Univ., USA, June 29-July 2, 37-43.

[George-1991] P.L. GEORGE (1991), *Automatic mesh generation. Applications to finite element methods*, Wiley.

[George-Borouchaki-1997] P.L. GEORGE ET H. BOROUCHAKI (1997), *Triangulation de Delaunay et maillage. Applications aux éléments finis*, Hermès, Paris. Also as *Delaunay triangulation and meshing. Applications to Finite Elements*, Hermès, Paris.

[Grice *et al.* 1988] K.R. GRICE, M.S. SHEPHARD, C.M. GRAICHEN (1988), Automatic, topologically correct, three-dimensional mesh generation by the finite octree technique, *Technical Report*, RPI Center for Interactive Computer Graphics.

[Kela *et al.* 1986] A. KELA, R. PERUCCHIO AND H. VOELCKER (1986), Toward automatic finite element analysis, *Comp. Mech. Eng.*, 57–71.

[Knuth-1975] D.E. KNUTH (1975), The Art of Computer Programming, 2nd ed., *Addison-Wesley*, Reading, Mass.

[Laug *et al.* 1996] P. LAUG, H. BOROUCHAKI ET P.L. GEORGE (1996), Maillage de courbes gouverné par une carte de métriques, *RR INRIA*, **2818**.

[Mitchell,Vavasis-1992] S. MITCHELL AND S. VAVASIS (1992), Quality Mesh Generation in Higher Dimensions, *Proc. 8th ACM Symp. Comp. Geometry*, 212-221.

[Perucchio *et al.* 1989] R. PERUCCHIO, M. SAXENA AND A. KELA (1989), Automatic mesh generation from solid models based on recursive spatial decompositions, *Int. j. numer. meth. eng.*, **28**, 2469–2501.

[Samet-1984] H. SAMET (1984), The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys*, **16**(2), 187-260.

[Shephard-1988] M.S. SHEPHARD (1988), Approaches to the automatic generation and control of finite element meshes, *Applied Mechanics Reviews*, **41**, 169-185.

[Shephard *et al.* 1988] M.S. SHEPHARD, F. GUERINONI, J.E. FLAHERTY, R.A. LUDWIG, P.L. BAEHMANN (1988), Adaptive solutions of the Euler equations using finite quadtree and octree grids, *Computers & Structures*, **30**, 327-336.

[Shephard,Georges-1991] M.S. SHEPHARD AND M.K. GEORGES (1991), Automatic three-dimensional Mesh Generation by the Finite Octree Technique, *Int. j. numer. meth. eng.*, **32**, 709-749.

[Thacker-1980] W.C. THACKER (1980), A brief review of techniques for generating irregular computational grids, *Int. j. numer. methods eng.*, **15**, 1335-1341.

[Yerry,Shephard-1983] M.A. YERRY AND M.S. SHEPHARD (1983), A modified-quadtree approach to finite element mesh generation, *IEEE Computer Graphics Appl.*, **3**(1), 39–46.

Mini-Tutorials

Computational Geometry for Mesh Generation

Marshall Bern
Xerox Palo Alto Research Center
333 Coyote Hill Road
Palo Alto, CA 94304
bern@parc.xerox.com

Abstract

This mini-tutorial will give an overview of the computational geometry relevant to mesh generation. Topics will include Delaunay triangulation, constrained Delaunay triangulation, surface interpolation, numerical conformal mapping, nonobtuse triangulation, minmax angle triangulation, and optimal smoothing.

The tutorial will be most relevant to unstructured triangular and tetrahedral meshing, somewhat relevant to unstructured quad and hexahedral meshing, and largely irrelevant (but still pretty geometry!) to structured meshing. The tutorial will concentrate on recent work that has not yet found its way into computational geometry books.

Mesh Quality: A Function of Geometry, Error Estimates or Both?

M. Berzins*

Abstract. *The issue of mesh quality for unstructured triangular and tetrahedral meshes is considered. The theoretical background to finite element methods is used to understand the basis of present-day geometrical mesh quality indicators. A survey of more recent work in the development of finite element methods reveals work on anisotropic meshing algorithms and on providing good error estimates that reveal the relationship between the error and both the mesh and the solution gradients. The realities of solving complex three dimensional problems is that such indicators are presently not available for many problems of interest. A simple tetrahedral mesh quality measure using both geometrical and solution information will be described. Some of the issues in mesh quality for unstructured tetrahedral meshes will be illustrated by means of a simple example.*

keywords. Mesh quality, unstructured meshes, error estimates, mesh generation.

1 Introduction

The range of problems solved by finite element and finite volume p.d.e. solvers based on triangular and tetrahedral meshes e.g [7] [44] is rapidly increasing. The original applications problem class for many such solvers was in the area of solid mechanics and elasticity in particular. These methods are being applied at present to a wide range of problems in solid and fluid mechanics ranging from linear elasticity to turbulent flows, [23]. This very broad spectrum of applications naturally raises the issue of whether or not the meshes being used are appropriate for the applications being considered.

The issue of whether the mesh is appropriate to represent the solution has been investigated almost as long as finite elements have been used. In order to state the important finite element results that formed a basis for existing mesh quality measures it is necessary to introduce some notation. Without loss of generality the case of linear finite elements on triangular or tetrahedral meshes will be considered. Define the error as being the difference between the linear approximation, u_{lin} and the true solution u i.e. $e_{lin}(x,y) = u_{lin}(x,y) - u(x,y)$. The L^2 error norm is defined by $||e_{lin}(x,y)||_{L^2}$ where

$$||e_{lin}(x,y)||^2_{L^2} = \int_T (e_{lin}(x,y))^2 dxdy \ . \tag{1}$$

The H^1 error norm is defined by $||e_{lin}(x,y)||_{H^1}$ where

$$||e_{lin}(x,y)||^2_{H^1} = \int_T (e_{lin}(x,y))^2 + (e_{lin,x}(x,y))^2 + (e_{lin,y}(x,y))^2 dxdy \ . \tag{2}$$

The seminorm of the H^2 space is defined by $|u|_2$ where

$$|u|_2 = \left(\sum_{|\delta|=2} \frac{2!}{\delta_1!\delta_2!} ||(\partial_x)^{\delta_1}(\partial_y)^{\delta_2} u||^2_{L^2} \right)^{1/2} \ . \tag{3}$$

Aside from the notion that meshes with regular or smoothly varying element sizes are more aesthetically pleasing, the starting point for the notion of mesh quality would appear to be the analysis leading to the minimum angle condition that the smallest angle should be bounded away from zero. This perhaps originated with Zlamal [45] and is quoted by Strang and Fix [38] together with a statement regarding how "poorly shaped" triangles may have an effect on the condition number of the linear algebra problem that must be solved. The correct version of this result came with the analysis of Babuska and Aziz [5], who showed that the requirement for triangles was that there should be no large angles. The general results of both Zlamal and Babuska and Aziz are of the form

$$||e_{lin}(x,y)||^2_{H^1} \ \leq \ \Gamma(\theta)|u|_2 \tag{4}$$

*Computational PDEs Unit, School of Computer Studies, The University, Leeds LS2 9JT.

where Zlamal [45] showed that $\Gamma(\theta) = h/sin(\theta_{min})$ for the minimum angle $\theta_{min} = min(\theta_1, \theta_2, \theta_3)$, see Figure 1. In contrast Babuska and Aziz showed that $\Gamma(\theta) = h/\Psi(\theta)$ where $\Psi(\theta)$ is a positive continuous and finite function and for $\theta \leq \gamma < \pi, \Psi(\theta) \geq \Psi(\gamma)$ where γ is a bound on the maximum interior angle of the triangle in Figure 1. This work was extended, much later, to tetrahedral elements by Krizek [24] in a similar spirit.

The precise way that these results influenced mesh generation code writers is unclear. Early mesh generation papers are covered by the surveys of Shephard [36] and Thacker [39]. In these surveys there is little explicit reference to how the theoretical work has been adopted, though Thacker does say that elements should be nearly equilateral otherwise instability may result. More recent surveys by Bern and Epstein [12] and Nielson [31], do mention the theoretical results and the monographs of Carey [16] and George and Borouchaki [19] treat the subject in more detail. The perceived meshing wisdom has thus been that if possible elements should have no small or large angles. In the case of tetrahedral meshes this has has led to geometric mesh quality indicators as described in Liu and Joe [26]. One example being Weatherill's edge quality estimator for tetrahedra of volume V and edge lengths h_i:

$$Q_w = \frac{1}{8.48528V} \left[(\Sigma \frac{h_i}{6})^3 \right] \quad . \tag{5}$$

Such indicators do a good job of identifying geometric imperfections in the mesh -an important task before any solution is computed on the mesh. The difficulty is that it is unclear that such indicators are valid for every solution on every mesh. The ideal solution is thus to understand the relationship between the error and the mesh. Recently there have been many attempts to dynamically modify triangular meshes so as to fit the solution better. Some of these methods will be described below - most of them lead to stretched meshes for anisotropic solutions. The main requirement is thus for error estimators that include both solution and geometry information. Such estimators are still in their infancy especially in 3D but it will be shown that it is possible to use interpolation errors, [13] and through a simple example on a tetrahedral mesh that the accuracy in the solution can depend critically on the mesh.

2 A Quality Indicator Based upon Finite Element Interpolation Theory

The decision as to whether or not (and how) a mesh should be refined should be based on an error estimate that reflects not only the interpolation error caused by approximating the solution by a finite element space on a given mesh but also the discretization error of the numerical method used to approximate the p.d.e. and the choice of norm used to measure the error. Rippa [34] makes a convincing case based on interpolation errors that long thin triangles do indeed form part of a good mesh for strongly anisotropic solutions. A good discussion of this topic also occurs in Nielson [31].

Berzins [13] derives a new mesh quality indicator from the work of Nadler [29] which gives a particularly appropriate expression for the interpolation error when a quadratic function is approximated by a piecewise linear function on a triangle. Consider the triangle T defined by the vertices v_1, v_2 and v_3 as shown in Figure 1. Let h_i be the length of the edge connecting v_i and v_{i+1} where $v_4 = v_1$. Nadler [29] considers the case in which a quadratic function

$$u(x,y) \;=\; \frac{1}{2}\underline{x}^T \; H \; \underline{x} \; where \quad \underline{x} \;=\; [x,y]^T \;, \tag{6}$$

where H is a constant $2x2$ real matrix, is approximated by a linear function $u_{lin}(x,y)$, as defined by linear interpolation based on the values of u at the vertices and shows that the error denoted by equation (1) above satisfies

$$\int_T (e_{lin}(x,y))^2 dx \; dy \;=\; \frac{A}{180} \left[((d_1 + d_2 + d_3)^2 + {d_1}^2 + d_2^2 + d_3^3 \right] \tag{7}$$

where A is the area of the triangle and $d_i = \frac{1}{2}(v_{i+1} - v_i)^T \; H \; (v_{i+1} - v_i)$ is the edge derivative along the v_i and v_{i+1} edge. Berzins [12] uses this result as the basis for an indicator that takes into account both the geometry and the solution behaviour by defining scaled edge derivatives by $\tilde{d}_i = |d_i|/d_{max}$ where $d_{max} \;=\; max\,[|d_1|, |d_2|, |d_3|]$. For notational convenience define $\underline{\tilde{d}} = [\tilde{d}_1, \tilde{d}_2, \tilde{d}_3]^T$ and

$$\tilde{q}(\underline{\tilde{d}}) \;=\; (\tilde{d}_1 + \tilde{d}_2 + \tilde{d}_3)^2 + \tilde{d}_1^2 + \tilde{d}_2^2 + \tilde{d}_3^2 \tag{8}$$

A measure of the anisotropy in the derivative contributions to the error is then provided by

$$q_{aniso} \;=\; \tilde{q}(\underline{\tilde{d}})/12 \; . \tag{9}$$

The relationship between q_{aniso} and the linear interpolation error is that in the case when the matrix H is positive definite, i.e. $d_i > 0$, then the indicator q_{aniso} is a scaled form of the interpolation error, [13], in this special case.

A consistent and related but geometry-only based indicator is then defined by:

$$q_m(\underline{h}) = \tilde{q}(\underline{h})/(16\sqrt{3}\,A), \quad where\ \underline{h} = [h_1, h_2, h_3]^T, \tag{10}$$

has value 1 for an equilateral triangle and tends to the value infinity as the area of a triangle tends to zero but at least one of its sides is constant. Bank [7] and Weatherill's [44] indicators are denoted by q_b and q_w and defined by

$$\frac{1}{q_b} = \frac{1}{4\sqrt{3}\,A}\left[(h_1^2 + h_2^2 + h_3^2\right], \quad q_w = \frac{1}{3\,A}\left[(h_1 + h_2 + h_3)^2\right] \tag{11}$$

respectively. Hence, from equations (8) and (9) the connection between these indicators is that

$$q_m(\underline{h}) = \frac{1}{4\,q_b} + q_w\,\frac{\sqrt{3}}{16}\,. \tag{12}$$

The choice of norm is not often considered but may be critical in deciding what is the best mesh. Given the linear interpolation error defined by equation (2), Berzins [12] considers the example of Babuska and Aziz [5] in which triangles of the form of that in Figure 1 are used to interpolate the function x^2 with x horizontal. Berzins [14] shows that in the L_2 norm the isosceles triangle is more accurate whereas in the H^1 norm right triangles are more accurate and the isosceles triangle is the worst choice as $\alpha \downarrow 0$ in Figure 1. Hence a good mesh in one norm is not a good mesh in another norm.

The extension to the case of non-quadratic functions may be considered by assuming that the exact solution is locally quadratic. Bank [7] uses such an approach inside the code PLTMG and calculates estimates of second derivatives. Adjerid, Babuska and Flaherty [1] use a similar approach based on derivative jumps across edges to estimate the error. An alternative approach is to use the ideas of Hlavacek et al. [20] to estimate nodal derivatives and hence second derivatives.

3 Mesh Movement Redistribution in 2/3D

The idea that it is important for the the shape of the elements to reflect local solution behaviour, particularly for highly directional flow problems, is well-known [15, 9, 25]. One of the significant steps in realising this understanding was the Moving Finite Element method of Keith Miller, see Baines [6], which continuously moves the mesh for transient problems. Some of the meshes shown by Baines are highly distorted. A similar approach, but rather simpler, was derived by Peraire et al. [32], who applied a simple local iterative procedure based on quantities such as pressure gradients to produce stretched meshes for highly-directional Euler equations flow problems. A key part of their algorithm is a simple Laplacian smoothing approach that has also been used by many others, e.g. Barth [9, 10].

A slightly different approach still is employed by Tourigny and Baines [40], who investigate the construction of locally optimal piecewise polynomial fits to data and produce meshes which vary from smooth to skewed, depending on the solution. The idea is further extended by Tourigny and Hulseman [41], who minimise an energy functional using a Gauss-Siedel method locally to get similarly skewed meshes.

Beinert and Kroner [11] move edges so that they are aligned with shock waves and also define a *Blue* directional refinement approach. For example in the right side of Figure 1 if the edges eT_1, eT_2 are parallel and aligned with the flow direction then the pairs of triangles is replaced by four anisotropic triangles. Although the indicator used to guide refinement is the gradient of the Mach number rather than an explicit error estimator, the results are nevertheless impressive.

The relative size of the edge indicators, d_i defined by equation(7) in the previous section gives a means of indicating which edges should be refined to reduce the error. One recent method to take advantage of such local gradients is the modified Delaunay approach of Borouchaki et al. [15] in which the local gradient information, of the form of d_i values, is used in conjunction with the Delaunay mesh generator to compute highly stretched grids for anisotropic flows in two space dimensions. The results presented by Borouchaki et al. show that this approach can give good results on problems with highly directional flows.

Other methods using the gradient quantities d_i defined in the previous section are the mesh generation procedure of Simpson [35] and the mesh modification procedure of Ait-Ali-Yahia et al.[2]. In the latter case the H matrix is

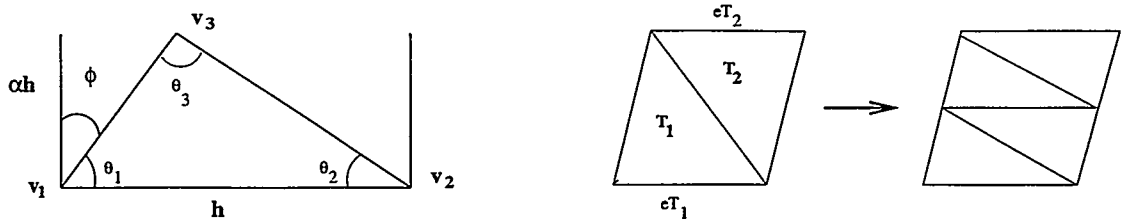


Figure 1: Babuska and Aziz Example Triangle and Blue Refinement of Two Triangles into Four.

modified to be positive definite and edge indicators, defined in the notation used here by $d_i/\sqrt{\Delta x_i^2 + \Delta y_i^2}$, are used to move the mesh. This approach thus scales the edge error component by the edge length. Ait-Ali-Yahia et al. [2] interpret d_i as the edge length in the H norm.

Mesh redistribution in 3D is less common but Freitag and Ollivier-Gooch [18] and Iliescu [22] give interesting algorithms for splitting tetrahedra. In Iliescu's approach pairs of tetrahedra satisfying convexity and angle conditions related to the flow direction are split into three tetrahedra so as to be aligned with the flow direction, see Figure (2). Freitag and Ollivier-Gooch [18] also provide convincing evidence that mesh smoothing can have beneficial consequences for the rate of convergence of the iterative solver.

A common feature of all the methods listed in this section is that although the mesh is improved in some sense, the criterion used is only indirectly related to the error.

4 Error estimators with Geometry effects

Recent work in error estimates is starting to reveal the explicit dependence of the error on both solution derivatives and on the mesh. An important stepping stone in this process was the work of Appel, [3, 4], which proved that one can benefit from the presence of small and even large angles of the elements. Appel also shows for bilinear elements that the interpolation and finite element errors coincide. Tsukerman [42, 43] derives a maximal eigenvalue condition which shows that it is the maximum eigenvalue of the element stiff matrix that characterises the impact of the shape of the element on the energy norm of the error of the finite element approximation.

Bank and Smith [7], in error analysis for the method used in the PLTMG code shows how the error can be written using d_i and q_b from Section 2 as a quotient of solution and geometry information:

$$\int_t |\bigtriangledown e_{lin}(x,y)|dx \approx \frac{d_1^2 + d_2^2 + d_3^2}{q_b} \tag{13}$$

This somewhat simpler form than the expressions in equation(7) and [14] comes about because Bank and Smith consider only the diagonal terms in a matrix to arrive at their approximation. While this error estimator only applies to steady problems Lang [25] considers transient problems and explicitly includes both solution derivative and geometry information in the error estimates he derives. For 2D reaction-diffusion p.d.e.s modelling highly-directional phenomena such as flame propagation, Lang proves the error estimate:

$$||e_{lin}(x,y)||_{H^1}^2 \quad \leq \quad \tilde{c}\left(\sum_{T \epsilon T_k} \eta_T^2\right)^{1/2} \tag{14}$$

where the local error estimator $\eta_T^2 = C^2(\tau, \lambda, T), D_T^2 U$ and $D_T^2 U$ is a computed approximation to $|u|_2$ as defined by equation (3). The constant $C(\tau, \lambda, T)$ is defined by

$$C(\tau,\lambda,T) = (1 + |\lambda| + \lambda^2)^2 h^2 (0.2587(1 + \frac{1}{\tau})h^2 + \frac{1}{\pi^2}(1 + |\lambda| + \lambda^2)) \tag{15}$$

and where with reference to Figure 1, $\lambda = tan(\phi)$, h is the longest edge and τ is the timestep. This estimate thus precisely describes the effect of both the geometry and the solution on the error and enables decisions regarding directional refinement to be taken.

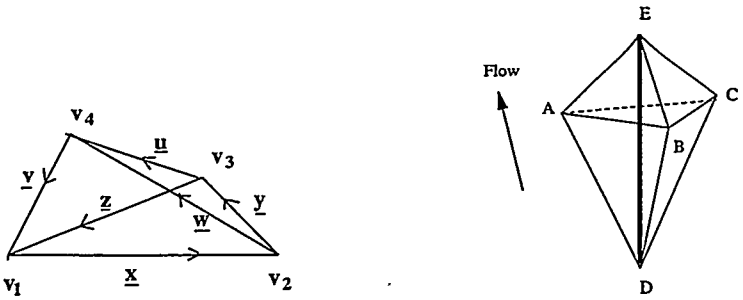


Figure 2: Example Tetrahedron and Iliescu's Directional Refinement Procedure.

5 Linear Tetrahedral Approximation of a Quadratic Function

Although there are now data-dependent tetrahedralisations, see Nielson [31], there are unfortunately very few error estimates for tetrahedral meshes that show the explicit dependence of the error on the mesh and the solution. The natural starting point is perhaps to try and use the interpolation error to assess how appropriate the mesh is for the computed solution. The simple mesh quality indicator of Berzins [13, 14] is based on linear interpolation error estimates and is derived by extending Nadler's [29] approach to tetrahedra by considering the case in which a quadratic function

$$u(x,y,z) \;=\; \frac{1}{2}\,\underline{x}^T\,H\,\underline{x} \;where\;\; \underline{x} \;=\; [x,y,z]^T \tag{16}$$

is approximated by a linear function $u_{lin}(x,y,z)$ defined by linear interpolation based on the values of u at the vertices of a tetrahedron T defined by the vertices v_1, v_2 , v_3 and v_4 as shown in Figure 2.

Let h_i be the length of the edge connecting v_i and v_{i+1} where $v_5 = v_1$. With reference to Figure 2 define the vectors $\underline{\hat{x}}$, $\underline{\hat{y}}$, $\underline{\hat{z}}$, $\underline{\hat{u}}$, $\underline{\hat{v}}$ and $\underline{\hat{w}}$ by $v_2 = v_1 + \underline{\hat{x}}$, $v_3 = v_2 + \underline{\hat{y}}$, $v_1 = v_3 + \underline{\hat{z}} v_4 = v_1 - \underline{\hat{v}}$, $v_4 = v_2 + \underline{\hat{w}}$, $v_4 = v_3 + \underline{\hat{u}}$. Berzins [13] defines the vector of second directional derivatives along edges by

$$\underline{d}^T = \frac{1}{2}\,[d_1,...,d_6\,] \;=\; \frac{1}{2}\,\left[\underline{\hat{x}}^T H \underline{\hat{x}},\;\; \underline{\hat{y}}^T H \underline{\hat{y}},\;\; \underline{\hat{z}}^T H \underline{\hat{z}},\;\; \underline{\hat{u}}^T H \underline{\hat{u}},\;\; \underline{\hat{v}}^T H \underline{\hat{v}},\;\; \underline{\hat{w}}^T H \underline{\hat{w}}\,\right].$$

and shows that the error may written in terms of the six directional derivatives along the edges d_i as:

$$\int_T (e_{lin}(x,y,z))^2 dx\; dy\; dz = \;\frac{6}{4}\,V\,\frac{8}{7!}\,\left[(\Sigma d_i)^2 - d_1 d_4 - d_2 d_5 - d_3 d_6 + \Sigma d_i^2\,\right] \;\;. \tag{17}$$

It is then possible to define the mesh quality indicator in the same way as in Section 2 in that the error is scaled by the maximum directional derivative d_{max}, the integral is scaled by the volume before taking the square root. In a similar way to as in Section 2 define

$$\tilde{Q}(\underline{\tilde{d}}) \;=\; \left[(\Sigma \tilde{d}_i)^2 - \tilde{d}_1\tilde{d}_4 - \tilde{d}_2\tilde{d}_5 - \tilde{d}_3\tilde{d}_6 + \Sigma \tilde{d}_i^{\,2}\,\right] \;\; where\;\underline{\tilde{d}} = [\tilde{d}_1,\tilde{d}_2,\tilde{d}_3,\tilde{d}_4,\tilde{d}_5,\tilde{d}_6]^T. \tag{18}$$

A measure of the anisotropy in the derivative contributions to the error is then provided by Q_{aniso} and a related geometry based indicator by Q_m where

$$Q_{aniso} \;=\; \tilde{Q}(\underline{\tilde{d}})/39 \;\; and\; Q_m(\underline{h}) = \frac{C}{V}\;\left[\;\tilde{Q}(\underline{\tilde{h}})\right]^{\frac{3}{2}} \tag{19}$$

where C is a scaling factor to ensure that the indicator has value one when $h_i = h$. A comparison between this geometry indicator, $Q_m(\underline{h})$, with that of Weatherill Q_w as defined by equation(5) was done by Berzins [13] who showed that the values of the two indicators are very similar. The anisotropic interpolation example used by Berzins, [14], shows that in such circumstances it is important to use indicators such as Q_{aniso} which involve solution information.

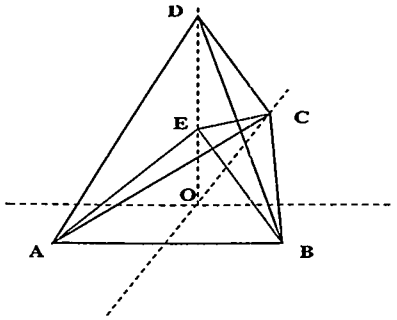


Figure 3: Example Mesh of Four Tetrahedra: ABCE,ABED,ACED and BCED

6 Example Laplace's Equation with Anisotropic Tetrahedra and Finite Element/Volume Schemes

The issue of mesh suitability for a given solution and numerical solver is recognised as a complex one with no easy answers. There are a variety of views concerning the sensitivity of numerical schemes to distorted meshes. Shephard [37] states that the stabilized FEM for example, appear to have no real problem with elements with angles of 179 degrees and 1,000,000 to 1 aspect ratios and that tetrahedra with small angles are well-understood to be needed for boundary layer calculations. In contrast, Millar [27, 28], et al. state that for Laplace's equation, finite volume schemes are less sensitive than finite element schemes to sliver-type tetrahedra in meshes. Given the similarity between the finite volume and element schemes in this case, see [9] the difference may be due to implementation issues such as those discussed by Putti and Cordes [33].

In order to understand better the dependency between the mesh and the error, the Laplaces equation, $\nabla^2 U = 0$, in three space dimensions of [27] will be used. The mesh of five points consists of a single tetrahedrob sub-divided into four by the addition of an internal point and is shown in Figure 3. The analytic solution given by

$$u(x,y,z) = e^{\pi z} cos(\pi y/\sqrt{2}) sin(\pi(x+0.5)/\sqrt{2}) \tag{20}$$

$$O = [0,0,0]^T \ , \ A = [-0.5,-0.5,0]^T, \ B = [0.5,-0.5,0]^T \ C = [0,1,0]^T \ , \ D = [0,0,1]^T, and \ \ E = [0,0,\epsilon]^T$$

where ϵ is a parameter that will be varied to test the sensitivity of the numerical solution to the mesh and in particular to distorted elements. The values at A, B, C, D are given by the exact solution and denoted by U_A, U_B, U_C, U_D. The scheme used to approximate the Laplacian is Barth's cell-vertex scheme [9, 10]. This gives a challenging situation for mesh quality indicators as the region associated with each node is composed of parts of all neighbouring tetrahedra. At point E the Laplacian is approximated by

$$\nabla^2 U = W_{EA}(U_A - u_E) + W_{EB}(U_B - u_E) + W_{ED}(U_A - u_E) + W_{ED}(U_D - u_E) \tag{21}$$

where u_E is the numerical approximation to the exact value U_E and is explicitly defined by the equation

$$u_E = (\ W_{EA}U_A + W_{EB}U_B + W_{EC}U_C + W_{ED}U_D \ )/(W_{EA} + W_{EB} + W_{EC} + W_{ED}) \tag{22}$$

In order that the solution satisfies a maximum principle all the weights W_{**} must be positive. [9, 10]. Barth also shows how this condition may not be met on a distorted mesh, but Putti and Cordes [33] show how to modify the method to avoid this and that this also improves the accuracy.

Denote the exact solution of the problem at node E by U_E then the p.d.e. truncation error, T.Error, is defined by

$$TError = W_{EA}(U_A - U_E) + W_{EB}(U_B - U_E) + W_{ED}(U_A - U_E) + W_{ED}(U_D - U_E) \tag{23}$$

and the relationship between the truncation error and the error is

$$Error = U_E - u_E = -TError/(W_{EA} + W_{EB} + W_{EC} + W_{ED}) \tag{24}$$

Table 1 shows the different mesh quality indicators and the interpolation error as the value of ϵ changes for two tetrahedra given by the points ABCE and ACED. The values for the tetrahedra ABED and BCED being similar to those of ACED. With reference to Table 1 Interp is the square of the interpolation error based on the exact solution. Error and T.Error are the error and truncation error defined by equations (23) and (24) respectively. The results in Table 1 show that the anisotropy indicator follows (not surprisingly) the trend of the interpolation error, but that the pointwise discretization error behaves very differently, especially for small values of ϵ. The low values of

Table 1: Q_{aniso} , Standard Mesh Quality Q_w and Error Values

	Tet. ABCE			Tet. ACED			Numerical Error		
ϵ	Q_{aniso}	Q_w	Interp	Q_{aniso}	Q_{nw}	Interp	U_E	Error	T. Error
0.001	0.35	621	3.4e-6	0.15	2.2	1.0e-3	-2.6e-2	0.42	640.
0.01	0.35	62	3.4e-5	0.15	2.2	1.0e-3	-1.1e-2	0.41	65.5
0.1	0.38	6.2	3.4e-4	0.15	2.3	9.5e-4	1.4e-1	0.03	7.45
0.5	0.38	1.5	1.6e-3	0.17	3.9	6.2e-4	6.7e-1	-0.02	-0.45
0.9	0.23	1.1	3.1e-3	0.21	20	1.8e-4	1.015	-3.7e-2	-4.843
0.99	0.21	1.1	3.6e-3	0.22	211	2.0e-5	1.075	-4.4e-3	-5.9
0.999	0.20	1.1	3.6e-3	0.23	211	2.1e-6	1.08	-4.4e-4	-6.05

Table 2: Values of the coefficients $W_{ea}, W_{eb}, W_{ec}, W_{ed}$

ϵ	W_{ea}	W_{eb}	W_{ec}	W_{ed}
0.001	-5.04e+2	-5.02e+2	-5.01e+2	-1.514
0.01	-5.4e+1	-5.2e+1	-5.1e+1	-1.636
0.5	-5.0	-3.0	-2.0	-1.5
0.99	-4.5	-2.5	-1.5	-1.34e+3
0.999	-4.5	-2.5	-1.5	-1.34e+4

the anisotropy indicator Q_{aniso} indicate potential problems. The geometry indicator does a good job of picking up the very large error for small ϵ but also erroneously identifies a problem with ϵ close to one, when the error is small. The change in sign of the error makes it possible to identify the optimal value of ϵ, and hence the optimal mesh. Experimentation shows that when $\epsilon = 0.45819234130$ the error is zero to roundoff error level.

The interesting result is that both mesh quality indicators do not really identify the relationship between the mesh and the error in the numerical solution. It is the differing size of the truncation error as caused by the method coefficients that has a dramatic effect on the error. In the case when $\epsilon = 0.001$ the large size of the three coefficients W_{ea}, W_{eb}, W_{ec} arises because the face angle between faces such as EBC and ABC is very close to π . Hence in this case the value $U_D$play little part in determining u_E. In contrast when ϵ is close to one only one coefficient is large and u_E is determined almost solely by U_D its closest neighbour. The values of these coefficients are shown in Table 2, the negative values indicating that the mesh is not a good one from the point of view of approximating the diffusion operator, [9].

7 Conclusions

The overall conclusion is that the only really satisfactory approach would seem to be to have an error estimator based on both solution and geometry information This would appear to be true for strongly directional fluid flows for which highly distorted meshes appear to be very effective. One approach to resolving this issue is to have computable error estimates for each solution component. At present, it is still often the case that such estimates may not be available or may not be reliable. It is also the case that the availability of such error estimates will always lag behind the problems being solved by practitioners. Hence the requirement must be to allow the user to supply mesh quality measures and to choose anisotropic remeshing options. There are, of course, many applications areas in which it is still rather difficult to even understand what constitutes a good mesh. One such area is turbulent combustion which may involve the interaction between many chemical species and complex fluid flows. Such problems are like to provide interesting challenges to the meshing community for some time to come.

References

[1] S.ADJERID, I.BABUSKA AND J.E.FLAHERTY , *On Finite Element Method* , SIAM J. Numer. Anal. (1998), (to appear)

[2] D.AIT-ALI-YAHIA, W.G. HABASHI, A.TAM, M-G.VALET AND M.FORTIN, *A directionally adaptive methodology using an edge based error estimate on quadrilateral grids*, Int. Jour. for Num. Meths in Fluids. (1996), Vol. 23, pp. 673–690.

[3] T.APPEL, *Anisotropic interpolation with applications to the finite element method.* Computing (47) 1992 pp 277-293.

[4] T.APPEL, *Interpolation of non-smooth functions on anisotropic finite element meshes.* Preprint SFB393/97-06. Fakultat fur Mathematik, Technische Universitat Chemnitz-Zwickau, March 1997. (http://www.tu-chemnitz.de/~tap/)

[5] M.BABUSKA AND K.AZIZ, *On the Angle Condition in the Finite Element Method,* SIAM J. Numer. Anal. 13 (1976), No 2, pp. 214–226.

[6] M.J.BAINES, *Moving Finite Elements.* Oxford Science Publications, 1994.

[7] R.E.BANK AND R.K.SMITH *Mesh Smoothing Using A Posteriori Error Estimates.* SIAM J. Numerical Analysis, 34 (1997), 979-997.

[8] D.W.BARNETTE. A Mathematical Basis for Automated Structured Grid Generation with Close Coupling to the Flow Solver. Sandia Report SAND97-2382, February 1998.

[9] T.J.BARTH, *Numerical Aspects of Computing Viscous High Reynolds Flows on Unstructured Meshes.* AIAA Paper 91-0721, 29th Aerospace Sciences Meeting, January 7-10 1991, Reno Nevada.

[10] T.J.BARTH, *Aspects of Unstructured Grids and Finite Volume Solvers for the Euler and Navier Stokes Equations* Lecture Notes Presented at the VKI Lecture Series 1994-95. See http://www.nas.nasa.gov/~Barth

[11] R.BEINERT AND D.KRONER. *Finite Volume Methods with Local mesh refinement in 2D* pp 39-53 in *Adaptive Methods-Algorithms Theory and Applications Numerical Methods for Fluid Dynamics* Volume 46. Proc of 9th Gamm Seminar Kiel January 22-24. Vieweg, Wiesbaden 1993.

[12] M.BERN, AND D.EPSTEIN, *Mesh Generation and Optimal Triangulation.* report CSL 92-1, Xerox Corporation, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.

[13] M.BERZINS, *A Solution-Based Triangular and Tetrahedral Mesh Quality Indicator* , SIAM Journal on Scientific Computing Vol 19, pp2051-2060, 1998.

[14] M.BERZINS, *Solution- Based Mesh Quality for Triangular and Tetrahedral Meshes,* pp 427-436 in Proceedings of 6th International Meshing Roundtable, Sandia Report SAND 97-2399, Sandia National Labs, PO Box 5800 MS 0441, Albuquerque, NM 87185-0441

[15] M.BOROUCHAKI, M.J.CASTRO-DIAZ, P.L. GEORGE, F.HECHT AND B.MOHAMMADI, *Anisotropic adaptive mesh generation in two dimensions for CFD* in Numerical Grid Generation in Computational Field Simulations, (eds) B.K.Soni, J.F. Thompson, J. Hauser, P.Eiseman. Published by NSF Center for Computational Field Simulations , Mississippi State University, Mississippi 39762 USA , April 06 ISBN 0-965 1627-02

[16] G.CAREY , *Computational Grids - Generation, Adaptation and Solution Strategies* , Taylor and Francis, Bristol PA 19007, 1997.

[17] E.F.D'AZEVEDO, C.H.ROMINE AND J.M.DONATO. *Coefficient Adaptive triangulation for Strongly Anisotropic Problems.* Report ORNL/TM 13086, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831.

[18] L.A.FREITAG AND C.OLLIVIER-GOOCH, *A Cost/Benefit Analysis of Simplicial Mesh Improvement Techniques as Measured by Solution Efficiency.* To appear in Journal of Computational Geometry and Applications.

[19] P.L.GEORGE AND M.BOROUCHAKI, *Delaunay Triangulation and Meshing- Applications to Finite Elements.* Editions Hermes, Paris France 1998, ISBN 2-86601-692-0.

[20] I.HLAVACEK, M.KRIZEK AND V.PISTORA, *How to recover the gradient of linear elements on nonuniform triangulations* , Applications of Mathematics 41 (1996), No 4, pp. 241–267.

[21] T.ILIESCU, *A flow-aligning algorithm for convection-dominated problems* , Accepted by Int. Jour. Num. Meths. in Engng. 1998

[22] T.ILIESCU, *A 3D flow-aligning algorithm for convection-dominated problems* , Accepted by Applied Math. Letters.

[23] K.E. JANSEN, *A stabilized finite element method for computing turbulence* to appear in Computer Methods in Applied Mechanics and Engineering, also see `http://scorec.rpi.edu/~kjansen/`.

[24] M.KRIZEK, *On the Maximum Angle Condition for Linear Tetrahedral Elements,* SIAM J. Numer. Anal. 29 (1992), No 2, pp. 513–520.

[25] J.LANG, *Two-dimensional fully adaptive solutions of reaction-diffusion equations.* Applied Numerical Mathematics 18 (1995) 223-240.

[26] A.LIU AND B.JOE, *Relationship between tetrahedron shape Measures.* BIT, 34 (1994), pp.268–287.

[27] G.L.MILLAR, D.TALMOR, S.H.TENG AND N.WALKINGTON. *A Delaunay Based Numerical Method for Three Dimensions: generation, formulation and partition.* pp.683-692 in Proc 27th ACM Symposium Theory of Computing '95, Las Vegas, USA. ACM Publications, New York.

[28] G.L.MILLAR, D.TALMOR, S.H.TENG N.WALKINGTON AND HAN WANG. *Control Volume Meshes using Sphere Packing- generation, refinement, and coarsening.* In the *5th International Meshing Roundtable,* page 47–61, October, 1996.

[29] E.J.NADLER, *Piecewise linear best l_2 approximation on triangles.* in C.K.Chui, L.L.Schumacher and J.D. ward (Eds) Approximation Theory V: Proceedings Fifth International Symposium on Approximation Theory. Academic Press New York 1986 pp. 499–502.

[30] E.J.NADLER, *Piecewise linear approximation on triangulations of a planar region.* M.Sc. Thesis, Division of Applied Mathematics, Brown University, Providence, RI , May 1985.

[31] G.M.NIELSON *Tools for triangulations and Tetrahedralisations and Constructing Functions Defined over Them.* Scientific Visualisation - Overviews,Methodologies and Techniques. eds. G.M.Nielson, H.Hagen and H.Muller E.J.Nadler., IEEE Computer Society, Los Alamitos , California, 1997.

[32] J.PERAIRE, M.VAHDATI, K.MORGAN AND O.C.ZIENKIEWICZ, *Adaptive Remeshing for Compressible Flow Calculations.* J. of Comp. Physics, 22 (1976), pp.131–149.

[33] M.PUTTI AND C.CORDES, *Finite Element Approximation of the Diffusion OPerator on Tetrahedra* , SIAM Journal on Scientific Computing Vol 19, pp1154-1168, 1998.

[34] S.RIPPA, *Long and thin triangles can be good for linear interpolation.* SIAM J. Numer. Anal. 29 (1992), No 1, pp. 257–270.

[35] R.B.SIMPSON, *Anisotropic Mesh Transformations and optimal error control,* Appl. Numer. Math., 14 (1994), pp183–198.

[36] M.SHEPHARD, *Approaches to the automatic generation and control of finite element meshes,* Appl. Mech. Rev. 41,4 April 1988), pp169–185.

[37] M.SHEPHARD, *Private communication* ,

[38] G.STRANG AND G.J.FIX *An Analysis of the Finite Element Method* Prentice Hall Series in Automatic Computation, Englewood Cliffs N.J.

[39] W.C.THACKER. *A brief review of techniques for generating irregular computational grids* Int. Jour. for Num. Meths. in Engng. Vol 15, No. 1980, pp. 1335-1341.

[40] Y.TOURIGNY AND M.J.BAINES. *Analysis of an Algorithm for Generating Locally Optimal Meshes for L2 Approximation by Discontinuous Piecewise Polynomials.* Math. Comp. Vol 66, No. 218, April 1997, pp. 623-650.

[41] Y.TOURIGNY AND F.HULSEMANN. *A New Moving Mesh Algorithm for the Finite Element Solution of Variational Problems.* SIAM J. Numer. Anal, Vol. 35, No 4. pp1416-1438, August 1998.

[42] I.TSUKERMAN, *A General Accuracy Criterion for Finite Element Approximation.* IEEE Transactions on Magnetics, Vol 34. No5 September 1998.

[43] I.TSUKERMAN, *Comparison of Accuracy Criteria for Approximation of Conservative Fields on Tetrahedra.* IEEE Transactions on Magnetics, Vol 34. No5 September 1998.

[44] N.P.WEATHERILL, M.J.MARCHANT AND O.HASSAN, *Unstructured Grid Generation and Adaptation for a transport Aircraft Configuration,* Paper presented at 1993 European Forum on Recent Developments and Applications in Aeronautical Computational Fluid Dynamics. Held at Bristol UK 1-3 September 1993.

[45] M.ZLAMAL. *On the Finite Element Method.* Numer. Math. v.12 1968, pp394-409.

A Survey of Unstructured Mesh Generation Technology

Steven J. Owen
Department of Civil and Environmental Engineering, Carngie Mellon University, Pittsburgh, PA.
and
ANSYS Inc., Canonsburg, PA.
steve.owen@ansys.com

Abstract

A brief survey of some of the fundamental algorithms in unstructured mesh generation is presented. Included is a discussion and categorization of triangle, tetrahedral, quadrilateral and hexahedral mesh generation methods currently in use in academia and industry. Also included is a brief discussion of smoothing, cleanup and refinement algorithms. An informal survey of currently available mesh generation software is also provided comparing some of their main features.

1. Introduction

Automatic unstructured mesh generation is a relatively new field. Within its short life span we have seen tremendous advances in many diverse fields. Once in a while, it is useful to step back from our own expertise and look at the entire picture of what is going on in the field. The purpose of this survey is to give some perspective to what the current trends are in mesh generation and outline some of the major technology areas, who is working in these fields and what software is available.

Probably the simplest approach is to first break down the technology based on the shape of element generated. We will consider triangle and quad generation methods in 2D and tetrahedral and hexahedral methods in 3D. Straddled between 2D and 3D, we have surface meshing, which has it's own set of issues. In addition we have another set of issues dealing with post processing of the mesh including smoothing, cleanup and refinement. Within each of these issues, have emerged a few clear categories of algorithms, which tend to dominate much of the literature and software. Not included in this survey are a wide variety of equally important related topics such as adaptive, anisotropic and parallel mesh generation as well as data structure and geometry management issues. Because of the immense scope of the field of unstructured mesh generation, I have limited this survey to include what I consider the more fundamental aspects of the field. Since I do not purport to be an expert in all fields of mesh generation, this will be at best, a cursory look at the main issues in each category.

1.1 Software Survey

As part of this paper, I conducted an informal survey of software vendors, research labs and educational institutions that develop mesh and grid generation software. The purpose was to get a broad picture of who was currently involved in developing software and what common algorithms were employed. The results of the survey are included as an appendix to this paper. They are also posted on the World Wide Web[1]. From the over 100 surveys mailed, approximately 80 responded. While the emphasis of the survey was unstructured, many unstructured codes are also included.

The survey is certainly not a complete list of all those developing software, but it does illustrate the wide range of mesh generation technology currently available. Included are simple research codes used by only a few people, to commercial codes integrated within complex analysis packages.

1.2 Structured vs. Unstructured

This survey paper focuses on unstructured meshing technology. There is a large group of literature[2,3] and software[4] that deals with structured meshing commonly referred to as "grid generation". Strictly speaking, a structured mesh can be recognized by all interior nodes of the mesh having an equal number of adjacent elements. For our purposes, the mesh generated by a structured grid generator is typically all quad or hexahedral. Algorithms employed generally involve complex iterative smoothing techniques that attempt to align elements with boundaries or physical domains. Where non-trivial boundaries are required, "block-structured" techniques can be employed which allow the user to break the domain up into topological blocks. Structured grid generators are most commonly used within the CFD field, where strict alignment of elements can be required by the analysis code or necessary to capture physical phenomenon.

Unstructured mesh generation, on the other hand, relaxes the node valence requirement, allowing any number of elements to meet at a single node. Triangle and Tetrahedral meshes are most commonly thought of when referring to unstructured meshing, although quadrilateral and hexahedral meshes can also be unstructured. While there is certainly some overlap between structured and unstructured mesh generation technologies, the main feature which distinguish the two fields are the unique iterative smoothing algorithms employed by structured grid generators.

2.0 Tri/Tetrahedral Meshing

Triangle and tetrahedral meshing are by far the most common forms of unstructured mesh generation. Most techniques currently in use can fit into one of three main categories:

1. Octree
2. Delaunay
3. Advancing Front

Although there is certainly a difference in complexity when moving from 2D to 3D, the algorithms discussed are for the most part applicable for both triangle and tetrahedral mesh generation.

2.1 Octree

The Octree technique was primarily developed in the 1980s by Mark Shephard's[5,6] group at Rensselaer. With this method, cubes containing the geometric model are recursively subdivided until the desired resolution is reached. Figure 1 shows the equivalent two-dimensional quadtree decomposition of a model. Irregular cells are then created where cubes intersect the surface, often requiring a significant number of surface intersection calculations. Tetrahedra are generated from both the irregular cells on the boundary and the internal regular cells. The Octree technique does not match a pre-defined surface mesh, as an advancing front or Delaunay mesh might, rather surface facets are formed wherever the internal octree structure intersects the boundary. The resulting mesh also will change as the orientation of the cubes in the octree structure is changed and can also require. To ensure element sizes do not change too dramatically, a maximum difference in octree subdivision level between adjacent cubes can be limited to one. Smoothing and cleanup operations can also be employed to improve element shapes.

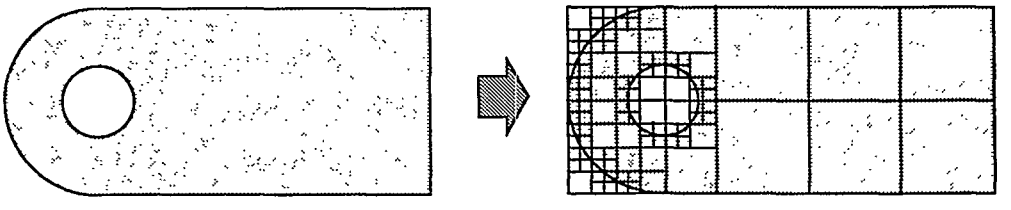


Figure 1. Quadtree decomposition of a simple 2D object

From the survey, only four of the 38 codes generating tetrahedral meshes reported using some form of octree technique. SCOREC[7] at Rensselaer develops a set of mesh generation tools called MEGA that

utilizes the Octree technique that is available through their partners program. A public domain octree mesh generator called QMG[8] is available from Steve Vivasis at Cornell.

2.2 Delaunay

By far the most popular of the triangle and tetrahedral meshing techniques are those utilizing the Delaunay[9] criterion. The Delaunay criterion, sometimes called the "empty sphere" property simply stated, says that any node must not be contained within the circumsphere of any tetrahedra within the mesh. A circumsphere can be defined as the sphere passing through all four vertices of a tetrahedron. Figure 2 is a simple two-dimensional illustration of the criterion. Since the circumcircles of the triangles in (a) do not contain the other triangle's nodes, the empty *circle* property is maintained. Although the Delaunay criterion has been known for many years, it was not until the work of Charles Lawson[10] and Dave Watson[11] that the criterion was utilized for developing algorithms to triangulate a set of vertices. A simple public domain 3D Delaunay triangulation program called Qhull is available from the University of Minneapolis. The criterion was later used in developing meshing algorithms by Timothy Baker[12] at Princeton, Nigel Weatherill[13] at Swansea, Paul-Louis George[14] at INRIA among others.

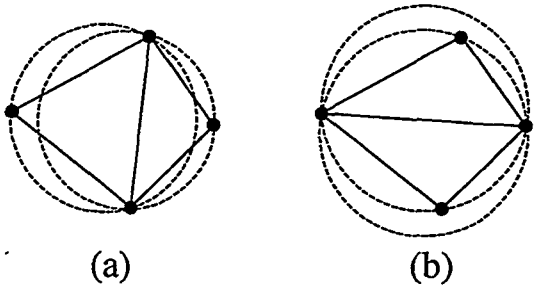


Figure 2. Example of Delaunay criterion. (a) maintains the criterion while (b) does not.

The Delaunay criterion in itself, is not an algorithm for generating a mesh. It merely provides the criteria for which to connect a set of existing points in space. As such it is necessary to provide a method for generating node locations within the geometry. A typical approach is to first mesh the boundary of the geometry to provide an initial set of nodes. The boundary nodes are then triangulated according to the Delaunay criterion. Nodes are then inserted incrementally into the existing mesh, redefining the triangles or tetrahedra locally as each new node is inserted to maintain the Delaunay criterion. It is the method that is chosen for defining where to locate the interior nodes that distinguishes one Delaunay algorithm from another.

2.2.1 Point insertion

The simplest point insertion approach is to define nodes from a regular grid of points covering the domain at a specified nodal density. In order to provide for varying element sizes, a user specified sizing function can also be defined and nodes inserted until the underlying sizing function is satisfied. Another approach is for nodes to be recursively inserted at triangle or tetrahedral centroids. Weatherill and Hassan[13] propose a tetrahedral mesh generation scheme where nodes are inserted at a tetrahedron's centroid provided the underlying sizing function is not violated.

An alternate approach is to define new nodes at element circumcircle/sphere centers as proposed by Chew[15] and Ruppert[16]. When a specific order of insertion is followed, this technique is often referred to as "Guaranteed Quality" as triangles can be generated with a minimum bound on any angle in the mesh. Jonathon Shewchuk[17] at CMU has developed a 2D version of this algorithm and makes it available free of charge for research purposes.

Similar to the circumcircle point insertion method, another technique introduced by Rebay[18] is the so-called, Voronoi-segment point insertion method. A Voronoi segment can be defined as the line segment between the circumcircle centers of two adjacent triangles or tetrahedra. The new node is introduced at a

point along the Voronoi segment in order to satisfy the best local size criteria. This method tends to generate very structured looking meshes with six triangles at every internal node.

Another method, introduced by Marcum[19] is an advancing front approach to node insertion. Nodes are inserted incrementally, but added from the boundary towards the interior. Each facet is examined to determine the ideal location for a new fourth node on the interior of the existing Delaunay mesh. The node is then inserted and local reconnection is performed. This method tends to generate elements well aligned with the boundary with a very structured appearance to the mesh. Dave Marcum provides both a 2D and 3D version of his mesh generators through the ERC[20] at Mississippi State.

One straightforward method used by INRIA[21] in their mesh generator GSH3D[22], is point insertion along edges. A set of candidate vertices is generated by marching along the existing internal edges of the triangulation at a given spacing ratio. Nodes are then inserted incrementally, discarding nodes that would be too close to an existing neighbor. This process is continued recursively until a background sizing function is satisfied.

A variety of other methods for point insertion have also been proposed, but most have a similar flavor to those discussed above

2.2.2 Boundary Constrained Triangulation

In many finite element applications, there is a requirement that an existing surface triangulation be maintained. In most Delaunay approaches, before internal nodes are generated, a three dimensional tessellation of the nodes on the geometry surface is produced. In this process, there is no guarantee that the surface triangulation will be satisfied. In many implementations, the approach is to tessellate the boundary nodes using a standard Delaunay algorithm without regard for the surface facets. A second step is then employed to force or recover the surface triangulation. Of course by doing so, the triangulation may no longer be strictly "Delaunay", hence the term "Boundary Constrained Delaunay Triangulation".

In two dimensions the edge recovery is relatively straightforward. George[23] describes how the edges of a triangulation may be recovered by iteratively swapping triangle edges. The process is considerably more complex in three dimensions, since after recovering all edges in the surface triangulation, there is no guarantee that the surface facets themselves will be recovered. Additional facet recovery operations can be required to maintain the surface triangulation. While the two dimensional recovery process is guaranteed to produce a boundary conforming triangulation, there are cases[24] in three dimensions where a valid triangulation can not be defined without first inserting additional vertices. This fact increases the complexity of any three dimensional boundary recovery procedure. Two different methods presented in the literature for recovery of the boundary include George[14] and Weatherill[13].

In the first approach defined by George[14] and implemented in INRIA's GSH3D[22] software, edges are recovered by performing a series of tetrahedral transformations by swapping two adjacent tetrahedra for three, as shown in Figure 3. Where a swap cannot resolve the edge, nodes must sometimes be inserted. After edges have been recovered, in order to recover the face, additional transformations are performed, mostly characterized by swapping three adjacent tetrahedra at an edge for two. More complex transformations or additional nodes can be inserted during the face recovery phase if the transformations do not resolve the surface facet.

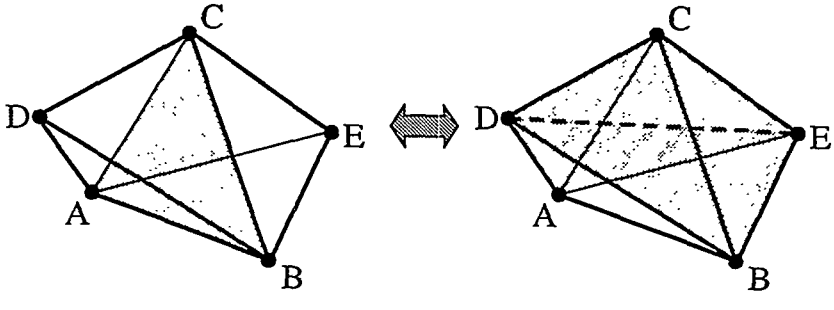


Figure 3. Tetrahedral transformation where two tets are swapped for three.

The second approach defined by Weatherill also involves an edge recovery phase and a face recovery phase. The main difference with this approach is that rather than attempting to transform the tetrahedra to recover edges and faces, nodes are inserted directly into the triangulation wherever the surface edge or facet cuts non-conforming tetrahedra. This process temporarily adds additional nodes to the surface. Once the surface facets have been recovered, additional nodes that were inserted to facilitate the boundary recovery are deleted and the resulting local void retriangulated.

Another approach presented by Barry Joe[25], is able to avoid the boundary recovery problem altogether. Provided the geometry is convex, Joe is able to define a boundary conforming tetrahedral mesh. The emphasis in this method, rather than attempting to repair the boundary of an arbitrary non-convex surface triangulation, is to decompose the geometry into convex regions that can be separately processed. An older unsupported public domain version of Barry Joe's code, Geompack, is available from the University of Alberta[26] via anonymous ftp.

2.3 Advancing Front

Another very popular family of triangle and tetrahedral mesh generation algorithms is the advancing front, or moving front method. Two of the main contributors to this method are Rainald Lohner[27,28] at George Mason University and S. H. Lo[29,30] at the University of Hong Kong. In this method, the tetrahedra are built progressively inward from the triangulated surface. An active front is maintained where new tetrahedra are formed. Figure 4 is a simple two-dimensional example of the advancing front, where triangles have been formed at the boundary. As the algorithm progresses, the front will advance to fill the remainder of the area with triangles. In three-dimensions, for each triangular facet on the front, an ideal location for a new fourth node is computed. Also determined are any existing nodes on the front that may form a well-shaped tetrahedron with the facet. The algorithm selects either the new fourth node or an existing node to form the new tetrahedron based on which will form the best tetrahedron. Also required are intersection checks to ensure that tetrahedron do not overlap as opposing fronts advance towards each other. A sizing function can also be defined in this method to control element sizes. Lohner[28] proposed using a course Delaunay mesh of selected boundary nodes over which the sizing function could be quickly interpolated. A version of S. H. Lo's advancing front mesh generator is available with the ANSYS[31] suite of mesh generation tools.

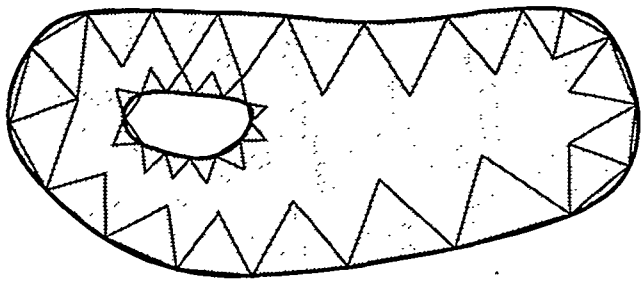


Figure 4. Example of advancing front where one layer of triangles has been placed

A form of the advancing front method, sometimes called "advancing layers", is also used for generating boundary layers for CFD, Navier-Stokes applications. This method lends itself well to control of element sizes near the boundary. Pirzadeh[32] presents a method where the elements are stretched in the direction of the boundary, the expected direction of fluid flow. A public domain version of Pirzadeh's code, VGRID[33] is available from NASA, Langley.

3. Quad/Hexahedral Meshing

Automatic unstructured mesh generation algorithms have lent themselves more readily to triangle and tetrahedral meshing. As a result, most of the literature and software are triangle and tetrahedral. In spite of this, there is a significant group of literature that focuses on unstructured quad and hexahedral methods. Unstructured quad[34] and hex[35] meshing software have also become widely available in recent years. Unlike

triangle and tetrahedral methods, extension from a 2D quadrilateral algorithm to a 3D hexahedral method is not generally straightforward.

3.1 Mapped Meshing

When the geometry of the domain is applicable, quad or hex mapped meshing[36] will generally produce the most desirable result. Although mapped meshing is considered a structured method, it is quite common for unstructured codes to provide a mapped meshing option. For mapped meshing to be applicable, opposite edges of the area to be meshed must have equal numbers of divisions. In 3D, each opposing face of a topological cube must have the same surface mesh. This can often be impossible for an arbitrary geometric configuration or can involve considerable user interaction to decompose geometry into mapped meshable regions and assign boundary intervals. In order to reduce human interaction, research has be done in recent years through the CUBIT[37] project at Sandia National Labs to automatically recognize features[38]and decompose geometry[39] into separate mapped meshable areas and volumes. Work has also been done to automate interval assignments[40].

Another category of mapped meshing, also developed as part of the CUBIT[37] project is referred to as sub-mapping[41]. This method, rather than decomposing the geometry directly, determines an appropriate *virtual* decomposition based on corner angles and edge directions. The separate map-meshable regions are then meshed separately. This method is suitable for blocky shapes and volumes that have well defined corners and cube-like regions.

Sweeping, sometimes referred to as 2 ½-D meshing, is another class of mapped hexahedral meshing. A quadrilateral mesh can be *swept* through space along a curve. Regular layers of hexahedra are formed at specified intervals using the same topology as the quadrilateral mesh. This technique can be generalized to mesh certain classes of volumes by defining so-called *source* and *target* surfaces. Provided the source and target surface have similar topology and the surfaces are connected by a set of map-meshable surfaces, the quad elements of the source area can be swept through the volume to generate hexahedra as shown in Figure 5. Care must be taken in locating internal nodes during the sweeping process and several papers[42,43] have been presented addressing this issue.

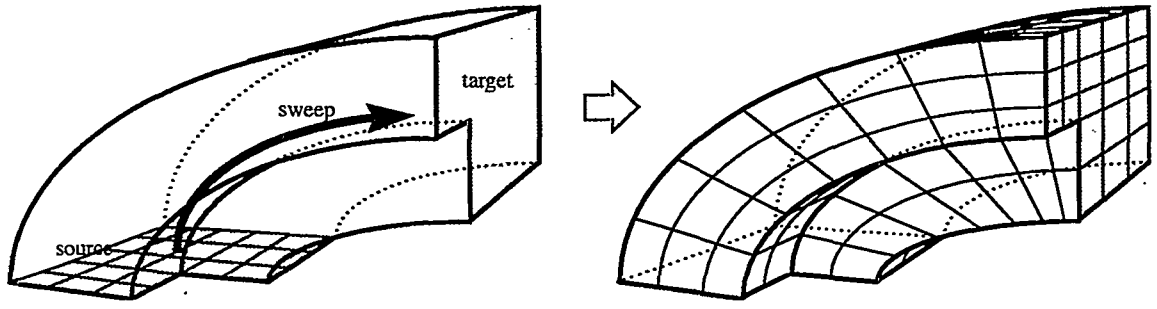


Figure 5. Hex elements generated by sweeping

Blacker[44] generalizes and extends the applicability of sweeping when he introduces the *Cooper Tool.* The Cooper tool allows for multiple source and target surfaces while still requiring a single sweep direction. With this tool, the topology is allowed to branch or split along the sweep direction. In addition, the topology of source and target surfaces are not required to be similar. With these requirements relaxed, a greater subset of geometry may be meshed with generally very high quality elements. The cooper tool is provided as part of the Fluent pre-processor, Gambit[45].

3.2 Unstructured Quad Meshing

Unstructured quadrilateral meshing algorithms can, in general, be grouped into two main categories: direct and indirect approaches. With an indirect approach, the domain is first meshed with triangles. Various algorithms are then employed to convert the triangles into quadrilaterals. With a direct approach,

quadrilaterals are placed on the surface directly, without first going through the process of triangle meshing.

3.2.1 Indirect Methods

One of the simplest methods for indirect quadrilateral mesh generation includes dividing all triangles into three quadrilaterals, as shown in Figure 6. This method guarantees an all-quadrilateral mesh, but a high number of irregular nodes are introduced into the mesh resulting in poor element quality. An alternate algorithm is to combine adjacent pairs of triangles to form a single quadrilateral as shown in Figure 7. While the element quality increases using this method, a large number of triangles may be left.

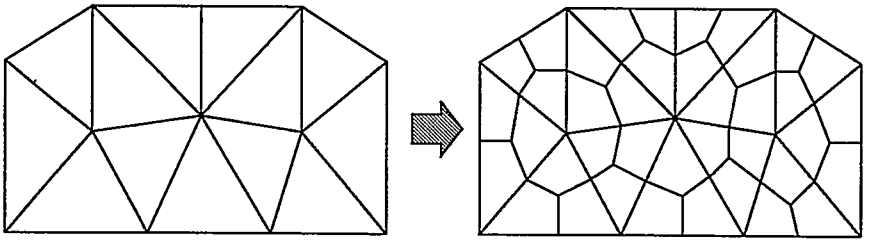


Figure 6. Quad mesh generated by splitting each triangle into three quads

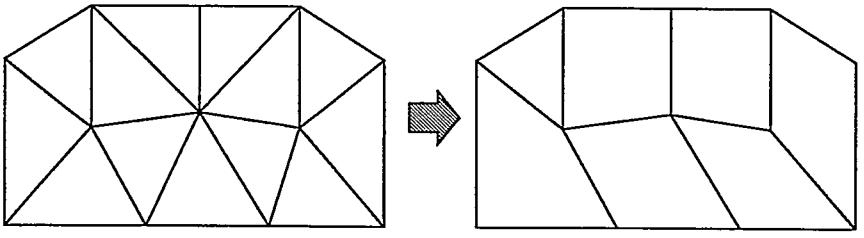


Figure 7. Quad-dominant mesh generated by combining triangles.

The triangle combining method can be improved, if some care is taken in the order in which triangles are combined. In an effort to maximize the number of quadrilaterals, Lo[46] defined an algorithm that suggested several heuristic procedures for the order in which triangles could be combined. The result is a quad-dominant mesh containing a minimal number of triangles. Johnston[47] proposes additional local element splitting and swapping strategies to increase the number and quality of quads.

Lee[48] later enhances Lo's [46] strategy by including local triangle splitting. In addition, an advancing front approach is used over the initial triangles. An initial set of fronts is defined consisting of the edges of triangles at the boundary of the domain. Triangles are systematically combined at the front, advancing towards the interior of the area. Each time a set of triangles is combined the front advances. The front always defines the division between quadrilaterals already formed and triangles yet to be combined. With this technique, Lee is able to guarantee an all-quadrilateral mesh, provided the initial number of edges on the boundary is even.

Since all operations are local, indirect methods have the advantage of being very fast. Global intersection checks are not necessary as is required with some forms of direct methods. The drawback to indirect methods is that there are typically many irregular nodes left in the mesh. Even if few irregular nodes exist, there is no guarantee that the elements will align with the boundary, a desirable property for some applications. Some of the irregular nodes can be reduced, and hence element quality increased by performing topological clean-up operations (discussed later).

Another method recently introduced by the author, known as Quad Morphing[49] also utilizes an advancing front approach to convert triangles to quads, but is able to significantly reduce the number of irregular nodes in the mesh. With this approach, local edge swaps are performed and additional nodes introduced in order to ensure boundary alignment and orthogonality. Any number of triangles may be deleted to create a single quad.

3.2.2 Direct Methods

Many methods for direct generation of quad meshes have been proposed. Among these methods, there appears to be two main categories. The first are methods that rely on some form of decomposition of the domain into simpler regions than can be resolved by one of a series of templates. The second category are those that utilize a moving front method for direct placement of nodes and elements.

3.2.2.1 Quad Meshing by Decomposition

The quad-tree decomposition technique proposed by Baehmann[50] is among the first methods utilizing decomposition of the area for quadrilateral meshing. After an initial decomposition of the 2D space into a quad-tree based on local feature sizes, quadrilateral elements are fitted into the quad-tree *leaves*, adjusting nodes in order to conform to the boundary.

Talbert[51] later introduces another decomposition technique. With this approach, the domain is recursively subdivided into simple polygonal shapes. The resulting polygons satisfy a limited number of templates into which quadrilateral elements are inserted. Chae[52] has recently proposed enhancements to Talbert's algorithm with similar work presented by Nowottny[53].

Quadrilateral meshing utilizing a *medial axis* decomposition of the domain was first introduced by Tam[54]. The medial axis can be thought of as a series of lines and curves generated from the midpoint of a maximal circle as it is rolled through the area (Figure 8). Having decomposed the area into simpler regions, sets of templates are then employed to insert quadrilaterals into the domain. Linear programming techniques are used in order to maintain compatibility of element divisions between adjoining regions of the domain.

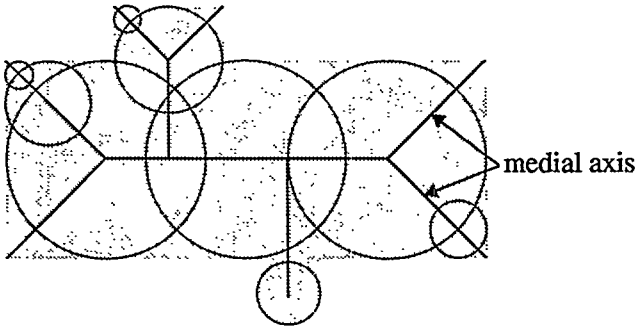


Figure 8. Decomposition of an area using the medial axis

Joe[55] also utilizes decomposition algorithms to decompose the area into convex polygons. Using techniques previously developed for triangle mesh generation[56], Joe constructs a boundary constrained quadrilateral mesh within each convex sub-domain of the area.

3.2.2.2 Advancing Front Quad Meshing

Zhu[57] is among the first to propose a quadrilateral meshing algorithm using an advancing front approach. Starting with an initial placement of nodes on the boundary, individual elements are formed by projecting edges towards the interior. Two triangles are formed using traditional triangle advancing front methods and then combined to form a single quadrilateral.

The *paving* algorithm introduced by Blacker and Stephenson[58], presents a method for forming complete rows of elements starting from the boundary and working in. Methods for projection of nodes, handling of special geometric situations and intersection of opposing fronts are discussed. Cass[59] further developed paving, by generalizing the method for three-dimensional surfaces. White[60] recently proposed enhancements to the paving algorithm suggesting individual placement of elements rather than complete rows. The paving algorithm is currently implemented as part of the CUBIT[37] software as well as several commercial packages including MSC Patran[61] and Fluent's Gambit[45] software.

3.3 Unstructured Hex Meshing

Similar to quadrilateral meshing, there are both direct and indirect methods for unstructured hex meshing.

3.3.1 Indirect Methods

Indirect methods, although not in wide use have been proposed for some applications[62]. Provided a solid can be tet meshed, each tetrahedron can be subdivided into four hexahedra as shown in Figure 9. Most finite element analysts, because of the poor element quality that will in general result, have rejected this solution.

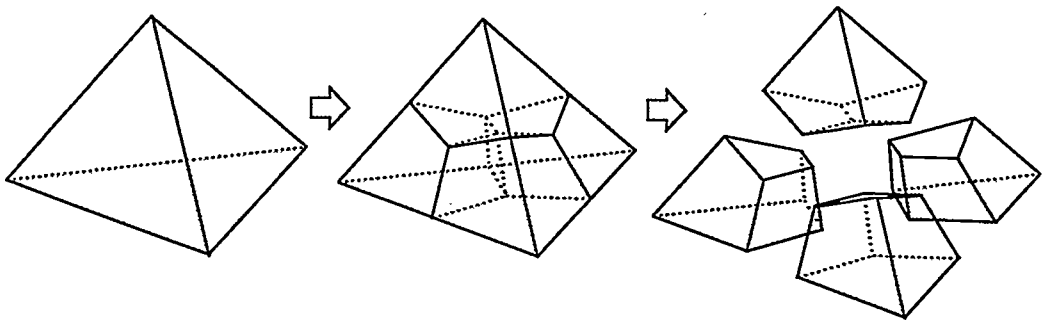


Figure 9. Decomposition of a tetrahedron into four hexahedra

An equivalent indirect hexahedral mesh generation scheme that will combine tetrahedra, similar to combining triangles to form quadrilaterals has not been presented in the literature. The simplest tetrahedralization of a cube will contain five tetrahedra. An indirect method that combines tets to form hexes would therefore need to look for combinations of five or more tetrahedra to form a single hexahedra. This problem to date has not proved a reasonable nor tractable method for mesh generation.

3.3.2 Direct Methods

There are currently four distinct strategies proposed for unstructured all-hex mesh generation that are predominant in the literature:

1. grid-based
2. medial surface
3. plastering
4. whisker weaving

3.3.2.1 Grid-Based

The grid-based approach, proposed by Schneiders[63] involves generating a *fitted* three dimensional grid of hex elements on the interior of the volume. Hex elements are added at the boundaries to fill gaps where the regular grid of hexes does not meet flush with the surface. This method, while robust, tends to generate poor quality elements at the boundary of the volume. Hex elements will in general not be aligned with the boundary. The resulting mesh generated from the grid-based approach is also highly dependent upon the orientation of the interior grid of hex elements. In addition, element sizes must be approximately all the same. In recent work, Weiler[64] and Schneiders[65] have introduced modifications that allow for significant transition in element sizes utilizing an octree decomposition of the domain. Mesh generators based on the grid-based approach are available in the Hexar[66] software from Cray Research and in MARC's Mentat[67] software.

3.3.2.2 Medial Surface

Medial surface methods[68,69,70] involve an initial decomposition of the volume. As a direct extension of the medial axis method for quad meshing, the domain is subdivided by a set of medial surfaces, which can be thought of as the surfaces generated from the midpoint of a maximal sphere as it is rolled through the volume. The decomposition of the volume by medial surfaces is said to generate map meshable regions. A series of templates for the expected topology of the regions formed by the medial surfaces are utilized to fill the volume with hexahedra. Linear programming is used to ensure element divisions match from one region to another. This method, while proving useful for some geometry, has been less than reliable for general geometry. Robustness issues in generating the medial surfaces as well as providing for all cases of regions defined by the medial surfaces has proved to be a difficult problem. Medial surface methods are incorporated into the FEGS' CADFix[71] hexahedral mesh generator and within Solidpoint's Turbomesh[72] software.

3.3.2.3 Plastering

Plastering[73,74] is an attempt to extend the paving algorithm to three dimensions. With this method, elements are first placed starting with the boundaries and advancing towards the center of the volume as shown in Figure 10. A heuristic set of procedures for determining the order of element formation is defined. Similar to other advancing front algorithms, a current front is defined consisting of all quadrilaterals. Individual quads are projected towards the interior of the volume to form hexahedra. In addition, plastering must detect intersecting faces and determine when and how to connect to pre-existing nodes or to *seam* faces. As the algorithm advances, complex interior voids may result, which in some cases are impossible to fill with all-hex elements. Existing elements, already placed by the plastering algorithm must sometimes be modified in order to facilitate placement of hexes towards the interior.

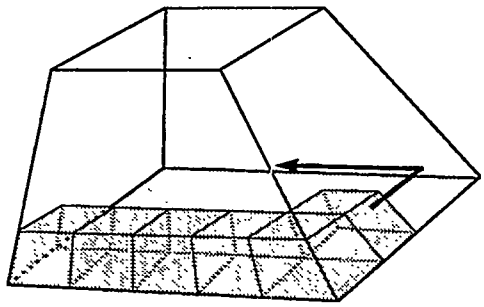


Figure 10. Plastering process forming elements at the boundary.

Currently, the plastering algorithm has not been proven to be reliable on a large class of problems. Although in many cases, several layers of hex elements may be successfully placed on the boundary of the volume, intersection and closure procedures are less than reliable. Sandia's CUBIT[37] project is continuing research on plastering and makes it available in their software.

3.3.2.4 Whisker Weaving

Whisker weaving, first introduced by Tautges and Blacker[75], is based on the concept of *the spatial twist continuum* (STC)[76]. Tautges describes the STC as the dual of the hexahedral mesh, represented by an arrangement of intersecting surfaces which bisect hexahedral elements in each direction. Figure 11 shows a simple representation of the *twist planes* of the STC defined for a volume composed of only two hexahedra.

The principal behind whisker weaving is to first construct the STC or dual of the hex mesh. With a complete STC, the hex elements can then be fitted into the volume using the STC as a guide. This is done by beginning with a topological representation of the loops formed by the intersection of the twist planes with the surface. The loops can be easily determined from an initial quad mesh of the surface. The objective of the whisker weaving algorithm is to determine where the intersections of the twist planes will

occur within the volume. Since this is done topologically, there are no actual intersection calculations performed. Once a valid topological representation of the twist planes has been achieved, hexes are then formed inside the volume. One hex is formed wherever three twist planes converge.

The whisker weaving algorithm has achieved some success, but has yet to prove itself as robust and reliable for a wide variety of problems.

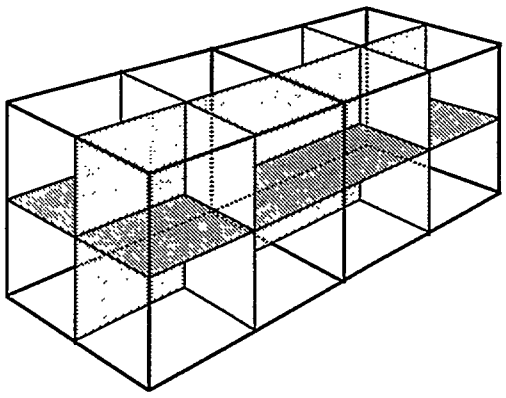


Figure 11. The STC composed of four twist planes, for a solid composed of two hexahedra

3.4 Hex-Dominant Methods

Since most methods for all-hex meshing appear to be less than robust, some researchers have proposed using a mixed hexahedra/tetrahedra mesh. A hex-dominant approach appears to be satisfactory in many cases. One simple approach introduced by the author[77] is to manually subdivide the geometry into regions that will readily accept a mapped mesh and those that are more geometrically complex. Within the complex regions a tet mesh is defined. Wherever the tet elements interface directly with hex elements, a pyramid shaped element may be formed. This option is provided with the ANSYS[31] mesh generation software.

Tuchinsky[78] recently proposed an algorithm for combining both plastering and tetrahedral meshing technologies. Using the plastering algorithm, hex elements are advanced as far as possible into the volume. The remaining voids within the volume are then filled with tetrahedra. The user also has the option of forming pyramid shaped elements at the interface between hex and tet elements. The CUBIT[37] software now provides an option to allow a hex-dominant mesh.

Min[79] also presents a similar method for hex-dominant meshing, utilizing offset geometry from the boundaries in order to form layers of hexes. After a series of shrunken shells have been advanced towards the interior of the volume, the remainder of the volume is filled with tetrahedra. In addition to tets and hexes, Min introduces pyramid and wedge shaped elements where applicable.

4. Surface Meshing

Many of today's mesh generation problems involve the formation of elements on arbitrary three-dimensional surfaces. These surfaces are typically represented by NURBS, which have been generated within a commercial CAD package. The resulting surface elements can either be used directly as structural shell elements, or used as input to a volumetric mesh generator. In either case, the algorithms used for two-dimensional mesh generation require some modification in order to generalize them for use on three-dimensional surfaces. Surface mesh generation algorithms can be classified as either parametric space or direct 3D.

4.1 Parametric Space

Parametric space algorithms will form elements in the two-dimensional parametric space of the surface. Since all NURBS surfaces have an underlying u-v representation, it can often be efficient to mesh in two dimensions and as a final step, map the u-v coordinates back to world space, x-y-z coordinates. The drawback to this method is that the elements formed in parametric space may not always form well-shaped elements in three dimensions once mapped back to the surface. To resolve this, parametric surface meshers can do one of two things: 1) modify or reparamaterize the underlying parametric representation so there is a reasonable mapping from parametric space to world space; or 2) modify the mesh generation algorithm so that stretched or anisotropic elements meshed in 2D will map back to well-shaped, isotropic elements in 3D.

The first method requires that in order to have a good paramaterization, the surface derivatives, (Δu, Δv), should not vary widely over the domain. Some exact arc-length reparamaterizations have been defined in the literature[80], but can be excessively costly. An approximate arc-length paramaterization or "warped parametric space" can be defined by selectively evaluating surface derivatives over the domain and adjusting local u-v values to hold the magnitude of Δu, Δv roughly constant. For many cases, a warped parametric space can generate reasonable surface meshes, but there are many problems that the reparamaterization cannot adequately resolve. For this reason, much of the literature on surface meshing focuses on the second option of forming anisotropic elements in 2D that will map back to isotropic elements in 3D.

A common method used in practice is to take advantage of surface derivatives, Δu, Δv, easily computed from a NURBS surface. George and Borouchaki[81] propose the use of a metric derived from the first fundamental form of the surface. The metric is in the form of a 2X2 matrix and is used to transform vectors and distances in parametric space. With their Delaunay approach, the "empty circle" property, effectively becomes an "empty ellipse" property. Also included with the metric is the option to incorporate element sizing and stretching properties. A similar approach to parametric Delaunay surface meshing is presented by Chen and Bishop[82] and available in MARC's Mentat[67] software. Equivalent advancing front surface mesh generation algorithms, which utilize a metric derived from the first fundamental form of the surface are presented independently by Cuilliere[83] and Tristano[84]. Tristano's implementation is available in a recent release of the ANSYS[31] mesh generation tools.

4.2 Direct 3D

Direct 3D surface mesh generators form elements directly on the geometry without regard to the parametric representation of the underlying geometry. In some cases where a parametric representation is not available or where the surface paramaterization is very poor, direct 3D surface mesh generators can be useful. Lau and Lo[85,86] present an advancing front approach for arbitrary 3D surfaces. In this method surface normals and tangents must be computed in order to compute the direction of the advancing front. In addition, a significant number of surface projections are required to ensure that new nodes remain on the surface. Also of significance is the increased complexity of the intersection calculations required to ensure that triangles on the surface do not overlap.

A direct 3D implementation[59] of the paving[44] algorithm is also available in the CUBIT[37] software. Similar issues regarding additional projection and evaluations are also of significance to 3D paving. Cass[59] defines a heuristic "sticky space" in order to detect intersecting or overlapping quadrilaterals.

5. Mesh Post-processing

It is rare that any mesh generation algorithm will be able to define a mesh that is optimal without some form of post-processing to improve the overall quality of the elements. The two main categories of mesh improvement include smoothing and clean-up. Smoothing includes any method that adjusts node locations while maintaining the element connectivity. Clean-up generally refers to any process that changes the element connectivity.

5.1 Smoothing

Most smoothing procedures involve some form of iterative process that repositions individual nodes to improve the local quality of the elements. A wide variety of smoothing techniques have been proposed. These methods can generally be classified as follows:

1. Averaging methods
2. Optimization-based methods
3. Physically-based methods
4. Mid-node placement

5.1.1 Averaging Methods

Of the wide variety of smoothing algorithms, the simplest and most straight forward is Laplacian smoothing[87]. With this method, an internal node in the mesh is placed at the average location of any node connected to it by an edge. With little modification, this technique can be applicable for any element shape. Most smoothing procedures will iterate through all the internal nodes in the mesh several times until any individual node has not moved more than a specified tolerance. Although it has its problems, it is simple to implement and is in wide use. Similar to Laplacian, there are a variety of other smoothing techniques, which iteratively reposition nodes based on a weighted average of the geometric properties of the surrounding nodes and elements. Canann[88] provides an overview of some of the common methods in use.

Averaging methods quite often also employ some form of additional constraint on the movement of a node. For example, because Laplacian smoothing alone sometimes has the tendency to invert or degrade the local element quality, a comparison of local element quality is made before and after the proposed move and the node moved only if element quality is improved. This is often referred to as *constrained Laplacian smoothing*. Canann[88] presents criteria for the movement of the node with this method.

5.1.2 Optimization-Based Methods

Rather than relying on heuristic averaging methods, some codes use optimization techniques to improve element quality. Optimization-based smoothing techniques measure the quality of the surrounding elements to a node and attempt to optimize by computing the local gradient of the element quality with respect to the node location. The node is moved in the direction of the increasing gradient until an optimum is reached. Canann[88] and Freitag[89] both present optimization-based smoothing algorithms.

While maintaining that optimization-based smoothing techniques provide superior mesh quality, the computational time involved is generally too excessive to use in standard practice. Canann[88] and Freitag[90] both recommend a combined Laplacian/optimzation-based approach. What is generally advocated is that Laplacian smoothing is done for the majority of the time, reverting to optimization based smoothing only when local element shape metrics drop below a certain threshold.

5.1.3 Physically-Based Methods

Another important area of mesh improvement includes methods that reposition nodes based on a simulated physically based attraction or repulsion force. Lohner[91] simulates the force between neighboring nodes as a system of springs interacting with each other. Shimada[92] and Bossen[93] view the nodes as the center of bubbles that are repositioned to attain equilibrium. With changes in the magnitude and direction of inter-particle forces, different anisotropic characteristics and element sizes can be achieved.

5.1.4 Mid-node Placement

While most smoothing methods focus on repositioning corner nodes, Salem[94] recently introduced a method providing criteria for repositioning mid-nodes on quadratic elements to improve element quality. This

method computes a region surrounding the mid-node known as the mid-node admissible space (MAS), shown in Figure 12, where the mid-node can safely be moved and maintain or improve element quality.

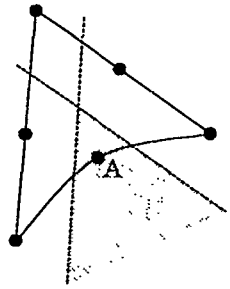


Figure 12. Mid-node admissible space for node at A

5.2 Cleanup

Like smoothing, there are a wide variety of methods currently employed to improve the quality of the mesh by making local changes to the element connectivities. Cleanup methods generally apply some criteria that must be met in order to perform a local operation. The criteria in general can be defined as: 1) shape improvement or 2) topological improvement.

In addition, cleanup operations are generally not done alone, but are used in conjunction with smoothing. Freitag[95] describes how smoothing and cleanup may be combined to efficiently improve overall element quality.

5.2.1 Shape improvement

For triangle meshes, simple diagonal swaps are often performed. For each interior edge in the triangulation a check can be made to determine at what position the edge would effectively improve the overall or minimum shape metric of its two adjacent triangles. The Delaunay criteria can also be used to determine the position of an edge. For Tetrahedral meshes, Barry Joe[96] presents a series of local transformations that are designed to improve the element quality. These include swapping two adjacent interior tets sharing the same face for three tets (see Figure 3). Likewise, three tets can be replaced with two. Other more complex transformations are also defined.

In some applications where mixed element meshes are supported, the element quality of two adjacent triangles may be preferable to a single poor quality quadrilateral. When this is the case, selected quadrilaterals may be split.

In some cases, particularly with curved surfaces, the elements resulting from the mesh generator may deviate significantly from the underlying geometry. For a triangle mesh, edge swaps can be performed based on which local position of the edge will deviate least from the surface. Although not strictly a cleanup operation, local refinement of the mesh may also be considered to capture surface features.

5.2.2 Topological Improvement

A common method for improving meshes is to attempt to optimize the number of edges sharing a single node. This is sometimes referred to as node *valence* or *degree*. In doing so, it is assumed that the local element shapes will improve. For a triangle mesh there should optimally be 6 edges at a node and four edges at a node surrounded by quads. Whenever there is a node that does not have an ideal valence, the quality of the elements surrounding it will also be less than optimal. Performing local transformations to the elements can improve topology and hence element quality. Several methods have been proposed for improving node valence for both triangle[97] and quadrilateral[98,99] meshes.

For volumetric meshes, valence optimization becomes more complex. In addition to optimizing the number of edges at a node, the number of faces at an edge can also be considered. For tetrahedral meshes this can involve a complex series of local transformations. For hexahedral elements, valence optimization is generally not considered tractable. The reason for this is that local modifications to a hex mesh will typically propagate themselves to more than the immediate vicinity. One special case of cleanup in hex meshes used in conjunction with the whisker weaving algorithm is presented by Mitchell[100].

5.3 Refinement

Element refinement procedures are numerous. For our purposes, refinement is defined as any operation performed on the mesh that effectively reduces the local element size. The reduction in size may be required in order to capture a local physical phenomenon, or it may be done simply to improve the local element quality. Some refinement methods in themselves can be considered mesh generation algorithms. Starting with a coarse mesh, a refinement procedure can be applied until the desired nodal density has been achieved. Quite frequently, refinement algorithms are used as part of an adaptive solution process, where the results from a previous solution provide criteria for mesh refinement. Methods have been proposed for triangle and tet refinement as well as quad and hex.

5.3.1 Triangle/Tetrahedral Refinement

Although there are certainly more methods defined, three of the principal methods for triangle and tetrahedral refinement include:

1. Edge bisection
2. Point insertion
3. Templates

5.3.1.1 Edge Bisection.

Edge bisection involves splitting individual edges in the triangulation. As a result, the two triangles adjacent the edge are split into two. Extended to volumetric meshing, any tetrahedron sharing the edge to be split must also be split as illustrated in Figure 13. Rivara[101] proposes criteria for the splitting of edges based on the longest edge of a triangle or tetrahedron.

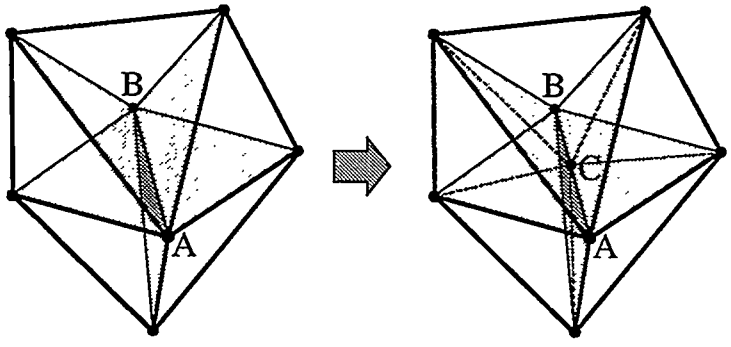


Figure 13. Edge bisection in a tetrahedral mesh. Edge A-B is split at point C, also splitting its surrounding tetrahedra.

5.3.1.2 Point Insertion

A simple approach to refinement is to insert a single node at the centroid of an existing element, dividing the triangle into three or tetrahedron into four. This method does not generally provide good quality elements, particularly after several iterations of the scheme. To improve upon the scheme, a Delaunay approach can be used that will delete the local triangles or tetrahedra and connect the node to the

triangulation maintaining the Delaunay criterion. Any of the Delaunay point insertion methods discussed previously could effectively be used for refinement.

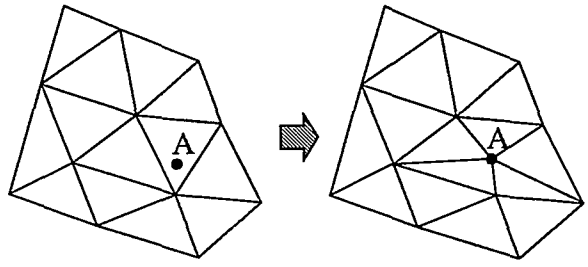


Figure 14. Example of Delaunay refinement, where point A is inserted.

5.3.1.3 Templates

A template refers to a specific decomposition of the triangle. One example is to decompose a single triangle into four similar triangles by inserting a new node at each of its edges as show in Figure 15. The equivalent tetrahedron template would decompose it into eight tetrahedra where each face of the tet has been decomposed into 4 similar triangles. To maintain a conforming mesh, additional templates can also be defined based on the number of edges that have been split. Staten[102] outlines the various templates needed to locally refine tetrahedra while maintaining a conforming mesh.

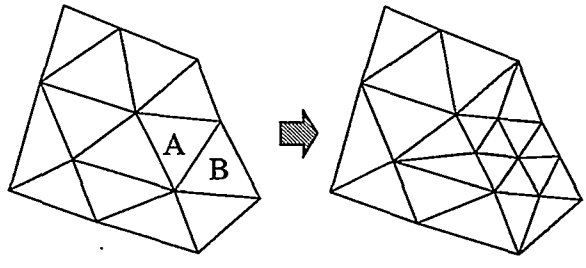


Figure 15. Example of local triangle refinement using a template where elements at A and B are refined

5.3.2 Quad/Hex Refinement

Because of the structured nature of quad and hex meshes, the point insertion and edge bisection methods are generally not applicable. The main methods used for quad and hex refinement involve decomposing the elements based on a set of predefined templates. Both Schneiders[103] and Staten[98] propose algorithms and a series of templates for element decomposition. An example of local quad refinement is shown in Figure 16. In order to maintain a conforming mesh, some quad and hex refinement schemes will often necessarily introduce triangle or alternate shaped elements including tetrahedra and pentahedra.

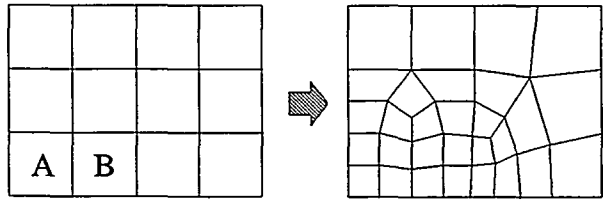


Figure 16. Example of local quad refinement where elements at A and B are refined by one half.

6. Conclusion

This survey has touched only briefly on some of the main issues and algorithms used in unstructured mesh generation. There are many more important aspects of unstructured mesh generation that were not addressed. Due to time and space constraints, it was not intended to be a comprehensive overview of the subject. Instead, it was the intent to focus on some of the more fundamental algorithms and procedures. Often times in the research and development of software, we tend to forget what has gone before us, or fail to look at what is already readily available. The most efficient way to provide new and innovative technology is to build on the accomplishments of others. We should recognize the innovations and creativity of others in the field and try to improve upon what has gone before.

References

1 Steven J. Owen, (1998) Meshing Software Survey, *web page:* http://www.andrew.cmu.edu/user/sowen/softsurv.html

2 Joe F. Thompson, (1985) "Numerical Grid Generation, Foundation and Applications", Elsevier. Posted on www at http://www.erc.msstate.edu/education/gridbook/index.html

3 Joe F. Thompson, (1996) "A Reflection on Grid generation in the 90s: Trends Needs and Influences", *5th International Conference on Numerical Grid Generation in Computational Field Simulations, Mississippi State University, April 1996.* pp.1029-1110

4 Steven J. Owen, Meshing Software Survey, Structured Grid Generation Software, *web page:* http://www.andrew.cmu.edu/user/sowen/software/structured.html

5 Mark A.Yerry and Mark S, Shephard, (1984) "Three-Dimensional Mesh Generation by Modified Octree Technique", International Journal for Numerical Methods in Engineering, vol 20, pp.1965-1990

6 Mark S. Shephard and Marcel K. Georges, (1991) "Three-Dimensional Mesh Generation by Finite Octree Technique", *International Journal for Numerical Methods in Engineering*, vol 32, pp. 709-749

7 Scientific Computation Research Center (SCOREC), Rensselaer Polytechnic Institute, *web site:* http://www.scorec.rpi.edu/

8 Stephen A. Vavasis, QMG *web site:* http://simon.cs.cornell.edu/Info/People/vavasis/qmg-home.html

9 Boris, N. Delaunay, (1934) "Sur la Sphere" *Vide. Izvestia Akademia Nauk SSSR*, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk Vol 7 pp.793-800

10 C. L. Lawson, (1977) "Software for C^1 Surface Interpolation", *Mathematical Software III*, pp.161-194

11 David F. Watson, (1981) "Computing the Delaunay Tesselation with Application to Voronoi Polytopes", *The Computer Journal*, Vol 24(2) pp.167-172

12 Timothy J. Baker, (1989) "Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation", *Engineering with Computers*, vol 5, pp.161-175

13 N. P. Weatherill and O. Hassan (1994) "Efficient Three-dimensional Delaunay Triangulation with Automatic Point Creation and Imposed Boundary Constraints", *International Journal for Numerical Methods in Engineering*, vol 37, pp.2005-2039

14 P.L. George, F. Hecht and E. Saltel (1991) "Automatic Mesh Generator with Specified Boundary", *Computer Methods in Applied Mechanics and Engineering*, North-Holland, vol 92, pp.269-288

15 Paul L. Chew, (1989) "Guaranteed-Quality Triangular Meshes", *TR 89-983*, Department of Computer Science, Cornell University, Ithaca, NY, April 1989

16 Jim Ruppert, (1992) "A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation". Technical Report UCB/CSD 92/694, University of California at Berkely, Berkely California

17 Jonathan Richard Shewchuk, (1996) "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator", http://www.cs.cmu.edu/~quake/triangle.html , 1996

18 S. Rebay, (1993) "Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm", *Journal Of Computational Physics*, vol. 106, pp.125-138

19 David L. Marcum and Nigel P. Weatherill, "Unstructured Grid Generation Using Iterative Point Insertion and Local Reconnection", *AIAA Journal*, vol 33, no. 9, pp.1619-1625, September 1995

20 David L. Marcum Solidmesh *web site:* http://www.erc.msstate.edu/thrusts/grid/solid_mesh/

21 H. Borouchaki, F. Hecht, E. Saltel and P. L. George, "Reasonably Efficient Delaunay Based Mesh Generator in 3 Dimensions", *Proceedings 4th International Meshing Roundtable*, pp.3-14, October 1995

22 TetMesh, GSH3D *web site:* http://www.simulog.fr/tetmesh/

23 P.L. George, , F. Hecht and E. Saltel, (1991) "Automatic Mesh Generator with Specified Boundary", *Computer Methods in Applied Mechanics and Engineering*, vol 92, pp.269-288

24 B. Joe, (1992) "Three-dimensional boundary-constrained triangulations", Artificial Intelligence, Expert Systems, and Symbolic Computing -- Proceedings of the 13th IMACS World Congress, ed. E. N. Houstis and J. R. Rice, Elsevier Science Publishers, pp. 215-222.

25 B. Joe, (1991) "GEOMPACK - A Software Package for the Generation of Meshes Using Geometric Algorithms", *Advances in Engineering Software*, vol 56, no. 13, pp.325-331

26 B. Joe, GEOMPACK, *anonymous ftp:* ftp://ftp.cs.ualberta.ca/pub/geompack

27 Rainald Lohner, Paresh Parikh and Clyde Gumbert, (1988) "Interactive Generation of Unstructured Grid for Three Dimensional Problems", *Numerical Grid Generation in Computational Fluid Mechanics '88*, Pineridge Press, pp.687-697

28 R. Lohner, (1996) "Progress in Grid Generation via the Advancing Front Technique", *Engineering with Computers*, vol 12, pp.186-210

29 S. H. Lo, (1991) "Volume Discretization into Tetrahedra-I. Verification and Orientation of Boundary Surfaces", *Computers and Structures*, vol 39, no. 5, pp.493-500

30 S. H. Lo, (1991) "Volume Discretization into Tetrahedra - II. 3D Triangulation by Advancing Front Approach", *Computers and Structures*, vol 39, no 5, pp.501-511

31 ANSYS *web site:* http://www.ansys.com

32 Shahyar Pirzadeh, (1993) "Unstructured Viscous Grid Generation by Advancing-Layers Method", AIAA-93-3453-CP, AIAA, pp.420-434

33 TetrUSS, Tetrahedral Unstructured Software System (includes VGRID mesh generator), *web site:* http://ad-www.larc.nasa.gov/tsab/tetruss/

34 Steven J. Owen, Meshing Software Survey, Quadrilateral Mesh Generation Software: *web page:* http://www.andrew.cmu.edu/user/sowen/software/quadrilateral.html

35 Steven J. Owen, Meshing Software Survey, Hexahedra Mesh Generation Software: *web page:* http://www.andrew.cmu.edu/user/sowen/software/quadrilateral.html hexahedra.html

36 W.A. Cook, and W.R. Oakes (1982). "Mapping Methods for Generating Three-Dimensional Meshes", *Computers in Mechanical Engineering*, August 1982, pp. 67-72

37 CUBIT Mesh Generation Toolkit, *web site:* http://endo.sandia.gov/SEACAS/CUBIT/Cubit.html

38 Timothy J. Tautges, , Shang-sheng Liu, Yong Lu, Jason Kraftcheck, Rajit Gadh, (1997) "Feature Recognition Applications in Mesh Generation", *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, ASME, pp.117-121

39 Shang-Sheng Liu, and Rajit Gadh, (1996) "Basic LOgical Bulk Shapes (BLOBS) for Finite Element Hexahedral Mesh Generation", *5th International Meshing Roundtable*, pp.291-306

40 Scott A. Mitchell, (1997) "High Fidelity Interval Assignment", *Proceedings, 6th International Meshing Roundtable*, pp.33-44

41 David R. White, (1995). "Automated Hexahedral Mesh Generation by Virtual Decomposition", *Proceedings, 4th International Meshing Roundtable*, Sandia National Laboratories, pp.165-176

42 Matthew L. Staten, Scott A. Canann, and Steve J. Owen (1998) "BMSWEEP: Locating Interior Nodes During Sweeping", *7th International Meshing Roundtable*

43 Mingwu, Lai, Steven E. Benzley, Greg Sjaardema and Tim Tautges (1998) "A Multiple Source and Target Sweeping Method for Generating All-Hexahedral Finite Element Meshes", *5th International Meshing Roundtable*, pp.217-228

44 Ted D. Blacker, (1996). "The Cooper Tool", *Proceedings, 5th International Meshing Roundtable*, pp.13-29

45 Fluent, Gambit *web site:* http://www.fluent.com/software/gambit/gambit.htm

46 S.H. Lo, (1989). "Generating Quadrilateral Elements on Plane and Over Curved Surfaces", *Computers and Structures*, Vol.31(3), pp.421-426

47 Bruce P Johnston, John M. Sullivan Jr. and Andrew Kwasnik (1991). "Automatic Conversion of Triangular Finite Element Meshes to Quadrilateral Elements", *International Journal for Numerical Methods in Engineering*, Vol.31, pp.67-84

48 C.K Lee, and S.H. Lo (1994). "A New Scheme for the Generation of a Graded Quadrilateral Mesh," *Computers and Structures*, Vol.52 pp.847-857

49 Steven J. Owen, Matthew L. Staten, Scott A. Canann and Sunil Saigal, (1998) "Advancing Front Quad Meshing Using Local Triangle Transformations", *Proceedings, 7th International Meshing Roundtable*

50 Peggy L. Baehmann, Scott L. Wittchen, Mark S. Shephard, Kurt R. Grice and Mark A. Yerry, (1987). "Robust Geometrically-based, Automatic Two-Dimensional Mesh Generation," *International Journal for Numerical Methods in Engineering*, Vol.24, pp.1043-1078

51 J.A. Talbert, and A.R. Parkinson, (1991). "Development of an Automatic, Two Dimensional Finite Element Mesh Generator using Quadrilateral Elements and Bezier Curve Boundary Definitions", *International Journal for Numerical Methods in Engineering*, Vol.29 pp.1551-1567

52 Soo-Won Chae, and Jung-Hwan Jeong, (1997). "Unstructured Surface Meshing Using Operators", *Proceedings, 6th International Meshing Roundtable*, pp.281-291

53 Dietrich, Nowottny, (1997). "Quadrilateral Mesh Generation via Geometrically Optimized Domain Decomposition", *Proceedings, 6th International Meshing Roundtable*, pp.309-320

54 T. K. H. Tam and C. G. Armstrong (1991). "2D Finite Element Mesh Generation by Medial Axis Subdivision", *Advances in Engineering Software*, Vol.13, pp.313-324

55 Barry Joe, (1995). "Quadrilateral Mesh Generation in Polygonal Regions", *Computer Aided Design*, Vol.27, pp.209-222

56 Barry Joe, (1986). "Delaunay Triangular Meshes in Convex polygons", *SIAM J. Sci. Stat. Comput.*, Vol.7, pp.514-539

57 J.Z. Zhu, O.C. Zienkiewicz, E. Hinton and J. Wu (1991). "A New Approach to the Development of Automatic Quadrilateral Mesh Generation," ", *International Journal for Numerical Methods in Engineering*, Vol.32 pp.849-866

58 Ted D. Blacker, and Michael B. Stephenson (1991). "Paving: A New Approach to Automated Quadrilateral Mesh Generation", *International Journal for Numerical Methods in Engineering*, Vol 32 pp.811-847

59 Roger J. Cass, , Steven E. Benzley, Ray J. Meyers and Ted D. Blacker (1996). "Generalized 3-D Paving: An Automated Quadrilateral Surface Mesh Generation Algorithm", *International Journal for Numerical Methods in Engineering*, Vol. 39 pp.1475-1489

60 David R. White and Paul Kinney (1997). "Redesign of the Paving Algorithm: Robustness Enhancements through Element by Element Meshing," *Proceedings, 6th International Meshing Roundtable*, Sandia National Laboratories, pp. 323-335

61 MacNeal-Schwendler Home Page, *web site:* http://www.macsch.com/

62 Takeo Taniguchi, Tomoaki Goda, Harald Kasper and Werner Zielke, (1996) "Hexahedral Mesh Generation of Complex Composite Domain", *5th International Conference on Grid Generation in Computational Field Simmulations*, Mississippi State University. pp 699-707

63 Robert Schneiders, (1996) "A Grid-Based Algorithm for the Generation of Hexahedral Element Meshes", *Engineering With Computers.* Vol.12 pp.168-177

64 P. Weiler, R. Schindler and R. Schneiders, (1996) "Automatic Geometry-Adaptive Generation of Quadrilateral and Hexahedral Element Meshes for the FEM", *Proceedings, 5th International Conference on Numerical Grid Generation in Computational Field Simmulations*, Mississippi State University, pp.689-697

65 Robert Schneiders, (1997) "An Algorithm for the Generation of Hexahedral Element Meshes Based On An Octree Technique", *Proceedings, 6th International Meshing Roundtable*, Abstract only pp.195-196

66 Monika Wierse, Jean Cabello and Yoshihiko Mochizuki, (1998) "Automatic Grid Generation with HEXAR", *Proceedings 6th International Conference on Numerical Grid Generation in Computational Field Simulations, ed. M. Cross et. al.*, University of Greenwich, UK., pp. 843-852

67 MARC *web site*: http://toto.marc.com/

68 T.S. Li, R.M. McKeag and C.G. Armstrong, (1995) "Hexahedral Meshing Using Midpoint Subdivision and Integer Programming", *Computer Methods in Applied Mechanics and Engineering*, Vol.124, pp.171-193

69 M.A. Price and C.G. Armstrong, (1995) "Hexahedral Mesh Generation by Medial Surface Subdivision: Part I", *International Journal for Numerical Methods in Engineering*. Vol 38(19), pp.3335-3359

70 M.A. Price and C.G. Armstrong, (1997) "Hexahedral Mesh Generation by Medial Surface Subdivision: Part II," *International Journal for Numerical Methods in Engineering*. Vol 40, pp.111-136

71 FEGS *web site:* http://fegs.co.uk

72 Solidpoint *web site:* http://www.99main.com/~diholm/

73 Scott A. Canann, (1991) "Plastering and Optismoothing: New Approaches to Automated, 3D Hexahedral Mesh Generation and Mesh Smoothing," Ph.D. Dissertation, Brigham Young University, Provo, UT.

74 Ted D. Blacker and R. J. Myers, (1993). "Seams and Wedges in Plastering: A 3D Hexahedral Mesh Generation Algorithm," Engineering With Computers, Vol.2, pp.83-93

75 Timothy J. Tautges, Ted Blacker and Scott Mitchell, (1996) "The Whisker-Weaving Algorithm: A Connectivity Based Method for Constructing All-Hexahedral Finite Element Meshes," *International Journal for Numerical Methods in Engineering*, Vol.39, pp.3327-3349

76 Peter Murdoch, and Steven E. Benzley, (1995) "The Spatial Twist Continuum", *Proceedings, 4th International Meshing Roundtable*, Sandia National Laboratories, pp.243-251

77 Steven J. Owen, Scott A. Canann and Sunil Saigal, (1997) "Pyramid Elements for Maintaining Tetrahedra to Hexahedra Conformability", *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, ASME, pp.123-129

78 Phillip Tuchinsky, M., Brett W. Clark, (1997) "The Hex-Tet, Hex-Dominant Automesher: An Interim Progress Report", *Proceedings, 6th International Meshing Roundtable*, pp.183-193

79 Weidong Min, (1997) "Generating Hexahedron-Dominant Mesh Based on Shrinking-Mapping Method", *Proceedings, 6th International Meshing Roundtable*, pp.171-182

80 R. T. Farouki, (1997) "Optimal paramaterizations," *Comuter Aided Geometric Design*, vol. 14 153-168

81 Paul-Louis George, and Houman Borouchaki (1998) *Delaunay Triangulation and Meshing: Application to Finite Elements*, Hermes, France, 413 p.

82 Hao Chen and Jonathan Bishop (1997) "Delaunay Triangulation for Curved Surfaces", *Proceedings, 6th International Meshing Roundtable*, pp.115-127

83 J. C. Cuilliere, (1998) "An adaptive method for the automatic triangulation of 3D parametric surfaces", *Computer-Aided Design*, vol 30, no. 2, pp.139-149

84 Joseph R. Tristano, Steven J. Owen and Scott A. Canann, (1998) "Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition", *7th International Meshing Roundtable*

85 Lau, T.S. and S.H. Lo, (1996) "Finite Element Mesh Generation Over Analytical Surfaces", *Computers and Structures*, vol 59, no. 2, pp.301-309

86 Lau, T.S., S. H. Lo and C. K. Lee, (1997) "Generation of Quadrilateral Mesh over Analytical Curved Surfaces", *Finite Elements in Analysis and Design*, vol 27, pp.251-272

87 Field, D. A.(1988), "Laplacian smoothing and Delaunay triangulations", *Commuications in Applied Numerical Methods.*, vol. 4, pp. 709-712.

88 Scott A. Canann, Joseph R. Tristano and Matthew L. Staten, (1998) "An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes," *Proceedings, 7th International Meshing Roundtable*

89 Lori Freitag, Mark Jones, and Paul Plassmann, (1995) "An Efficient Parallel Algorithm for Mesh Smoothing", *Proceedings, 4th International Meshing Roundtable*, pp.47-58

90 Lori A. Freitag, (1997) "On Combining Laplacian and Optimization-Based Mesh Smoothing Techniques", *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, pp.37-43

91 R. Lohner, K. Morgan and O. C. Zienkiewicz, (1986) "Adaptive Grid Refinement for Compressible Euler Equations", *Accuracy Estimates and Adaptive refinements in Finite Element Computations, I. Babuska et. al. eds.*, Wiley, pp. 281-297

92 Kenji Shimada, Atsushi Yamada and Takayuki Itoh, (1997) "Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles", *Proceedings, 6th International Meshing Roundtable*, pp.375-390

93 Frank, J Bossen and Paul S. Heckbert (1996) "A Pliant Method for Anisotropic Mesh Generation", *Proceedings, 5th International Meshing Roundtable*, pp.63-76

94 Ahmed Z.I.Salem, Scott A. Canann, and Sunil Saigal, (1997) "Robust Distortion Metric for Quadratic Triangular 2D Finite Elements", *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, pp.73-80

95 Freitag, Lori A. and Carl Ollivier-Gooch, (1997) "Tetrahedral Mesh Improvement Using Swapping and Smoothing", *International Journal for Numerical Methods in Engineering*, vol. 40, pp.3979-4002

96 Barry Joe, (1995) "Construction of Three-Dimensional Improved-Quality Triangulations Using Local Transformations", *Siam J. Sci. Comput.*, vol 16, pp.1292-1307

97 S. A. Canann, S. N. Muthukrishnan and R. K. Phillips (1996) "Topological Refinement Procedures for Triangular Finite Element Meshes", *Engineering with Computers*, vol 12, pp.243-255

98 Staten, Matthew L. and Scott A. Canann, (1997) "Post Refinement Element Shape Improvement for Quadrilateral Meshes", *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, pp.9-16

99 Paul Kinney, (1997) "CleanUp: Improving Quadrilateral Finite Element Meshes", *Proceedings, 6th International Meshing Roundtable*, pp.437-447

100 Scott A Mitchell and Timmothy J. Tautges, (1995) "Pillowing Doublets: Refining A Mesh to Ensure That Faces Share At Most One Edge", *Proceedings, 4th Internatinal Meshing Roundtable*, pp.231-240, October 1995

101 Rivara, Maria-Cecilia, (1997) "New Longest-Edge Algorithms For the Refinement and/or Improvement of Unstructured Triangulations", *International Journal for Numerical Methods in Engineering*, vol. 40, pp.3313-3324

102 Staten, M.L. and N.L. Jones (1997) "Local Refinement of Three-Dimensional Finite Element Meshes", *Engineering with Computers*, vol 13, pp.165-174

103 R. Schneiders, (1996) "Refining Quadrilateral and Hexahedral Element Meshes", *5th International Conference on Numerical Grid Generation in Computational Field Simulations*, Mississippi State University, pp.679-688

Appendix

Meshing Software Survey

A survey was conducted during September 1998 of current mesh and grid generation software. Over 100 surveys were mailed to software vendors, research labs and educational institutions. This list is definitely not all-inclusive, but I believe is fairly representative of what is currently available, as well as what is state-of-the-art. Only those responding to the survey are included here. While the emphasis of the survey is unstructured codes, there are also a considerable number of structured codes included. The codes range from simple research codes that are used only by a few people, to commercial products incorporated into sophisticated analysis packages. An online and up-to-date copy of this survey is available on the web as part of the *Meshing Research Corner* web site maintained at Carnegie Mellon University by the author at the following URL: http://www.andrew.cmu.edu/user/sowen/softsurv.html

Survey Statistics

Total number of software products in survey	81
Element Shapes	
Number of products that generate triangles	52
Number of products that generate quadrilaterals (non-structured codes)	25
Number of products that generate tetrahedra	39
Number of products that generate hexahedra (non-structured codes)	20
Number of products that generate structured quads or hexes	21
Availability	
Number of Public Domain Codes	34
Number of Research Codes	24
Number of Commercial Products	33
Number of Products Available as Stand-Alone Meshing Generator	40

Number of Products providing Source Code	21
Engineering Discipline	
Number of Products used for Structural Applications	23
Number of Products used for CFD (Fluids) Applications	47
Number of Products used for EMAG (Electro-magnetic) Applications	23
Number of Products used for Thermal Applications	9
Number of Products used for Environmental Applications	12
Tri/Tet Meshing Algorithm	
Number of tri/tet codes using some form of Delaunay Algorithm	37
Number of tri/tet codes using some form of Advancing Front Algorithm	23
Number of tri/tet codes using an Octree Algorithm	4
Quad/Hex Meshing Algorithm	
Number of quad/hex codes using an Advancing_Front Algorithm	9
Number of quad/hex codes using a Medial Axis/Surface Algorithm	2
Number of quad/hex codes using an indirect Algorithm (combine triangles)	5
Number of quad/hex codes using a Sweeping or Extrusion Algorithm	8
Number of quad/hex codes using a Mapped Meshing Algorithm	11
Other Features	
Number of Products providing Boundary Layer definition	17
Number of Products providing Adaptivity	18
Number of Products providing Anisotropy (stretched elements)	16
Number of Products providing Refinement	27
Number of Products providing Mesh Improvement	8

Software Products

The following is the complete list of software included in the survey ordered alphabetically by product name. Also included is a contact indvidual and web site. A separate web page for each product listing basic features and comments provided by the contact is also provided on-line.

2-D GWADAPT, University of Nevada Las Vegas/Nevada Center for Advanced Computational Methods (NCACM), Dr. Yitung Chen or Dr. Laxmi Gewali, nccm_www@aurora.nscee.edu, http://www.unlv.edu/Research_Centers/NCACM/

3DMAGGS (Three-Dimensional Multi-block Advanced Grid Generation System), NASA Langley Research Center/Lockheed Martin Engineering & Sciences, Stephen J. Alter, Charles G. Miller, s.j.alter@larc.nasa.gov, http://ab00.larc.nasa.gov/~salter/3DMAGGS.html

ADMesh, Varlog, Anthony D. Martin, amartin@varlog.com, http://www.varlog.com/products/admesh

AFLR2, Engineering Research Center for Computational Field Simulation, Mississippi State University, David L. Marcum, marcum@erc.msstate.edu,

AFLR3, Engineering Research Center for Computational Field Simulation, Mississippi State University, David L. Marcum, marcum@erc.msstate.edu, http://www.erc.msstate.edu/thrusts/grid/solid_mesh

Algor Finite Element and Event Simulation Software, Algor, Inc., Julie Halapchuk, Marketing Communications, info@algor.com, http://www.algor.com

Altair Hypermesh, Altair Computing, Inc., George Christ, gjc@altair.com, http://www.altair.com/Products/HyperMesh.html

AMESH - Multi-Region Finite Element Meshing for Casting Processes, EKK, Inc, shawnekk@mail.ic.net, ekk@mail.ic.net, http://ic.net/~ekk/amesh.htm

ANSYS, ANSYS, Inc., Local ANSYS Support Distributor, , http://www.ansys.com

Argus ONE (Argus Open Numerical Environments), Argus Interware, Inc., Joshua Margolin, margolinj@argusint.com, http://www.argusint.com http://www.argusint.com/MeshGeneration.html

AVL FAME, AVL LIST GmbH, Anton Plimon, Robert Schmitz (North America), ap@avl.com schmitz@avlna.com, http://www.avl.com/html/69.htm

BAMG, INRIA, Frédéric Hecht, Frederic.Hecht@inria.fr, http://www-rocq.inria.fr/gamma/cdrom/www/bamg/eng.htm

BL2D, INRIA Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex (France), Patrick Laug, Houman Borouchaki, Patrick.Laug@inria.fr Houman.Borouchaki@univ-troyes.fr, http://www-rocq.inria.fr/gamma/cdrom/www/bl2d/eng.htm

CADfix, FEGS Ltd., John Rawlinson, john.rawlinson@fegs.co.uk, http://www.fegs.co.uk/index.html http://fegs.co.uk

CAF2D / GENMESH, Yeungnam Univ., Dept. of Mechanical Engineering, CAF Lab or OnDemand Soft (venture company), Professor Jong-Youb Sah, jysah@ynucc.yeungnam.ac.kr, http://caflab.yeungnam.ac.kr/genmesh.html

CAGI, ERC, Mississippi State University, Bharat Soni, bsoni@erc.msstate.edu, http://www.erc.msstate.edu/thrusts/grid/cagi/index.html

Cart3D, NASA Ames Research Center, Michael J. Aftosmis, Cathy Pochel (licensing), aftosmis@nas.nasa.gov cpochel@mail.arc.nasa.gov, http://george.arc.nasa.gov/~aftosmis/cart3d/

CFD-GEOM, CFD Research Corporation, John Whitmire, jbw@cfdrc.com, http://www.cfdrc.com http://www.cfdrc.com/datab/Software/geom/cfdgeom.html

Chalmesh, Chalmers University of Technology, Dept. of Naval Arch. & Ocean Eng., Anders Petersson, andersp@na.chalmers.se, http://www.na.chalmers.se/~andersp/chalmesh.html

COG, WIAS Berlin, Ilja Schmelzer, schmelzer@wias-berlin.de, ftp://ftp.wias-berlin.de/pub/cog/index.html

CSCMDO, Computer Sciences Corporation/NASA LaRC GEOLAB, William T. Jones, w.t.jones@LaRC.nasa.gov, http://geolab.larc.nasa.gov/CSCMDO

CUBIT Mesh Generation Toolkit, Sandia National Laboratories, David R. White, drwhite@sandia.gov, http://endo.sandia.gov/SEACAS/CUBIT/Cubit.html

delaundo, ipol, Von Karman Institute, Brussels, Belgium, Jens-Dominik Müller, muller@comlab.ox.ac.uk, http://www.cerfacs.fr/~muller/grids.html http://www.comlab.ox.ac.uk/oucl/people/jens-dominik.muller.html

DesignSpace, ANSYS Inc., Local ANSYS Support Distributor, , http://www.designspace.com/

EasyMesh, University of Trieste, D.I.N.M.A, Bojan Niceno, niceno@wt.tn.tudelft.nl, http://www-dinma.univ.trieste.it/~nirftc/research/easymesh/

EMC2, INRIA, Frédéric Hecht and Eric Saltel, Frederic.Hecht@inria.fr, http://www-rocq.inria.fr/gamma/cdrom/www/emc2/eng.htm

FELISA, NASA and MIT, Karen Bibb, NASA Langley Research Center, k.l.bibb@larc.nasa.gov, http://ab00.larc.nasa.gov/~kbibb/felisa.html

FEMGV (Version 5.1-01), Femsys Ltd., Steve Attwood, Derek Styles, info@femsys.co.uk s.attwood@femsys.co.uk d.styles@femsys.co.uk, http://www.femsys.co.uk/

GENIE++, ERC, Mississippi State University, Professor Bharat K. Soni, bsoni@erc.msstate.edu, http://www.erc.msstate.edu/thrusts/grid/genie/index.html

GeoCad, Industrial Research Ltd. (NZ) / Geothermal Energy Research and Development (Japan), Dr Stephen P White, s.white@irl.cri.nz, http://tui.grace.cri.nz/~steve/

geomagic Wrap, Raindrop Geomagic, Inc., Ping Fu, Chantelle Hougland, inquiry@geomagic.com, http://www.geomagic.com/

Geompack, University of Alberta, Computer Science, Barry Joe, bjoe@netcom.ca, ftp://ftp.cs.ualberta.ca:/pub/geompack/

GMS (Groundwater Modeling System), Environmental Modeling Research Laboratory (formerly the ECGL), Norm Jones, njones@et.byu.edu, http://www.emrl.byu.edu/

GMSH, Ecole Polytechnique de Montreal & University of Liege, Jean-François Remacle, remacle@meca.polymtl.ca, http://www.meca.polymtl.ca/~remacle/Mesh.html

Gridgen, Pointwise, Inc, Rick Matus, gridgen@pointwise.com, http://www.pointwise.com/

Gridomatic, VKI, UCDavis, Cislunar Aerospace, Dave Banks, dbanks@cislunar.com, http://mae.engr.ucdavis.edu/CFD/dbanks/Hybrid/gridomatic.html

gridpak, Rutgers University, Kate Hedstrom, kate@ahab.rutgers.edu, http://marine.rutgers.edu/po/gridpak.html

GridPro/AZ-Manager 3000, CLE GmbH and PDC, New York, Dr. Jochem Hauser (CLE), Dr. Peter Eiseman (PDC), jh@cle.de, http://www.cle.de/cfd/products/GridPro/index.html

GridTool, GEOLAB at NASA Langley Reseach center, Pat Kerr, P.A.KERR@LaRC.NASA.GOV, http://geolab.larc.nasa.gov/GridTool/

GRUMMP, University of British Columbia, Carl Ollivier-Gooch, cfog@mech.ubc.ca, http://tetra.mech.ubc.ca/GRUMMP

GUM-B, Engineering Research Center, Miss. State, Mike Remotigue, remo@erc.msstate.edu, http://www.erc.msstate.edu/thrusts/grid/index.html

ICEM CFD, ICEM CFD Engineering, Kristian Debus, Support and Releases, debus@icemcfd.com, http://www.icemcfd.com/

Javamesh, University of Pittsburgh, Steven Lin, steven@leetide.net, http://www.steven.pop.net.tw/javamesh/

LaGriT (Los Alamos Gridding Toolbox), Los Alamos National Laboratory, Carl Gable or Denise George, gable@lanl.gov, dgeorge@lanl.gov, http://www.t12.lanl.gov/~lagrit/

MAFIA-M, CST, Marko Walter, info@cst.de, http://www.cst.de/

MEGA (Meshing Environment for Geometry-based Analysis), Scientific Computation Research Center, Rensselaer Polytechnic Institute, Mark Shephard, Shephard@scorec.rpi.edu, http://www.scorec.rpi.edu/

MegaCads (Multiblock-Elliptic-Grid-generation-And-CAD-System), DLR, Institute of Design Aerodynamics and MEGAFLOW project, Olaf Brodersen and Prof. Dr. Horst Körner, megacads@dlr.de Olaf.Brodersen@dlr.de, http://www.bs.dlr.de/sm/ea/Proj_MEGAFLOW/MegaCadsOverview.html

Mentat, MARC Analysis Research Corporation, Jon Bishop (Mentat Manager), jon@marc.com, http://toto.marc.com/

MESH, ISE Integrated Systems Engineering AG, Zurich, ISE support, support@ise.ch, http://www.ise.ch/mesh.htm

Mesh++, Center for Advanced Studies, Research and Development in Sardinia (CRS4), Gianluigi Zanetti, zag@crs4.it, http://www.crs4.it/Areas/cfd/GRID_GENERATION/link1.html

Mesh-Maker, Environment Centre, University of Leeds, Jason Lander, jason@lec.leeds.ac.uk, http://www.lec.leeds.ac.uk/%7Ejason/Mesh-Maker/

mesh2d, Scientific Compuattional Research Center, SCOREC, B. Kaan Karamete, kaan@scorec.rpi.edu, http://scorec.rpi.edu/~kaan/

MG (Mesh Generator), TeCGraf - The Computer Graphics Tecnology Group of PUC-Rio, Luiz Cristovão Gomes Coelho, lula@tecgraf.puc-rio.br, http://www.tecgraf.puc-rio.br/~lula/mg/mg.html

MTC, SCC/ CEMEF, Philippe DAVID, phdavid@scconsultants.com, http://www.scconsultants.com/

NETGEN, Institut of Analysis and Numerical Mathematics, Johannes Kepler University, Linz, Austria, Joachim Schöberl, joachim@numa.uni-linz.ac.at, http://nathan.numa.uni-linz.ac.at/netgen/usenetgen.html

OVERGRID, MCAT, Inc. at NASA Ames Research Center, William M. Chan, wchan@nas.nasa.gov, http://halfdome.arc.nasa.gov/cfd/CFD4/og_man.html

Overture, Centre for Applied Scientific Computing, Lawrence Livermore National Laboratory., Bill Henshaw, overture-support@c3serve.c3.lanl.gov, http://www.c3.lanl.gov/Overture

Preproc, Numerical Methods Laboratory, "POLITEHNICA" University of Bucharest, Tiberiu Chelcea, tibi@lmn.pub.ro, http://www.lmn.pub.ro/~tibi/mesh_gen/mesh_gen.html

PRISM, NASA Ames Research Center, Shishir Pandya, pandya@nas.nasa.gov, http://www.engr.ucdavis.edu/~spandya/prism.html

Qhull, The Geometry Center, University of Minneapolis, Brad Barber, bradb@geom.umn.edu, http://www.geom.umn.edu/locate/qhull

QMG, Cornell University, Stephen A. Vavasis, vavasis@cs.cornell.edu, http://simon.cs.cornell.edu/Info/People/vavasis/qmg-home.html

QUAD - GEN, Computational Mechanics Australia Pty. Ltd, Dr. Alexander Tsvelikh, Director, comecau@bigpond.com sashat@ozemail.com.au, http://www.ozemail.com.au/~sashat/ http://www.ozemail.com.au/~sashat/quadgen.htm

QuikGrid, Perspective Edge Software, John Coulthard, w.j.coulthard@ubc.ca, http://www.interchg.ubc.ca/coulthrd/pes.html

samm, adapco, Wayne R. Oaks, wayne@adapco.com, http://www.adapco.com/samm.html

SCP Grid Library, Scalable Concurrent Programming Laboratory, Syracuse University, Marc Rieffel, marc@scp.syr.edu, http://www.scp.syr.edu/~marc/grid

SD (Super Delaunay) librarySDI (Super Delaunay Indexed) library, David Kornmann, David Kornmann, david@iki.fi, http://www.iki.fi/~david

SKY/Mesh2, Skyblue Systems, James Joseph, sales@skybluesystems.com, http://skybluesystems.com/mesh2.htm

SolidMesh, Engineering Research Center for Computational Field Simulation, Mississippi State University, David L. Marcum, marcum@erc.msstate.edu, http://www.erc.msstate.edu/thrusts/grid/solid_mesh

TetMesh GHS3D, SIMULOG, Mark Loriot, loriot@simulog.fr, http://www.simulog.fr/tetmesh/

TIGER-II Turbomachinery Grid Generation System, Version 2.01, Catalpa Research, Inc., Dr. Alan M. Shih, shih@catalpa.net, http://www.catalpa.net/

TMG (triangular mesh generator), Istituto di Analisi Numerica (CNR) of Pavia, Dipartimento di Matematica, University of Milano, Maurizio Paolini, m.paolini@dmf.bs.unicatt.it, http://www.dmf.bs.unicatt.it/~paolini/tmg/

TOAST, University College London, Dr Martin Schweiger, Dr. Simon Arridge, martins@medphys.ucl.ac.uk S.Arridge@cs.ucl.ac.uk, http://www.medphys.ucl.ac.uk/toast/index.htm

Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator., Carnegie Mellon University, Jonathan Richard Shewchuk, jrs@cs.cmu.edu, http://www.cs.cmu.edu/~quake/triangle.html

TriGrid, Channel Consulting Ltd, Adrian Dolling, adolling@channel.bc.ca, http://www.channel.bc.ca

TrueGrid, XYZ Scientific Applications, Inc., Matthew Koebbe, Ph.D., xyz@netcom.com, http://www.truegrid.com/

TRUMPET, NASA Lewis Research Center, Philip C. E. Jorgenson, jorgenson@lerc.nasa.gov, http://www.lerc.nasa.gov/WWW/microbus/cese/jorgenson/jorgenson.html

TurboMesh, SolidPoint, David Holmes, diholm@99main.com, http://www.99main.com/~diholm/

VGM, NASA Langley Research Center/Lockheed Martin Engineering & Sciences, Stephen J Alter, Charles Miller, s.j.alter@larc.nasa.gov, http://ab00.larc.nasa.gov/~salter/VGM-web.html

VGRID, VGRIDns (Navier-Stokes version), NASA Langley Research Center, Shahyar Z. Pirzadeh, s.pirzadeh@larc.nasa.gov, http://ad-www.larc.nasa.gov/tsab/tetruss/

Xcog, Chalmers University of Technology, Dept. of Naval Arch. and Ocean Eng., Anders Petersson, andersp@na.chalmers.se, http://www.na.chalmers.se/~andersp/xcog/xcog.html

XGEN, Charles University, Prague, Pavel Solin, solin@karlin.mff.cuni.cz, http://www.karlin.mff.cuni.cz/katedry/knm/xgen/

Meshing Software Survey

Steven J. Owen

		Availability					Platform			Discipline				Elements						Surface Meshing	Tri/Tet Algorithm		Quad/Hex Algorithm				Other Features					Approx. Num. Users
Product Name	Company/Organization	Public Domain	Research Code	Commercial Product	Stand-Alone Mesher	Source Code	Customer Support	Windows 95/98/NT	Macintosh	UNIX	Structural	CFD	EMAG	Thermal	Environmental	Triangle	Quadrilateral	Tetrahedra	Hexahedra	Pyramid	Wedge	Surface Meshing	Delaunay	Advancing Front	Octree	Advancing Front	Medial Axis	Indirect	Sweeping	Mapped meshing	Structured	Boundary Layers	Adaptivity	Anisotropy	Refinement	Mesh Improvement	
2-D GWADAPT	UNLV		✓	✓	✓		✓		✓			✓				✓	✓						✓				✓					✓					1200
3DMAGGS	NASA, Langley & Lockheed	✓				✓	✓		✓		✓	✓	✓	✓	✓		✓		✓												✓						110
ADMesh	Varlog	✓			✓	✓	✓		✓							✓																					250
AFLR2	ERC, Miss. State		✓		✓					✓		✓			✓	✓	✓						✓	✓		✓						✓	✓	✓	✓	✓	50
AFLR3	ERC, Miss. State		✓	✓	✓		✓		✓		✓	✓	✓					✓		✓	✓			✓	✓								✓		✓		✓	1000-5000
Algor	Algor, Inc.			✓	✓		✓		✓		✓					✓	✓	✓	✓	✓	✓	✓		✓			✓											16,000
AMESH	EKK, Inc			✓	✓		✓			✓								✓	✓	✓	✓	✓							✓						✓			50
ANSYS	ANSYS, Inc.			✓			✓		✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓			✓		✓	✓	✓		✓	✓		✓	✓	5000
Argus ONE	Argus Interware, Inc.			✓			✓		✓	✓	✓				✓	✓	✓					✓	✓					✓							✓			1000s
AVL FAME	AVL LIST GmbH			✓			✓			✓		✓	✓					✓	✓	✓				✓	✓								✓					100
BAMG	INRIA		✓						✓	✓		✓	✓	✓		✓	✓						✓				✓						✓	✓	✓			20
BL2D	INRIA	✓	✓			✓				✓		✓	✓	✓		✓							✓										✓	✓				100
CADfix	FEGS Ltd.			✓			✓		✓	✓		✓		✓		✓	✓	✓	✓		✓	✓	✓				✓				✓							200-300
CAF2D/GENMES	Yeungnam Univ., KR	✓							✓			✓				✓	✓					✓	✓				✓											unknown
CAGI	ERC, Miss. State	✓								✓		✓					✓		✓			✓								✓				✓				20
Cart3D	NASA Ames	✓		✓	✓		✓			✓		✓	✓						✓			✓											✓	✓				45
CFD-GEOM	CFD Research Corp.			✓	✓		✓		✓	✓		✓				✓	✓	✓	✓		✓	✓	✓					✓										250+
Chalmesh	Chalmers U., Sweden	✓			✓	✓				✓		✓	✓						✓												✓		✓			✓		25
COG	WIAS Berlin	✓		✓	✓	✓	✓			✓						✓		✓				✓	✓										✓	✓				not given
CSCMDO	CSC/NASA LaRC	✓			✓					✓							✓					✓									✓							50
CUBIT	Sandia Nat. Lab		✓		✓		✓			✓	✓		✓	✓		✓	✓	✓	✓	✓			✓				✓			✓	✓							20+
delaundo	Von Karman Institute	✓			✓	✓						✓				✓							✓	✓									✓		✓			10
DesignSpace	ANSYS Inc.			✓			✓		✓			✓			✓			✓					✓											✓		✓		500

Meshing Software Survey

Steven J. Owen

		Availability					Platform			Discipline				Elements						Surface Meshing	Tri/Tet Algorithm			Quad/Hex Algorithm						Other Features					Approx. Num. Users
Product Name	Company/Organization	Public Domain	Research Code	Commercial Product	Stand-Alone Mesher	Source Code	Customer Support	Windows 95/98/NT	Macintosh	UNIX	Structural	CFD	EMAG	Thermal	Environmental	Triangle	Quadrilateral	Tetrahedra	Hexahedra	Pyramid	Wedge	Surface Meshing	Delaunay	Advancing Front	Octree	Advancing Front	Medial Axis	Indirect	Sweeping	Mapped meshing	Structured	Boundary Layers	Adaptivity	Anisotropy	Refinement	Mesh Improvement	
EasyMesh	U. of Trieste	☑				☑						☑	☑			☑							☑	☑													5000
EMC2	INRIA	☑	☑						☑	☑						☑	☑						☑							☑	☑						500
FELISA	NASA/MIT				☑		☑			☑		☑	☑			☑		☑			☑		☑	☑									☑				50
FEMGV	Femsys Ltd.			☑			☑	☑		☑	☑				☑	☑	☑	☑	☑		☑	☑	☑				☑			☑	☑						500
GENIE++	ERC, Miss. State	☑								☑		☑	☑				☑		☑			☑									☑						140
GeoCad	Industrial research			☑	☑		☑	☑		☑					☑		☑		☑												☑						30
geomagic Wrap	Raindrop Geomagic				☑		☑	☑		☑	☑					☑	☑	☑	☑			☑	☑							☑				☑			100-200
Geompack	U. of Alberta	☑	☑		☑	☑		☑	☑	☑	☑					☑		☑					☑														unknown
GMS	EMRC, BYU			☑			☑	☑		☑					☑	☑	☑	☑	☑	☑	☑		☑					☑						☑			2500
GMSH	E. C. Montreal & U. Liege	☑					☑	☑		☑	☑	☑	☑			☑	☑	☑	☑		☑	☑	☑				☑		☑		☑		☑		☑		30
Gridgen	Pointwise			☑	☑		☑	☑		☑		☑				☑	☑	☑	☑			☑	☑								☑						250
Gridomatic	VKI, UCDavis		☑							☑		☑				☑	☑						☑	☑			☑				☑	☑		☑			10-20
gridpak	Rutgers University		☑							☑							☑														☑						50-200
GridPro	CLE GmbH and PDC			☑			☑	☑		☑		☑			☑		☑		☑			☑									☑						unknown
GridTool	NASA Langley	☑					☑			☑		☑	☑			☑		☑				☑		☑									☑		☑		250
GRUMMP	U. of British Columbia	☑				☑	☑			☑						☑		☑																	☑		700
GUM-B	ERC, Miss. State		☑				☑			☑		☑					☑		☑			☑									☑						30+
Hypermesh	Altair			☑			☑	☑		☑	☑					☑	☑	☑	☑		☑	☑		☑			☑			☑	☑						5000
ICEM CFD	ICEM CFD			☑	☑		☑	☑		☑	☑	☑		☑	☑	☑	☑	☑	☑		☑	☑			☑						☑						2000
Javamesh	U. of Pittsburgh		☑			☑										☑						☑		☑													unknown
LaGriT	Los Alamos	☑	☑		☑		☑			☑					☑	☑	☑	☑	☑	☑	☑	☑		☑									☑				20
MAFIA-M	CST		☑	☑			☑	☑		☑			☑			☑	☑	☑	☑			☑															2000
MEGA	SCOREC, RPI						☑	☑		☑	☑	☑				☑		☑				☑	☑	☑	☑							☑		☑	☑		unknown

Meshing Software Survey

Steven J. Owen

		Availability					Platform			Discipline				Elements						Tri/Tet Algorithm			Quad/Hex Algorithm						Other Features					
Product Name	Company/Organization	Public Domain	Research Code	Commercial Product	Stand-Alone Mesher	Source Code	Customer Support	Windows 95/98/NT	Macintosh	UNIX	Structural	CFD	EMAG	Thermal	Environmental	Triangle	Quadrilateral	Tetrahedra	Hexahedra	Pyramid	Wedge	Surface Meshing	Delaunay	Advancing Front	Octree	Advancing Front	Medial Axis	Indirect	Sweeping	Mapped meshing	Structured	Boundary Layers	Adaptivity	Anisotropy	Refinement	Mesh Improvement	Approx. Num. Users
MegaCads	DLR, Inst. of Des. Aerodyn.					☑	☑			☑		☑						☑		☑		☑								☑		☑					20-30
Mentat	MARC			☑	☑		☑	☑		☑	☑	☑	☑	☑		☑	☑	☑	☑		☑	☑	☑	☑		☑											6000
MESH	ISE, Zurich			☑	☑		☑	☑		☑						☑		☑		☑	☑				☑							☑	☑	☑	☑		confid.
Mesh++	CRS4, Sardinia		☑							☑		☑				☑	☑	☑	☑			☑		☑					☑				☑				unknown
Mesh-Maker	Env. Ctr., U. of Leeds		☑			☑	☑			☑		☑			☑			☑																		1+
mesh2d	SCOREC, RPI	☑				☑	☑	☑	☑	☑		☑				☑							☑									☑					100
MG	TeCGraf	☑		☑	☑		☑	☑	☑	☑						☑	☑	☑	☑			☑	☑	☑					☑	☑							40
MTC	SCC/ CEMEF		☑	☑	☑		☑	☑		☑								☑				☑															5
NETGEN	J. Kepler U., Austria	☑		☑	☑		☑	☑		☑		☑	☑			☑		☑			☑	☑	☑	☑								☑			☑		65
OVERGRID	NASA, Ames		☑				☑			☑		☑					☑		☑			☑								☑							60
Overture	CASC, LLNL	☑			☑	☑	☑			☑		☑					☑					☑								☑							10
Preproc	Num. Meth. Lab., U. of Buch	☑				☑	☑	☑		☑			☑			☑							☑									☑					10-20
PRISM	NASA Ames		☑			☑						☑									☑															5
Qhull	Geom. Center, U. of Minnea	☑				☑	☑	☑	☑	☑						☑		☑					☑														1000s
QMG	Cornell University	☑			☑		☑	☑		☑						☑		☑							☑												100s
QUAD - GEN	Comp. Mech., Australia			☑	☑	☑	☑										☑																☑	☑			confid.
QuikGrid	Perspective Edge	☑			☑		☑	☑									☑													☑							100s
samm	adapco			☑			☑			☑		☑				☑	☑	☑	☑	☑	☑	☑															75
SCP Grid Library	Syracuse University	☑			☑		☐	☑		☑		☑	☑					☑																	☑		10
SD	David Kornmann			☑	☑		☐	☑		☑						☑							☑														10
SKY/Mesh2	Skyblue Systems			☑	☑		☑	☑								☑	☑					☑													☑		80
SolidMesh	ERC, Miss. State		☑		☑		☑			☑		☑	☑					☑		☑	☑	☑		☑													15-20
TetMesh GHS3D	SIMULOG			☑	☑		☑	☑		☑		☑	☑					☑					☑														1000s

Meshing Software Survey

Steven J. Owen

		Availability					Platform			Discipline				Elements						Tri/Tet Algorithm			Quad/Hex Algorithm					Other Features					
Product Name	Company/Organization	Public Domain	Research Code	Commercial Product	Stand-Alone Mesher	Source Code	Customer Support	Windows 95/98/NT	Macintosh	UNIX	Structural	CFD	EMAG	Thermal	Environmental	Triangle	Quadrilateral	Tetrahedra	Hexahedra	Pyramid	Wedge	Surface Meshing	Delaunay	Advancing Front	Octree	Advancing Front	Medial Axis	Indirect	Sweeping	Mapped meshing	Structured	Boundary Layers	Adaptivity	Anisotropy	Refinement	Mesh Improvement	Approx. Num. Users
TIGER II	Catalpa Research, Inc.	☐	☐	☑	☑	☐	☑	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	30
TMG	CNR, U. of Milano	☑	☑	☐	☐	☑	☐	☐	☐	☑	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	10
TOAST	U. C. of London	☑	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☑	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	50+
Triangle	CMU	☐	☑	☐	☐	☑	☐	☑	☑	☑	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	1000?
TriGrid	Channel Consulting Ltd	☐	☐	☐	☑	☐	☑	☑	☐	☑	☐	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	50-200
TrueGrid	XYZ Sci. Apps., Inc.	☐	☐	☑	☑	☐	☑	☑	☐	☑	☑	☑	☑	☑	☐	☑	☑	☑	☑	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☐	☑	☑	500
TRUMPET	NASA, Lewis	☑	☑	☐	☐	☐	☑	☑	☐	☑	☐	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	not given
TurboMesh	SolidPoint	☐	☐	☑	☑	☐	☑	☑	☐	☐	☑	☐	☐	☑	☐	☐	☑	☐	☑	☐	☐	☑	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	10
VGM	NASA, Langley & Lockheed	☑	☐	☐	☐	☑	☑	☑	☐	☑	☑	☑	☑	☑	☑	☐	☑	☐	☑	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☑	☑	☑	100
VGRID	NASA Langley	☑	☐	☐	☑	☐	☑	☐	☐	☑	☐	☑	☐	☐	☐	☑	☐	☑	☐	☐	☐	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☑	☑	☑	☑	☐	190
Xcog	Chalmers U., Sweden	☑	☐	☐	☑	☐	☐	☐	☐	☑	☐	☑	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☑	☑	☐	30
XGEN	Charles U., Prague	☑	☐	☐	☐	☐	☑	☐	☐	☑	☐	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	100

Structured Grid Generation Technology

Bharat Soni
Professor, Aerospace Engineering
Sr. CFD Lead ARL_ASC PET_MSRC
NSF Engineering Research Center
Mississippi StateUniversity
PO Box 6176
Mississippi State, MS 39762

e-mail: bsoni@erc.msstate.edu
Phone: (601) 325-2647

Abstract

An overview of the current state-of-the-art and state-of-the-practice of the structured grid generation technology will be provided. The structured grid strategies including attached and overlapping(Chimera) grids will be discussed with generation and adaptation methodologies and softwares. In particular, algebraic, elliptic and hyperbolic generation schemes will be described with grid quality and response time issues. The utilization of Computer Aided Geometry Design (CAGD) techniques in structured grid generation and in interfacing CAD systems will be described.CAGD and grid generation softwares available in public domain and offered by commercial vendors will be presented. Current research efforts in autoblocking and parallel adaptive/ moving grid techniques will be summarized.

Session 4
Plenary II

Generation of Tetrahedral Finite Element Meshes: Variational Delaunay Approach

Petr Krysl* Michael Ortiz†

Abstract. *The goal is to generate tetrahedral decomposition of a general solid body, whose surface is given as a collection of triangular facets. The principle idea is that a vertex set in general positions guarantees existence of a unique triangulation which satisfies the Delaunay empty-sphere property. (Algorithms for robust, parallel construction of such triangulations are available.) However, all of the input surface facets do not necessarily appear in such a triangulation. In order to represent the boundary of the solid, we iterate two operations, edge flip and edge split with the insertion of additional vertex, until all of the boundary facets are present in the tetrahedral mesh. The outcome of the vertex insertion is another triangulation of the input surfaces, but one which is represented as a subset of the tetrahedral faces.*

Keywords: Finite element method, tetrahedral mesh, boundary constraints, variational Delaunay

Introduction

We deal with a tetrahedral decomposition of arbitrarily complex solids for the purposes of finite element analysis. The volume triangulation is expected to respect all the constraining surfaces in the sense that they must be geometrically and topologically similar to collections of tetrahedron faces [1]. If the input surfaces themselves are given by triangulations, it may come natural to think of the initial triangulation of the surfaces as a mere hint to the volume mesher on how to triangulate the solid in order to approximate the bounding surfaces well. For example, consider the tetrahedral mesh of Figure 1. The bounding or interface surfaces were in this case defined by triangulations, but they represent more or less smooth surfaces of bones and soft tissues. In this case, there is no compelling reason to preserve the input triangulations in the tetrahedral finite element mesh, especially with respect to the poor quality of the input surface meshes. (The situation may be different in cases where the surface triangulation has been designed to preserve certain characteristics of the original surface, for example convexity in fluid dynamics meshes. Section 4 deals with this issue in more detail.)

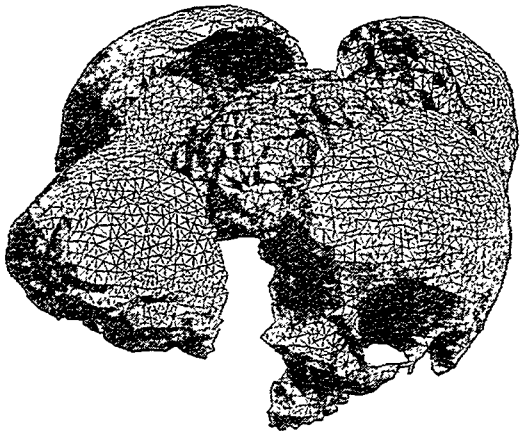


Figure 1: Mesh of cranium, cerebrospinal fluid and brain.

The principal idea of this work is actually quite well-aged. Hermeline seems to have been the first to propose incorporating boundaries of 3-D objects into unconstrained Delaunay triangulations [2]: Instead of trying to extract

*Staff scientist, California Institute of Technology, pkrysl@atlantis.caltech.edu
†Professor of Aeronautics and Applied Mechanics, California Institute of Technology, ortiz@atlantis.caltech.edu

the missing constraining facets a posteriori from a convex hull mesh, the input entities are complemented by additional vertices and facets which make an approximation (tesselation) of the constraining surfaces appear in the mesh ab initio. It is then easy matter either to construct only those tetrahedra which are in the region of interest, or to collect those tetrahedra from the convex hull mesh. Since then the idea re-surfaced in a number of papers; consult References [3, 4, 5, 6, 7]. Our approach differs from the preceding work in that we use the variational framework developed by Rajan [8] to determine if a facet of a constraining surface appears in the unconstrained Delaunay triangulation of the current set of vertices. If the facet is missing, we apply two local modification operations with the goal of improving sampling of the constraining surfaces. Decreased distances between surface vertices lead to the appearance of edges and facets. In difference to previous work we do not restrict the constraining surfaces in any way, in particular they may be non-planar, with multiple handles, and small input angles are allowed. However, this lack of restraint means that the technique currently comes without theoretical guarantees of termination.

1 Algorithm Overview

The main algorithmic steps are:

1. Read input and generate internal vertices. Randomly joggle vertex positions.
2. Determine if all constraining surface facets are represented by tetrahedron faces.
3. If there are any non-Delaunay facets, apply surface Delaunayzation algorithm.
4. Insert boundary facets into the advancing front and generate tetrahedra.
5. Remove slivers.

The input to the volume mesher consists of a list of vertices and a list of surface facets. Vertex data comprises a unique integer identifier, coordinates, and optionally mesh size at the vertex. Surface facets represent boundary or interface surfaces, and are specified by listing the three identifiers of its vertices given counterclockwise when looking against the "outer" normal of the surface. Each facet is associated with a unique surface. Furthermore, regions on each side of the facet are specified: The normal of the facet points into region r_1; region r_0 is on the other side. If the facet bounds just a single region (region r_1 is the semi-infinite space representing the "outside"), the facet is oriented such that its normal points out of the solid.

The locations of the input vertices are perturbed by a small random shift. By default the magnitude of the shift is 10^{-4} of the edge length (mesh size) at the vertex. For very closely spaced surfaces this could lead to (self)intersections, but such situations can be avoided by shifting the vertices in tangent planes only. The random perturbations yield vertex sets with a high probability of general vertex positions; see Reference [9] for a discussion.

2 Generation of Internal Vertices

The mesher has the ability to generate additional vertices "in the volume" of the solid. We adopt a simple two-stage technique. First, we generate vertices offset appropriately from the barycenters of boundary facets in an effort to generate good tetrahedra near the surfaces. Next, an octree is constructed which bounds the solid so that its leaves are of size proportional to the desired edge length at the centroid of the leaf, and the vertices are generated at the center and at the mid-points of the edges of the leaves so that for a uniform edge length specification one obtains arrangement in the form of an face-centered cubic crystal lattice [10]. The vertices are in both stages generated only if they do not fall too close to another vertex (which could have been given as input), and if they do not come too close to the surfaces of the solid. Note that it is not necessary to perform any expensive in/out tests to determine if the vertex is actually inside the solid, because the advancing front technique used to mesh the interior simply disregards vertices outside. The position of the generated vertices is randomly perturbed by a vector of magnitude of approximately 1% of the edge length.

3 Delaunayhood of Boundary Facets

How to determine if a boundary facet is represented in the (unconstrained) Delaunay mesh is crucial to our approach. The empty equatorial sphere criterion used by Miller et al. [5] and Shewchuk [7] is a sufficient, but not a necessary condition. Hence, one can hope to introduce less Steiner vertices with a less restrictive criterion. Our approach is

based on the optimality results for Delaunay triangulations advanced by Rajan [8]. Minimize function F defined on the convex hull of vertices P_i, $\mathrm{Conv}(P_i) \in R^n$ (in our case $n = 3$)

$$F(X, \lambda) = \sum_{i \in S_m} \lambda_i (P_i - X)^2 \tag{1}$$

subject to the constraints

$$\lambda_i \geq 0, \ \sum_{i=1}^{n} \lambda_i = 1, \ \sum_{i \in S_m} \lambda_i P_i = X \ . \tag{2}$$

The coefficients λ_i are recognized as the barycentric coordinates of the point X within the enclosing tetrahedron. The set S_m comprises all vertices in general. However, in order to make the technique computationally efficient, we build the set S_m adaptively as described below.

For the sake of brevity, we will denote facets represented by a tetrahedron face as *Delaunay* facets, and those that do not coincide with any face, *non-Delaunay* facets. In order to determine Delaunayhood of facet b we solve the linear programming problem (1) for a certain vertex set, S_m, at a point slightly offset against the direction of the facet normal (ie. into the solid). The distance is arbitrarily set at $K = 10^{-3}d$, where d is the characteristic dimension of the facet b. The vertex set S_m is initially chosen to include probable candidates for optimal solution of (1). Later, the set S_m is adjusted depending on the outcome of the solution of (1); see algorithm *TET_ON_FACET*. (The following notation is used in the psedo-code in this paper: $P_i(b)$ are the vertices of the facet b, $\widehat{P}(b)$ is the fourth vertex of the tetrahedron $T(b)$ piled up on a Delaunay facet b, CircS[T] is the circumsphere of tetrahedron T. The smallest, or equatorial, circumsphere of facet b is meant by CircS[b].)

The solution of (1) may be unbounded, in which case we assume there is no tetrahedron whose face represents the facet b. Another possibility is an infeasible solution (such as when the number of vertices is insufficient, or the vertices are all co-linear or co-planar). In that case the search region is inflated and the solution is re-tried. One can also arrive at a lower-dimensional solution, which happens when the sampling point X lies on an edge or on a face. In that case we remove the sampling point further from the facet and retry. Finally, the solution may give four non-zero lambdas in which case the circumsphere of the tetrahedron is checked if it is really empty as it should be (note that the vertex set used in the linear programming problem was not necessarily complete). If there is any vertex inside CircS[$T(b)$], it is an indication that the solution of (1) should be re-tried with an expanded vertex set S_m.

```
Algorithm: TET_ON_FACET (Facet b): returns Tetrahedron
  Compute facet barycenter C and normal n
  Sampling point X ← C − Kn;
  Estimate CircS[T(b)];
  B ← box enclosing CircS[T(b)];
  while (TRUE) do
    Solve (1) for S_m = vertices inside B
    case solution type do
    UNBOUNDED:
      return NULL
    INFEASIBLE:
      Increase B and continue
    LOWER_DIM:
      X ← C − Kn; recompute B and continue;
    OPTIMAL:
      T(b) ← tetrahedron from non-zero λ_i;
      if (Is CircS[T(b)] empty?) then
        return T
      else
        if (Solution of (1) yielded T for the second time) then return NULL;
        else Increase B to cover CircS[T(b)] and continue; endif
      endif
    done
  done
```

To speed up the computations we use not only the linear programming formulation (1) (which in itself yields all the information one needs, but at a relatively high cost), but we also perform some less expensive circumsphere tests

first, which can save computational effort by enabling early decisions. In order to be able to use the circumsphere tests, we save for Delaunay facets the vertex which belongs to the tetrahedron whose face coincides with the facet. The algorithm *IS_DELAUNAY?* used to decide Delaunayhood of a facet is described next. Note that the algorithm attempts to figure out status of a facet either after the facet has been created, in which case it does not know the status of the facet, or after some vertex has been added to the domain, in which case it needs to check whether the status of a previously Delaunay facet has changed or not.

Algorithm: *IS_DELAUNAY?*(Facet b): returns TRUE or FALSE
if (Was b previously Delaunay?)
  if (Is CircS[$T(b)$] empty?) **then return** TRUE; **endif**
endif
if (Is CircS[b] empty?) **then return** TRUE;
else
  foreach vertex P found in CircS[b] **do**
    $\widehat{P}(b) \leftarrow P$;
    if (Is CircS[$T(b)$] empty?) **then return** TRUE; **endif**
  done
  Solve linear programing problem of Equation (1) $\rightarrow$ tetrahedron t
  if (Does b coincide with one face of tetrahedron t?) **then**
    return TRUE;
  endif
endif
return FALSE;

4 Surface Delaunayzation

If there are any non-Delaunay facets in the constraining surfaces, the algorithm *DELAUNAYZATION* attempts to modify the surface meshes. There are two, to a certain measure contradicting, goals: (i) Make the surface mesh as well-shaped as possible, or at least do not cause its quality to deteriorate; and (ii) Introduce as few additional boundary facets (and vertices) as possible. Introduction of additional vertices into the surface mesh in order to make all facets Delaunay is in general unavoidable. Unfortunately, additional vertex can not only make some facet(s) Delaunay, it may also make others non-Delaunay. Hence, it is quite difficult to devise an algorithm provably terminating for all possible inputs. For smooth surfaces, Amenta and Bern [11] have shown that sufficiently dense sampling (as measured by the local feature size) makes the surface to appear in the unconstrained Delaunay tetrahedral mesh constructed with the input vertices and the Voronoi poles. The idea is to achieve "tangency" of the tetrahedron circumsphere to the constraining surface at surface vertices. Related ideas applied to the construction of shape skeletons have been advanced by Turkiyyah et al. [12].

For non-smooth surfaces, namely piecewise linear complexes (all constraining polytopes – edges and facets, have linear geometry), sphere packing algorithm of Miller et al. [5] provably yields a boundary conforming Delaunay triangulation. The input angles between any two boundary polytopes are restricted to 90°. A variation of this algorithm has re-appeared in the thesis of Shewchuk [7] in the framework of a Delaunay refinement algorithm. The principal idea is to make the constraining polytopes appear by enforcing their minimal circumspheres empty of vertices starting with vertices (trivial), edges, and finally proceeding to enforce facets. Empty minimal circumsphere is a sufficient condition for an edge or a face to be present in the Delaunay triangulation, but it is not a necessary one.

Two primitive surface modification operations are adopted here: *flip* of diagonal in a quadrilateral represented by two adjacent triangles, and *split* of a common edge by insertion of a new vertex. The first operation is inspired by the diagonal-swapping 2-D triangulation algorithm, the second is an attempt to maintain surface triangulation quality while introducing a Steiner point. The flip does not create any new vertex which could make other facets non-Delaunay, and is therefore preferable to split. On the other hand, in certain situations surface triangulations have been designed to possess some desirable properties such as convexity. Consider for example a coarsely discretized, very strongly curved leading edge of an airfoil. Concavities introduced by flips would unacceptably distort the air flow. In this case we could prohibit flips and use only splits which guarantee preservation of shape.

The *DELAUNAYZATION* algorithm is described next. While there are any non-Delaunay facets, a pass over all the facets is made, flipping as many "diagonals" as possible. Next, each still non-Delaunay facet is split along its longest edge.

Algorithm: *DELAUNAYZATION*
while (Is there a non-Delaunay boundary facet?) **do**
  perform *FLIPS* (NULL)
  perform *SPLITS*
done

The *FLIPS* algorithm works on a list of non-Delaunay facet, which can consist either of facets connected to a newly introduced vertex, or of all facets in the surface meshes. The procedure tries to find an adjacent facet with which flip can be accomplished using the algorithm *FLIPPABLE_NEIGHBOR*.

Algorithm: *FLIPS* (Vertex V)
if (Is V NULL?) **then** List $L \leftarrow$ all facets in mesh
else List $L \leftarrow$ all facets connected to V
endif
foreach non-Delaunay $b \in L$ **do**
  Facet $b' \leftarrow$ *FLIPPABLE_NEIGHBOR* (b)
  Flip diagonal in quadrilateral $b + b'$
done

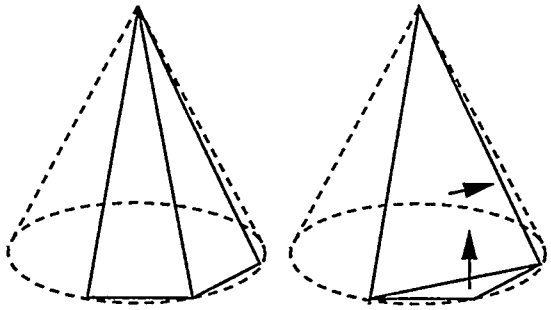


Figure 2: Prohibited flip leading to a sharp subtended angle. Left-hand side initial configuration; right-hand side situation resulting from flip.

The algorithm *FLIPPABLE_NEIGHBOR* assesses suitability of a facet adjacent to the facet b for flipping by checking (i) "convexity" of the quadrilateral composed of the two facets (convexity is judged in projection to an average plane using an area-based measure), (ii) angle subtended by the facets in their flipped connectivity (see Figure 2, where the flipped facets on the right-hand side subtend angle close to $\pi/2$), and (iii) result of the flip in terms of facet Delaunayhood and triangle quality measures (flip is rejected if no improvement is achieved).

Algorithm: *FLIPPABLE_NEIGHBOR* (Facet b): returns Facet
$L \leftarrow$ Order edges of b longest to shortest;
foreach edge $E \in L$ **do**
  Facet $b' \leftarrow$ neighbor across E
  if (Is quadrilateral $b + b'$ convex?
    and Do triangles after flip subtend reasonable angle?
    and (Does flip make b and b' Delaunay?
      or Are facets after flip of better triangle quality?)
) **then return** b'; **endif**
done
return NULL

The algorithm for facet splitting, *SPLITS*, introduces a new vertex at the mid-point of the longest edge. For acute triangles this means increase in the minimal circumradius, but this is most probably corrected immediately by flips attempted with all triangles connected to the new vertex.

Algorithm: *SPLITS*
List $L \leftarrow$ all non-Delaunay facets
foreach $b \in L$ **do**
  List $L' \leftarrow$ all facets across the longest edge of b
  perform *SPLIT_FACETS* (L')
  update Delaunayhood of all facets affected by the new vertex V
  perform *FLIPS* (V)
done

The procedure *SPLIT_FACETS* is rather straightforward, the only complication being that non-manifold situations need to be handled as illustrated in Figure 3.

Algorithm: *SPLIT_FACETS* (List L')
Insert new vertex V at the mid-point of edge common to facets in L'
foreach $b \in L'$ **do**
  replace b with new facets b' and b'' connected to V
done

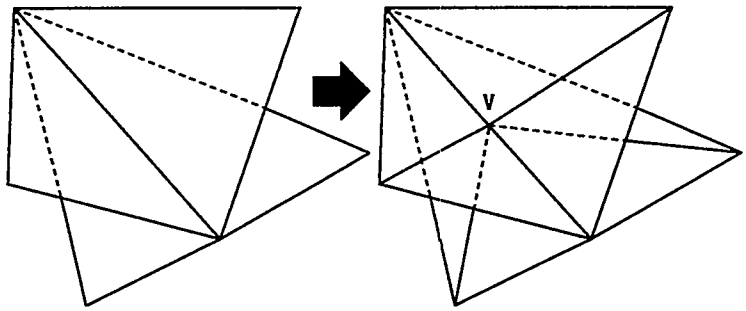


Figure 3: Splitting of facets sharing an edge.

5 Volume triangulation

Once the constraining facets are all made Delaunay, the tetrahedra can be generated by any unconstrained triangulation algorithm. We have chosen the advancing-front Delaunay [13, 14, 15, 10]. This technique is closely related to the original gift-wrapping algorithm [16] (also called incremental construction). Contrary to the classical advancing front algorithm [17], which generates nodes at the same time as tetrahedra, uses heuristics to compute the connectivity, and relies on intersection tests to ensure validity of the mesh, our implementation of the advancing front is based on the empty circumsphere property combined with the assumption of general vertex positions. The advancing-front Delaunay may not be the fastest serial algorithm, but it parallelizes easily [18].

6 Mesh Improvement

As is well known, creation of slivers (kites) cannot be avoided in Delaunay meshes constructed from pre-existing vertex sets in three dimensions. Since we use random perturbation of the vertex positions, the creation of slivers in the interior is unavoidable, and slivers also appear near the prescribed surfaces quite naturally. In order to reduce the number of slivers in the mesh, we modify the topology and geometry of the mesh in a post-processing step. Figure 4 shows three configurations of slivers with the adjacent tetrahedra. Slivers on the boundary can be deleted without difficulty (case A). Slivers in the interior can be deleted by swapping face Δdbe (case B), but are not removable in the case C (no two adjacent tetrahedra share a face). The non-removable slivers are opened up (expanded) by shifting slightly one of their vertices (the shift producing the largest minimal dihedral angle among all the connected tetrahedra is chosen). Further increase of the smallest dihedral angle can be achieved by insertion of additional vertices and local remeshing. (Not yet implemented.)

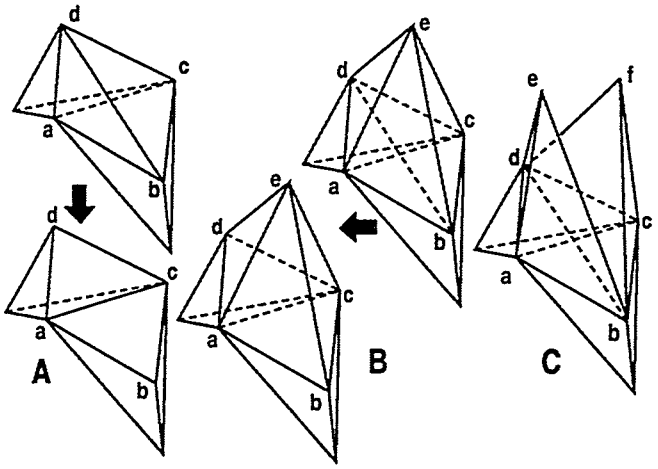
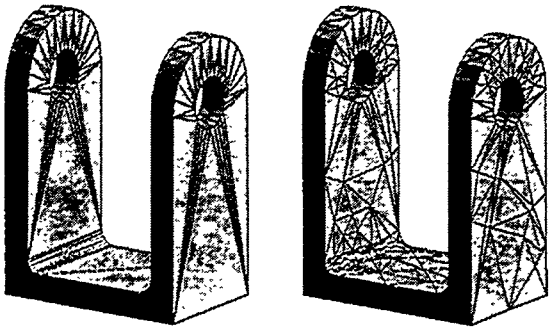


Figure 4: Removal of slivers.



(a) Input mesh (b) Output mesh

Figure 5: Mesh of a bracket.

7 Examples

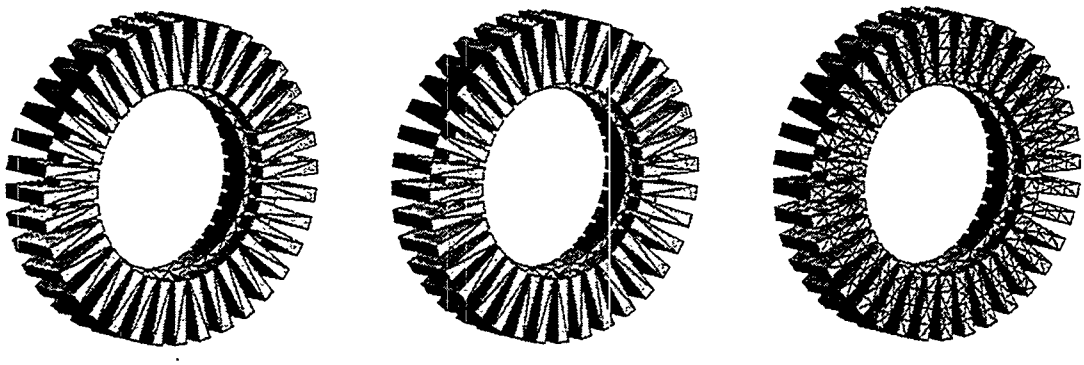
7.1 Mesh of Bracket

The first example illustrates the ability of the mesher to enforce the boundary constraint for a surface mesh consisting of very badly-shaped facets (Figure 5). The facets have been produced from a CSG model generated by the ACIS[1] geometry engine for the purpose of rendering with the default refinement settings. Hence the presence of needle-like and obtuse triangles. The input consisted of 180 vertices and 364 boundary facets (no vertices have been generated in the interior). The mesher inserted additional 55 vertices, increased the number of boundary facets to 474, and produced 460 tetrahedra in approximately 6.7 CPU seconds.[2]

7.2 Mesh of Wheel

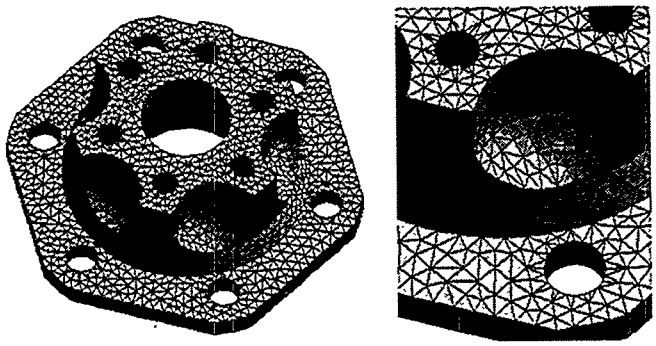
In this example we illustrate the effect of turning off the check relying on the solution of (1) (Figure 6). In other words, Delaunayhood of a facet is checked simply by the empty minimal circumsphere criterion. The input consisted of 624 points and 1248 boundary facets (no internal vertices have been generated). When the full check including

[1]Trademark of Spatial Technology Inc.
[2]The mesher has been run on an SGI Octane, with 195MHz R10K CPU. All timings in this section are given for this platform.



(a) Input mesh (b) Output mesh (c) Output mesh for (1) turned off

Figure 6: Mesh of a wheel.



(a) Input mesh (b) Detail

Figure 7: Mesh of a hub.

the linear programming problem of (1) was used, no additional vertices have been inserted, and the mesher produced 1378 tetrahedra (2.8 CPU seconds). When the check via (1) was turned off, the mesher added 219 surface vertices, 438 boundary facets, and produced 1810 tetrahedra (0.5 CPU seconds). The surface meshes are shown in Figure 6.

7.3 Mesh of Hub

In the next example we present timings for a series of uniform meshes for a mechanical part. The surface meshes have been again produced for rendering purposes and hence are not quite as good as can be expected from current finite element surface triangulation packages. Figure 7 shows the solid for the coarsest discretization (26722 elements); detail of the fan-like, acute triangles near the small holes is also included. The timings are given in Figure 8 with a breakdown into major steps: reading of input, generation of internal vertices, initial evaluation of Delaunayhood of boundary facets, Delaunayzation procedure, generation of tetrahedra by the advancing front, and finally, removal of slivers. The large amount of time spent in the generation of internal points is noteworthy (about 25% of total time). This step is costly mainly because of the need to verify that vertices are not generated too close to each other and to the constraining surfaces. In order to speed up this step in adaptive analyses with many remeshings, we have devised a technique that avoids any searches and checks by generating internal vertices inside existing elements.

Figure 9 summarizes quality indexes for one particular mesh which seems to be sufficiently fine for strength analysis

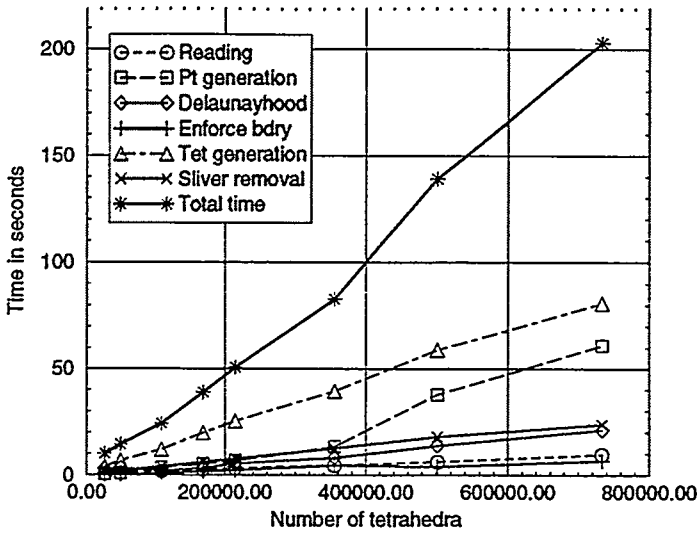
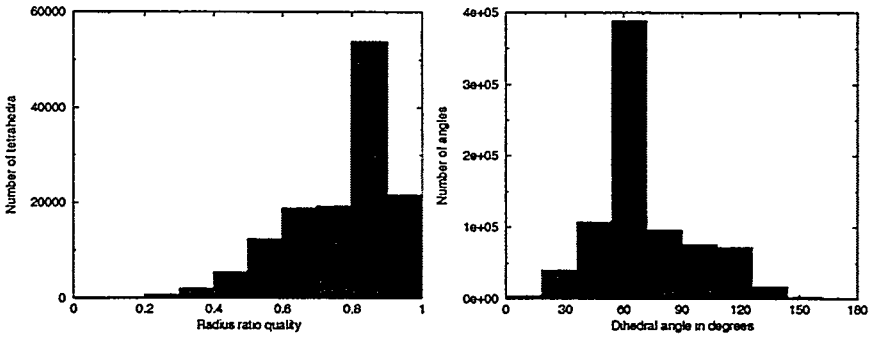


Figure 8: Timing for mesh of hub.



(a) Radius ratio

(b) Dihedral angle

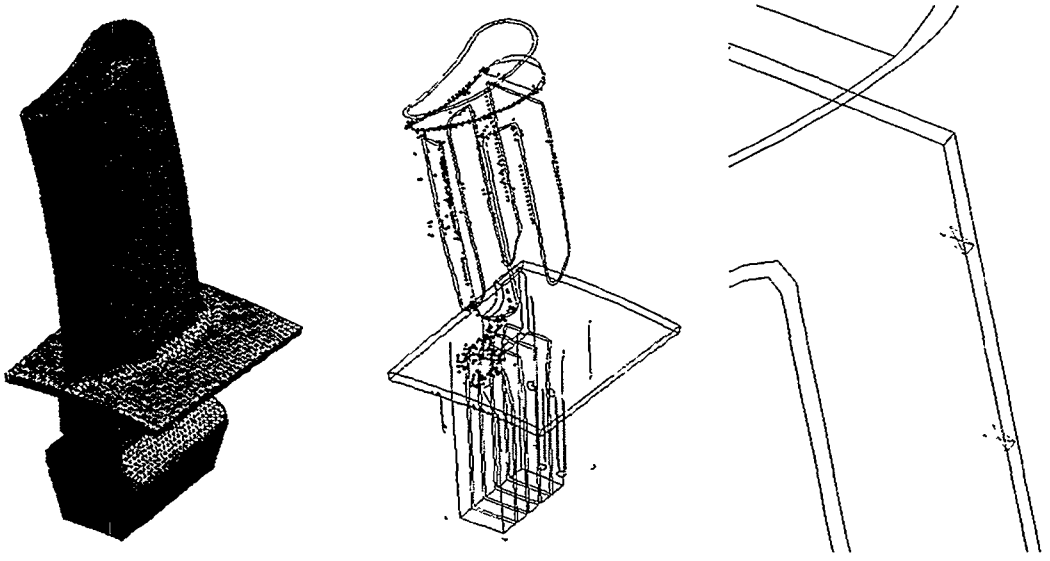
Figure 9: Mesh of a hub. Quality distribution for mesh with 169610 elements.

purposes (with 169610 tetrahedra). The graph on the left shows the distribution of the radius ratio (inscribed sphere radius over circumsphere radius scaled by 3; ideal ratio is one); the graph on the right documents the distribution of the dihedral angle. The minimum radius ratio quality was 0.053, the minimal dihedral angle was 2.94 degrees, and the maximum dihedral angle was 173 degrees.

7.4 Mesh of Airfoil

In the next example we present mesh of an airfoil[3]. The topology and geometry of the object is rather complex. The input to the mesher consisted of 20909 vertices and 41866 boundary facets (see Figure 10(a)). The surface triangulation was of relatively good quality. Nevertheless, the mesher had to introduce a number of additional vertices (508) and facets (1016) during the Delaunayzation procedure because of the large ratio of the mesh size and the local feature size (especially near intersections of thin walls). Figure 10(b) shows the additional vertices as dots on the background of topological edges. The tetrahedral mesh of 72881 elements was generated in 59.9 seconds of which 62% were spent in the Delaunayzation algorithm. As documented in Figure 10(c), the presence of thin walls caused a number of slivers to appear (minimal dihedral angle 0.42 degrees, maximal dihedral angle 179.21 degrees). The reason was that our present mesh optimization technique does not handle slivers with all four vertices bound to

[3]Prototype airfoil geometry courtesy of Howmet Research Corporation.



(a) Input mesh (b) Additional vertices (c) Slivers

Figure 10: Mesh of a prototype airfoil.

two or more constraining surfaces. One possible cure could be introduction of an additional point at the barycenter of the sliver followed by Delaunay remeshing of the cavity remaining after the broken tetrahedra.

Mesh has also been produced with the Delaunayzation procedure based on empty minimal circumspheres (decisions based on Equation (1) were disabled). As expected, the mesher needed more additional vertices (544), and more boundary facets (1088) to arrive at a fully Delaunay surface.

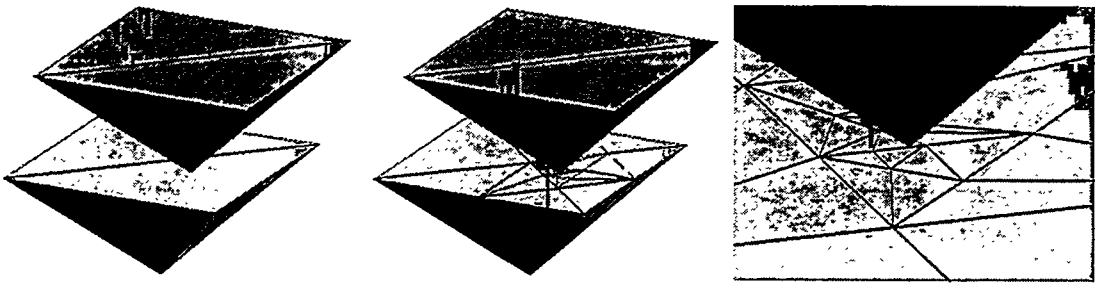
7.5 Mesh of Cranium and Brain

The approach presented above is applicable not only to external surfaces, but works equally well for internal surfaces, such as material interfaces or mathematically-sharp cracks. The only difference is that (in our implementation) the facets on these surfaces are not added to the initial advancing front.

One example of a mesh with material interfaces was presented in Figure 1 (exploded view of the mesh separated into regions). The surface triangulations are polygonal models of simplified skull and brain (11158 vertices and 22322 boundary facets). There are three regions representing the skull, the cerebrospinal fluid with the meninges, and the brain. The surface mesh required relatively minor stitching – 36 additional vertices and 72 facets. The finished mesh consisted of 15398 vertices and 82126 tetrahedra, and took 40 seconds to generate.

7.6 Mesh of Pyramids

One of the open problems is the limited ability of the algorithm to handle surfaces which are almost touching (this is actually common to all unconstrained Delaunay meshers). Consider two bodies very closely spaced. In applications to mechanics, unless the bodies touch there is no reason to take their proximity into account by refining the mesh. However, Delaunay mesh constructed for the ensemble of the two bodies will refine the neighborhood of the "almost in contact" region as shown in Figure 11, which depicts two pyramids arbitrarily offset so that the vertex of one is very close to the base of the other (base dimensions 1×1 unit, height 1 unit, offset of the tip of the second pyramid with respect to the center of the bottom one $\{0.1454, 0.3569, 0.0019\}$). The *DELAUNAYZATION* algorithm enhanced the input mesh consisting of 10 points and 12 facets by additional 14 points and 28 facets. If the two bodies were meshed separately, this problem would not arise, but frequently bodies in self-contact or solids with cracks lead to these situations. One possible way of attack is constrained Delaunay triangulation. While constrained Delaunay



(a) Input mesh (b) Output mesh (c) Detail

Figure 11: Mesh of two almost touching bodies.

triangulations are in general known not to exist in 3-D, Shewchuk demonstrates existence of conditions under which so-called conforming constrained Delaunay triangulation may be constructed [19].

Acknowledgements

We are grateful for support from the Department of Energy through Caltech's ASCI Center of Excellence for Simulating Dynamic Response of Materials. The anonymous reviewer is thanked for very useful comments.

References

[1] M. S. Shephard and M. K. Georges. Reliability of automatic 3D mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 101(1–3):443–462, 1992.

[2] F. Hermeline. Triangulation automatique d'un polyèdre en dimension N. *RAIRO Analyse Numérique*, 16(3):211–242, 1982.

[3] J-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[4] C. Hazlewood. Approximating constrained tetrahedrizations. *Computer Aided Geometric Design*, 10:67–87, 1993.

[5] Gary L. Miller, Dafna Talmor, Shang-Hua Teng, Noel Walkington, and Han Wang. Control volume meshes using sphere packing: Generation, refinement and coarsening. In *5th International Meshing Roundtable, Sandia National Laboratories*, pages 47–62, 1996.

[6] P. Fleischmann and S. Selberherr. Three-dimensional Delaunay mesh generation using a modified advancing front approach. In *Proceedings of the 6th International Meshing Roundtable*, pages 267–278. Sandia National Laboratories, 1997.

[7] J. R. Shewchuk. *Delaunay Refinement Mesh Generation.* PhD thesis, Carnegie Mellon University, May 1997.

[8] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. *Discrete Comput. Geom.*, 12:189, 202 1994.

[9] B. Barber. *Qhull manual.* The Geometry Center, Minneapolis MN, www.geom.umn.edu/software/qhull, 1998.

[10] R. Radovitzky and M. Ortiz. Tetrahedral mesh generation based on node insertion incrystal lattice arrangements and advancing-front-Delaunay triangulation. *International Journal of Numerical Methods in Engineering*, 1998.

[11] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 1998. submitted.

[12] G. M. Turkiyyah, D. W. Storti, M. Ganter, H. Chen, and M. Vimawala. An accelerated triangulation method for computing the skeletons of free-form solid models. *Computer-Aided Design*, 29(1):5–19, 1997.

[13] M. Tanemura, T. Ogawa, and N. Ogita. A new algorithm for three-dimensional Voronoi tesselation. *J. of Computational Physics*, 51(2):191–207, 1983.

[14] M. L. Merriam. An efficient advancing-front algorithm for Delaunay triangulation. *AIAA*, paper 91-0792, 1991.

[15] D. J. Mavriplis. An advancing-front Delaunay algorithm designed for robustness. *AIAA*, paper 93-0671, 1993.

[16] D. H. McLain. Two-dimensional interpolation from random data. *Comput. J.*, 19:178–181, 1976.

[17] R. Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers*, 12:186–210, 1996.

[18] P. Cignoni, C. Montani, R. Perego, and R. Scopigno. Parallel 3D Delaunay triangulation. In R. J. Hubbold and R. Juan, editors, *EUROGRAPHICS '93*, volume 12, pages C–129. Eurographics Association, Blackwell Publishers, 1993.

[19] J. R. Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Proc. 14th Annual Symposium on Computational Geometry*, 1998.

Coupling 1D Beams to 3D Bodies

Dermot J. Monaghan, Ian W. Doherty, David Mc Court and Cecil G. Armstrong

Department of Mechanical and Manufacturing Engineering,
The Queen's University of Belfast,
Ashby Building, Stranmillis Road,
Belfast, BT9 5AH, Northern Ireland.
email: c.armstrong@qub.ac.uk, http://sog1.me.qub.ac.uk/femgroup.html

Abstract: A new technique for coupling one dimensional beam elements to three dimensional bodies is described. The essence of the problem is to ensure compatibility of the displacements at the interface. This is achieved by equating the work done at the dimensional interface by the 3D nodes with the work done at the 1D node. This is carried out in a manner similar to that used in the development of Reissner plate theory and leads to 6 multipoint constraint equations, one for each degree of freedom of the node in the 1D element. The methods used to achieve the coupling for axial force, bending moment, shear force and torsion are discussed and it is shown that the proposed technique does not introduce any spurious stresses at the dimensional interface.

Keywords: Idealisation, Dimensional reduction, Multi-dimensional coupling, 3D - 1D.

1. Introduction

Efficient finite element modelling requires an appropriately idealised representation of 3D design geometry. The most recent contributions to this goal are in the areas of detail suppression [1],[2],[3], dimensional reduction [4], and multi-dimensional coupling [5], [6].

McCune [5],[6], concentrated on 3D-2D and 2D-1D coupling. The logical next step in the process is to reduce regions of constant cross-section in 3D solids to their 1D equivalent beam element. In order to analyse stress concentrations round discontinuities in geometry and loading, the ends of this 1D beam must then be coupled, to the new surfaces created by the dimensional reduction process, Figure 1. By doing so, a marked reduction in the number of DOF required to analyse the model can be achieved without any loss of accuracy.

In this paper the coupling of a 1D beam element to a 3D continuum is described. Though the procedure has only been implemented for linear elements, it illustrates the concept and may be further developed into a general and robust process. The reader is introduced to the area by firstly dealing with the simple case for axial force in section 3. This will be followed by the bending moment and torsion cases in sections 4 & 5 respectively. In section 6 the shear force case will be discussed, and section 7 provides some model examples and results. Finally sections 8 and 9 conclude the work presented and indicate the plans for further development.

2. Notation

Beam Cross-Sectional Area	A
Beam Displacements	u, v, w
Beam Forces	F_x, F_y, F_z
Beam Rotations	θ_x, θ_y, θ_z
Bending moments	M_x, M_y, M_z
Centroid of element face	x_c, y_c
Continuum Displacements	U, V, W
Moment of Inertia	I
Poisson's Ratio	ν
Shape functions	N
Shear Modulus	G
Stress (direct)	σ_x, σ_y, σ_z
Stress (shear)	τ_{xy}, τ_{xz}, τ_{yz}
Stress function	ϕ
Young's Modulus	E

3. Axial Force

The following derivation is based on an outcome of Reissner's transverse plate bending theory [7] by which simple relationships are shown to be obtainable between the translational and rotational degrees of freedom in a plate and the displacements in the 3D continuum. To couple a beam and a solid, the first step is to equate the work done on either side of the interface between dimensions.

Equating the work done by the axial force acting on the 1D beam with the work done by the surface stresses of the 3D body at the interface, the following equation results:

$$F_z w = \int_A \sigma_z W dA \quad \text{... [3-1]}$$

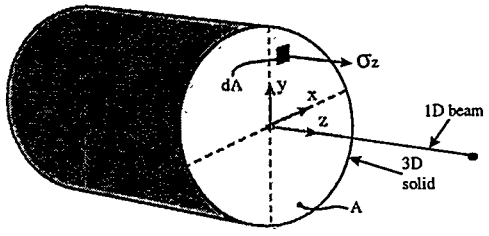


Figure 1

If the 3D region is long and slender, then the axial stress is uniform over the cross-section and is given by:

$$\sigma_z = \frac{F_z}{A} \quad \text{... [3-2]}$$

In the 3D model, the axial displacement at any point, in terms of the nodal displacements {W} and shape functions [N], is:

$$W=[N]\{W\} \quad \text{... [3-3]}$$

This implies:

$$F_z w = \frac{F_z}{A} \sum_{i=1}^{Nelements} \int_{A_i} [N] dA \{W\} \quad \text{... [3-4]}$$

Since this must be true for any axial force, the beam displacement w and the displacements of the 3D continuum nodes on the interface $\{W\}$ must be related by:

$$Aw = \sum_{i=1}^{Nelements} \int_{A_i} [N] dA \{W\} = [B]\{W\} \quad \text{... [3-5]}$$

For linear elements such as the 4-noded tetrahedron or the 6-noded wedge, the integral of the shape functions over the area of the element face on the interface between dimensions is:

$$\int_{A_i}[N]dA = \frac{A_i}{3} \quad \text{... [3-6]}$$

Each term B_i is the sum of the areas of all the elements faces on the interface attached to node i, divided by 3.

Displacement compatibility between the 1D beam element and the adjacent 3D continuum elements can therefore be enforced as a multipoint constraint equation of the form

$$-a_0 w + B_1 W_1 + B_2 W_2 + B_3 W_3 + \ldots = 0 \quad \text{... [3-7]}$$

This can be applied in the ABAQUS™ commercial finite element package as a *EQUATION command [8].

4. Bending Moment

Again Reissner's relationships can be used to equate the work done on either side of the coupling interface. Considering the moment about the x-axis and equating the work done at the interface gives the following equation:

$$M_x \theta_x = \int_A \sigma_z W dA \quad \text{... [4-1]}$$

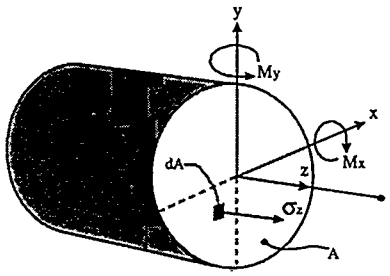


Figure 2

From simple beam theory, assuming for simplicity that the co-ordinate system is aligned with the principal axes of the cross section,

$$\sigma_z = \frac{M_x y}{I_{xx}} \quad \text{... [4-2]}$$

Substituting also for the displacements in the z-direction in the 3D finite element model,

$$W=[N]\{W\} \quad \text{... [4-3]}$$

On substitution, the beam rotation θ_x is related to the axial displacements of continuum nodes on the interface {W} by the equation:

$$\theta_x = \frac{1}{I_{xx}} \sum_{i=1}^{Nelements} \int_{A_i} y[N]dA\{W\} = [B]\{W\} \quad \text{... [4-4]}$$

This is the general solution for any type of element.

Consideration of the bending moment acting about the y-axis gives a similar equation:

$$\theta_y = \frac{1}{I_{yy}} \sum_{i=1}^{Nelements} \int_{A_i} x[N]dA\{W\} = [C]\{W\} \quad \text{... [4-5]}$$

Assuming that linear elements are to be used to implement the coupling procedure, then :

$$\int_{A_i} y[N]dA = \tfrac{1}{3} A_i y_c \quad \text{... [4-6]} \; ; \qquad \int_{A_i} x[N]dA = \tfrac{1}{3} A_i x_c \quad \text{... [4-7]}$$

where (x_c, y_c) are the co-ordinates of the centroid of the element face A_i of the cross section.

5. Torsion

The distribution of shear stress on the cross-section of a beam subjected to a torsional moment is commonly solved by introducing a stress function. If a function $\phi(x,y)$, the Prandtl stress function, is assumed to exist such that:

$$\tau_{xz} = \frac{\partial \phi}{\partial y} \quad \text{... [5-1]}; \qquad \tau_{yz} = -\frac{\partial \phi}{\partial x} \quad \text{... [5-2]}$$

then the stress function must satisfy the differential equation:

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+2G\theta=0 \qquad \text{... [5-3]}$$

where θ is the twist per unit length of the beam and G is the shear modulus. The stress function therefore must satisfy Poisson's equation [9]. A conductive heat transfer case also may be represented as a form of Poisson's equation.

Therefore, the variation of the Prandtl stress function over any cross-section can be found using the facilities available in standard finite element packages for conductive heat transfer. The shear stress on the cross-section can be inferred from the resulting temperature gradients. The cost of a 2 dimensional heat transfer analysis of the cross-section is much less than that of a 3D analysis of the stress.

The total torque generated by a given twist can be found from the torsion analysis [9] as:

$$M_z = 2\int_A \phi dA \quad \text{... [5-4]}$$

The coupling equation is again formed by equating the work done at the interface.

On the 3D side of the interface, the work done is:

$$\Pi = \tfrac{1}{2}\int_A (\tau_{xz}U + \tau_{yz}V)dA \quad \text{... [5-5]}$$

On the 1D side, the work done is:

$$\Pi = \tfrac{1}{2}\theta_z M_z \quad \text{... [5-6]}$$

Therefore,

$$\theta_z M_z = \int_A (\tau_{xz}U + \tau_{yz}V)dA \quad \text{... [5-7]}$$

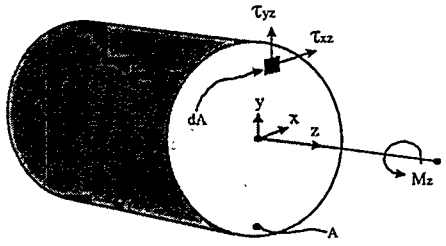


Figure 3

Replacing the integral over the whole cross-section with the sum of the integrals over the element faces lying on the interface and the continuum displacements with the 3D finite element displacements:

$$\theta_z M_z = \sum_{i=1}^{Nelements}\int_{A_i} \tau_{xz}[N]dA\{U\} + \sum_{i=1}^{Nelements}\int_{A_i} \tau_{yz}[N]dA\{V\} = [C]\{U\} + [D]\{V\} \quad \text{... [5-8]}$$

The total torque M_z in response to any arbitrary twist can be found from Equation *[5-4]*, as can the shear stresses τ_{xz} and τ_{yz} on each element face. Evaluating the [C] and [D] matrices reduces to summing integrals of the form:

$$\int_{A_i} \tau_{xz}[N]dA \quad \text{... [5-9]}$$

The shear stress τ_{xz} can be written in terms of shape functions [N] and nodal values of the stress function $\{\phi\}$ from the 2D analysis of the cross-section as:

$$\tau_{xz} = \frac{\partial\phi}{\partial y} = \frac{\partial}{\partial y}[N]_{2D}\{\phi\} = \{\phi\}^T[\frac{\partial N}{\partial y}]^T_{2D} \quad \text{... [5-10]}$$

so that the integral over an element face becomes:

$$\int_{A_i} \tau_{xz}[N]_{3D}dA = \{\phi\}^T\int_{A_i}[\frac{\partial N}{\partial y}]^T_{2D}[N]_{3D}dA \quad \text{... [5-11]}$$

where $\{\phi\}$ are the nodal temperatures found in the heat transfer analysis. The complete multipoint constraint equation can then be written in the form:

$$-M_z\theta_z + C_1u_1 + C_2u_2 + C_3u_3 + \ldots + D_1v_1 + D_2v_2 + D_3v_3 \ldots = 0 \quad \text{... [5-12]}$$

This equation can also be applied as a linear constraint equation in the finite element model (e.g. applied in ABAQUS™ as a *EQUATION command). The effect of this equation is to couple the displacements of the 3D continuum nodes on the interface to the twisting rotation of the beam node such that the distribution of shear stress on the interface is the same as that given by the St. Venant torsion analysis of the beam cross-section.

6. Shear Force

The shearing stresses, which arise from the action of shear force on the cross-section of the beam, are smaller than the bending stresses by a factor proportional to the slenderness ratio of the beam and so have been neglected here. The approach in sections 3-5 may be extended to ensure the proper distribution of shear stress in response to a given shear force using an analysis of the cross-section similar to that for torsion [10].

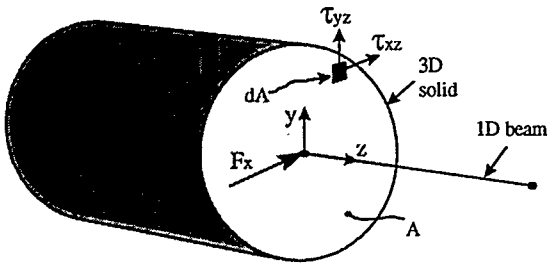


Figure 4

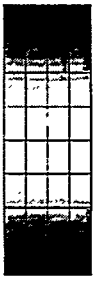


Figure 5: Shear stresses on rectangular section resulting from a shear force

The stresses on the cross-section at any point (x,y) due to a shear force F_x on the cross-section are given in terms of a stress function ϕ as:

$$\tau_{xz} = \frac{\partial \phi}{\partial y} - \frac{F_x x^2}{2I_{yy}} + \frac{\nu}{2(1+\nu)} \frac{F_x y^2}{I_{yy}} \quad \text{... [6-1]}; \qquad \tau_{yz} = -\frac{\partial \phi}{\partial x} \quad \text{... [6-2]}$$

where ϕ must obey the governing equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \quad \text{... [6-3]}$$

with the boundary condition on the boundary of the section

$$\phi = \frac{F_x}{I_{yy}} \int \frac{x^2}{2} dy - \frac{\nu}{2(1+\nu)} \frac{F_x y^3}{3I_{yy}} + const. \quad \text{... [6-4]}$$

Figure 5 illustrates the shear stress τ_{xz} derived from the results of a 2D thermal analysis of the cross-section. Given the shear stresses on the cross-section, the appropriate multipoint constraint equation can be generated as before by equating the work done on both sides of the interface. This yields:

$$F_x u = \int_A (\tau_{xz} U + \tau_{yz} V) dA \quad \text{... [6-5]}$$

This procedure has been demonstrated successfully for 2D - 1D coupling of beams and plates [6].

7. Results

Sample results for the cases considered are given in their respective sections below. They demonstrate the concept graphically and give an indication of the possible benefits of mixed dimensional modelling.

a) Axial force

The stress plot of the coupling for the Axial force case is shown in *Figure 6* on the right. The stress is constant across the section as expected, and is the same as the analytical result.

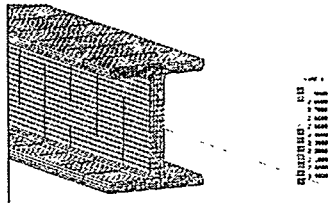


Figure 6: Coupling for axial force

b) Bending Moment

The stress plots of the coupling for the bending moment cases are shown in Figure 7 and Figure 8 below. Again the results compare favourably with analytical calculations. Note that there is no disturbance to the stress contours at the interface between the 1D beam and the 3D solid elements. The slight discontinuity of the neutral axis is due to the linear interpolation and stress smoothing on the four noded tetrahedron elements used.

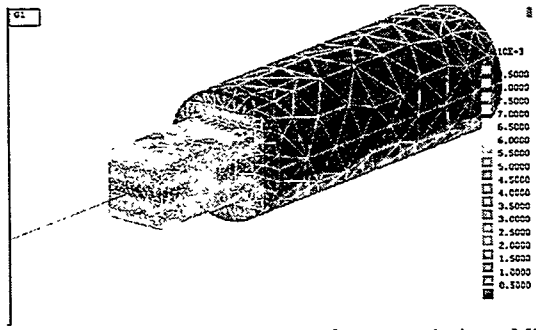


Figure 7: Coupling for moment about x-axis (von Mises stress)

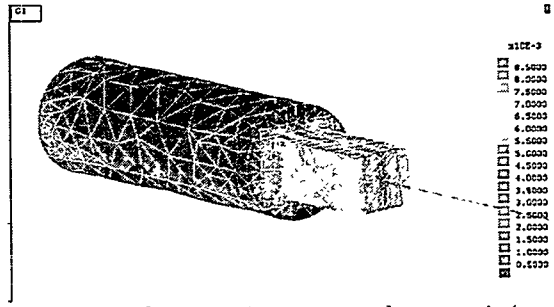


Figure 8: Coupling for moment about y-axis (von Mises stress)

c) Torsion

The stress function obtained from St. Venant torsion analysis of a square cross section is shown in Figure 9. From equations *[5-1]* and *[5-2]*, shear stresses are obtained from the derivatives of the stress function. The contour plot of the derivatives of the stress function is shown in Figure 10. These contour lines of stress, obtained from the 2D analysis, correspond with values obtained from a 3D analysis, Figure 11.

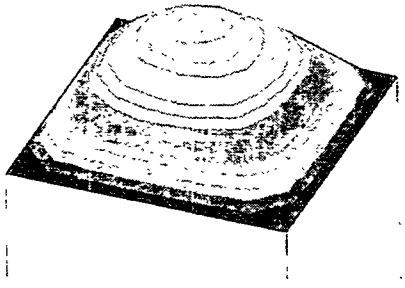


Figure 9: Stress function obtained from St. Venant torsion analysis of the cross section

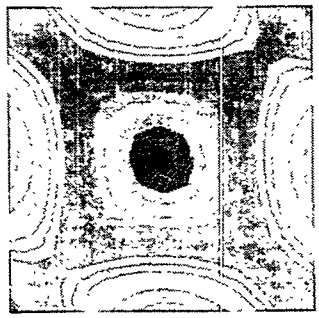


Figure 10: Plot showing slope of stress function in Figure 9

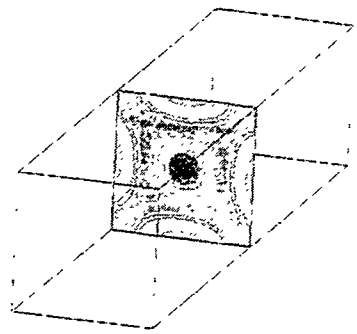


Figure 11: Section of 3D model shows von Mises stress for torsion of a square bar

Figure 12 and Figure 13 show the von Mises stress obtained for coupling a square bar to a beam element of equivalent properties. Again there is no disturbance to the stress contours at the dimensional interface and the stresses on the interface are effectively identical to those obtained by the St. Venant torsion analysis, Figure 10, or at sections in the interior of the 3D model Figure 11. There is some disturbance at the rigidly fixed end due to the method by which the fixing was defined.

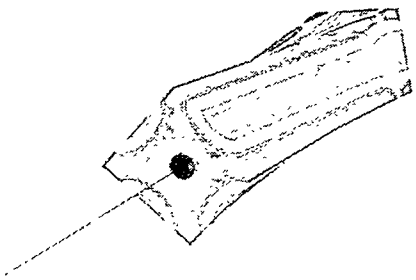


Figure 12: Torsional load (Max shear stress)

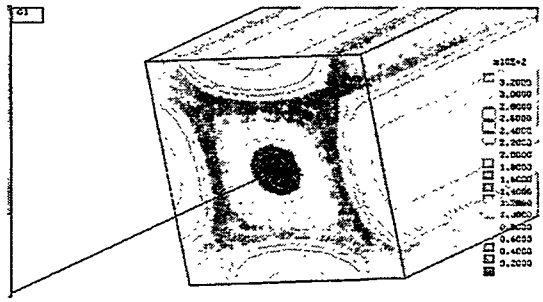


Figure 13: Torsion on block

As can be seen from the contours of maximum shear stress in Figure *14* and Figure *15* the coupling equations produce stress contours which are axisymmetric and as close to the analytic solution for a thin walled tube as can be obtained with constant stress elements and nodal averaging.

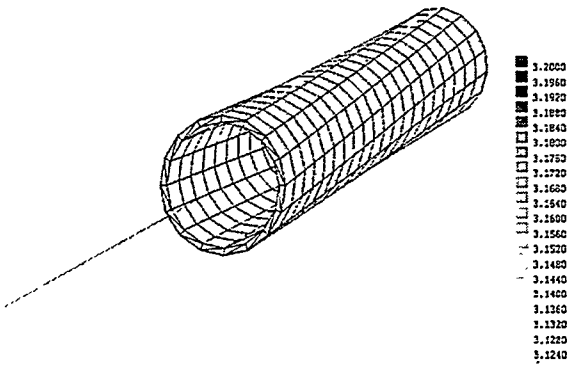


Figure 14: Coupling for Torsion of tubular section (max. shear stress)

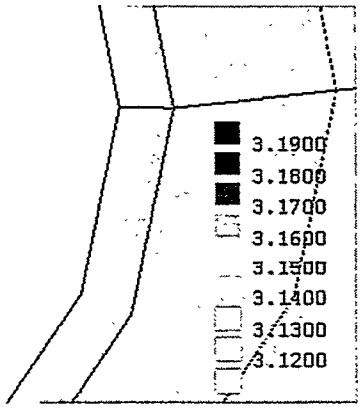


Figure 15: Torsion (max. shear stress)

8. Discussion

Using the procedure described here, the analysis of a slender region within a 3D solid can be reduced to a 1D analysis plus 6 simple analyses of the cross-section. These determine the stress distribution on the cross-section in response to each component of force and moment.

The cross-sectional analyses can also be used to generate 6 multipoint constraint equations linking the displacements and rotations of the beam element to the nodal displacements of the 3D continuum elements representing the material adjacent to the slender area of the model represented by the beam element. This greatly facilitates mixed-dimensional modelling, where 3D details such as joints, changes in section or loading are properly represented, but dimensionally-reduced beam elements are used to model slender parts (economically and accurately), where 3D elements are expensive and potentially ill-conditioned.

The coupling does not introduce spurious stresses at the interface, so that measures based on the implied stress jumps between the 3D and the reduced dimensional models can be used as an *a posteriori* estimate of the idealisation error introduced by dimensional reduction [5],[6]. The cross-sectional analyses can also be used to determine section properties for beam elements and to transfer beam force and moment results to stresses on the cross-section for visualisation.

The ideas presented here have already been applied to beam-plate-solid coupling [5] and should be extensible to the coupling of arbitrary combinations of beam, plate, shell and solid elements of any formulation.

9. Conclusions

- A general technique for coupling 1D beams to 3D continuum elements has been developed.
- Only 6 multipoint constraint equations are required to couple the 1D and 3D elements at the interface between dimensions.
- The coupling does not introduce any errors at the interface between dimensions.
- The technique should greatly facilitate appropriate mixed-dimensional modelling of structures containing slender parts and 3D details.

10. References

[1] Bridgett S.J., "Detail Suppression of Stress Analysis Models", PhD thesis, The Queen's University of Belfast, November 1997.

[2] Blacker T., Sheffer A., Clements J., Bercovier M., "Using Virtual Topology to Simplify the Mesh Generation Process", Proceedings of 1997 joint ASME/ASCE/SES summer meeting, Evanston, USA, published in American Society of Mechanical Engineers, AMD, Vol. 220, 1997, pp 45-50.

[3] Blacker T., Sheffer A., Clements J., Bercovier M., "Steps Towards Smooth CAD-FEM Integration", Proceedings of 6th International Conference of Numerical Grid Generation in Computational Field Simulation, Greenwich, July 1998.

[4] Donaghy R.J., "Dimensional Reduction of Stress Analysis Models", PhD thesis, The Queen's University of Belfast, May 1998.

[5] McCune R.W., "Mixed Dimensional Coupling and Error Estimation in Finite Element Stress Analysis", PhD thesis, The Queen's University of Belfast, March 1998.

[6] Armstrong, C.G., Bridgett, S.J., Donaghy, R.J., McCune, R.J., McKeag, R.M. and Robinson, D.J., "Techniques for Interactive and Automatic Idealisation of CAD Models", 6th International Conference on Numerical Grid Generation in Computational Field Simulation, London, July 1998.

[7] Reissner E., "On Bending of Plates", Quarterly of Appl. Math., Vol. 5, 1947, pp 55-68.

[8] Hibbitt, Karlsson & Sorenson Inc., ABAQUS / Standard Users Manual, 1997, pp20.2.1-1.

[9] Timoshenko S.P., Goodier J.N., "Theory of Elasticity", 3rd edition, McGraw-Hill, New York, 1970, pp 291-349.

[10] Timoshenko S.P., Goodier J.N., "Theory of Elasticity", 3rd edition, McGraw-Hill, New York, 1970, Art 128.

The All-Hex Geode-Template for Conforming a Diced Tetrahedral Mesh to any Diced Hexahedral Mesh

Scott A. Mitchell[1]

Abstract. Take a hexahedral mesh and an adjoining tetrahedral mesh that splits each boundary quadrilateral into two triangles. Separate the meshes with a buffer layer of hexes. Dice the original hexes into eight, and the tetrahedra into four hexahedra. Then I show that the buffer layer hexes can be filled with the geode-template, *creating a conforming all-hex mesh of the entire model. The geode-template is composed of 26 hexahedra. The hexahedra have acceptable quality, depending on the geometry of the buffer layer. The method used to generate the geode-template is general, based on interleaving completed dual surfaces, and might be extended to other transition problems.*

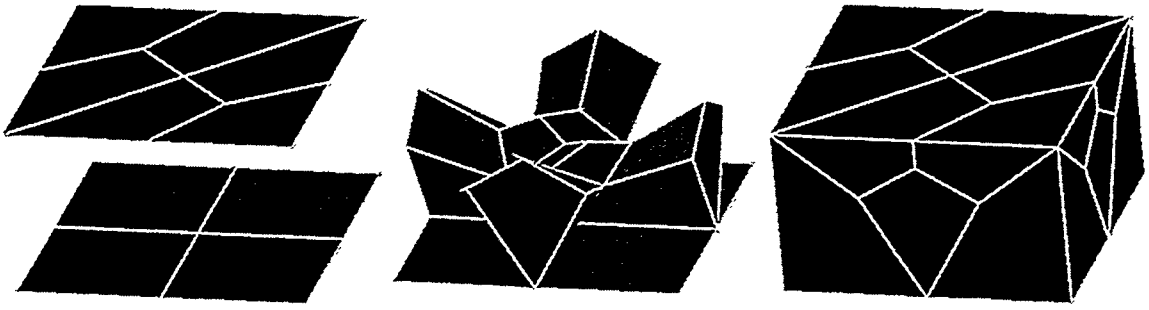


Figure 1. The all-hex geode-template: Left, the diced tet and hex interface; middle, the heart of the geode; right, the boundary of the template.

keywords. mesh generation, hexahedra, tetrahedra, conforming, transition

1. Introduction

For some FEM calculations, hexahedral meshes are preferred to tetrahedral meshes. However, for a geometrically complicated domain, automatically generating a hexahedral mesh is much more difficult than generating a tetrahedral mesh. A hexahedral mesh can always be obtained from a tetrahedral mesh by *dicing*, dividing each tetrahedra into four hexahedra, but the resultant mesh quality is relatively poor.

At Sandia National Laboratories, SNL, we often want to analyze models composed of hundreds of simple parts, arranged together in a complicated way. In many models, there is a potting material surrounding these parts whose geometry is the complement of the union of the other parts. This potting material is often the most difficult to mesh, but the least interesting to the analyst. Also, at SNL and in industry, we commonly have a complicated part that may be mostly decomposed into simple pieces, with one or two difficult nuggets remaining. The mesh of the entire model should be *conforming*, that is, the mesh should be a simplicial complex, with elements meeting face-to-face, edge-to-edge, and node-to-node.

For these scenarios, a reasonable solution is to first (automatically) mesh the simple or important parts with high-quality hexahedra, then automatically mesh the complicated or unimportant parts with tetrahedra. On the surfaces between hex-parts and tet-parts the quads can be divided into two triangles; many Delaunay-based tetrahedral meshers

1 samitch@sandia.gov, http://endo.sandia.gov/~samitch. Parallel Computing Sciences Dept., Sandia National Laboratories, Albuquerque, NM 87185. The author was supported by the Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy, Office of Energy Research, and works at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AL04-94AL8500. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE.

can generate a tet mesh that conforms to these triangles. Hence the tetrahedral and hexahedral meshes will conform node-wise, but each quadrilateral will have a diagonal edge cutting it into two triangles.

The problem is what to do with this non-conforming interface. One solution is *tied contacts,* modifying the analysis software to handle a non-conforming mesh by interpolating between intermediate FEM solutions along the interface. Besides complicating the software, in some situations the running time and analysis errors are unacceptably high. Another solution is to introduce square pyramid elements. The interface is now conforming, but again the analysis code must be modified, this time to handle square pyramid elements; many SNL analysis codes do not even support the more common tetrahedral elements.

In this paper I propose a new solution which generates a conforming mesh for the entire model composed entirely of hexahedra. First, the tetrahedral mesh is geometrically shrunk away from its interface with the hex mesh, creating a boundary layer of hexahedra. (The boundary layer could be created before the tet-mesh. Note that for some interface geometries, e.g. Figure 9 right, shrinking is impossible.) Each hex of the boundary layer has the same structure: the top quadrilateral face is shared with the tet mesh and divided into two triangles, the opposite bottom face is shared with the hex mesh, and the remaining four side faces are shared with other hexes of the boundary layer.

Second, the hexahedral mesh outside the boundary layer is diced by dividing each hex into 8, and the tetrahedral mesh inside the layer is diced into hexes. Boundary layer "hexes" now have six quads on the diced-tet-mesh interface and four quads on the diced-hex-mesh interface. Third, each boundary layer hex is filled in with the *geode-template,* which is composed of 26 hexahedra and conforms to the diced meshes. Each of the four side faces have the same quad mesh, so templates match up and the entire mesh is conforming.

There are several options and applications: The entirety of a solid-model part need not be meshed with tetrahedra. For example, Plastering could fill much of the volume, and tet-meshing fill only the remaining voids; see Meyers[4] and Tuchinsky[11]. There are a number of issues related to tool infrastructure and finding good geometric positions for the boundary layer as well.[10] Finding a hex-template for slightly different situations has been studied for some time. The most famous instance is Schneiders's open problem, which has an interesting history and is accessible from the web.[9] Two template problems with similar structure arise in Eppstein's hex-mesh existence proof.[1]

The method of generating the geode-template is general and could be extended to other transition problems. The method is based on the Spatial Twist Continuum, STC [6][7], the surface arrangement dual to a hex mesh. In particular, it is based on a new form of Whisker Weaving's *sheet* (surface) moving[2]: First I create a completion of the STC of the diced tet mesh, then a separate completion of the STC of the diced hex mesh. I then push these two arrangements together so that they intersect, combining the STCs. Dualize the combined STC creates the hexes of the geode-template.

The advantage of the geode-template is that all-hex meshing is automated and analysis codes do not need to be modified. One drawback is that the element count is high because of the dicing step. Also, the quality of the worst-element is probably worse than in a diced-tet-mesh of the entire model. This is offset by the geode-template allowing excellent-quality structured meshes in areas of interest.

The remainder of this paper is organized as follows. Section 2 describes the general transition method used to create geode-templates. Section 3 describes the hexes of the geode-template and comments on mesh quality. Section 4 presents some preliminary examples of models meshed with the geode algorithm. I discuss in section 5 extending the geode-template construction method, and conjecture why filling certain other hex-templates is difficult. Section 6 presents conclusions. Appendix A summarizes node-position and hex-connectivity for those wishing to reconstruct the geode-template.

2. General Transition Method

In this section I describe a general method for making two adjacent meshes conform by dicing them and inserting a transition layer. First, in section 2.1, I illustrate the process when the two adjacent meshes arise from a triangular mesh adjacent to a quadrilateral mesh, the two-dimensional geode-template. In section 2.2 I describe the construction of the three-dimensional geode-template.

2.1 Two-dimensional Geode-Template Construction

The two dimensional version of the conforming transition problem is trivial: The 1-dimensional elements, edges, are the same in triangular and quadrilateral meshes, so they already conform. In three-dimensions, the interface is non-conforming: Each interface-quad is subdivided into two triangles. Carrying out the geode-formation process in two-dimensions aids in understanding the three-dimensional case.

First, I separate the interface between the triangular and quad mesh. This means topologically splitting the shared nodes and edges into two, and geometrically shrinking the triangular mesh away from the quad mesh; see Figure 2. (Alternatively, the interface could be separated before the triangular mesh is generated.)

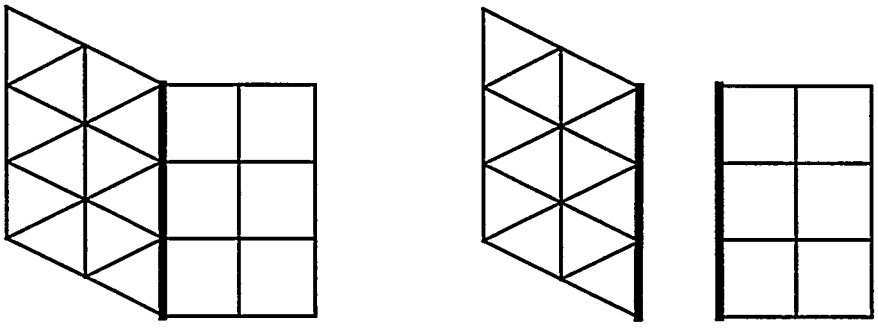


Figure 2. First, the interface is separated. In these pictures the meshes are structured, but both meshes can be unstructured.

Second, I dice the meshes, then complete the dual curves for each diced mesh separately; see Figure 3. Dicing the triangular mesh is necessary in order to convert triangles into quads. More subtly, dicing both meshes ensures that the dual curves can be completed in pairs. I join the two curves that arise from dicing an interface quad. Note that the dual of a diced triangular mesh is composed of circular curves, except where the interface interrupts it. For the triangular mesh I join curves to complete these circles: For a node N of the original triangular mesh on the interface, it has two original interface edges, E_1 and E_2. Each interface edge gets diced in two, creating E_{11}, E_{12}, E_{21}, and E_{22} in order. I join the ends of the curves dual to E_{11} and E_{22}, the diced edges not containing N, as in Figure 3.

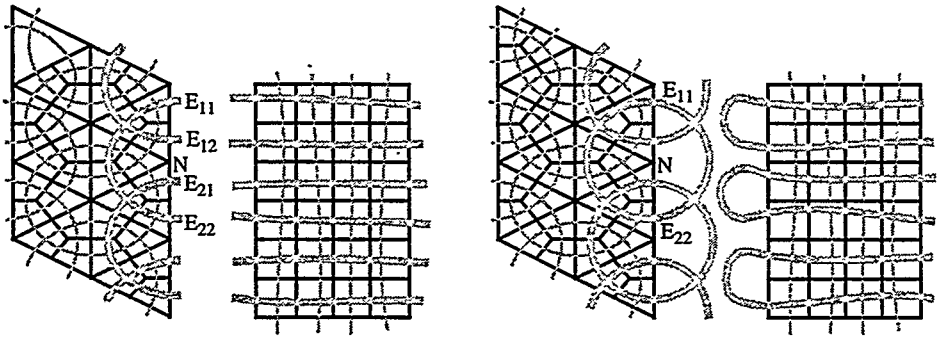


Figure 3. Second, left, the triangular and quadrilateral meshes are diced and the dual curves constructed. Right, the STCs are then completed for each mesh separately.

Third, I push the completed duals together so that they overlap; see Figure 4. Dualizing creates the primal mesh.

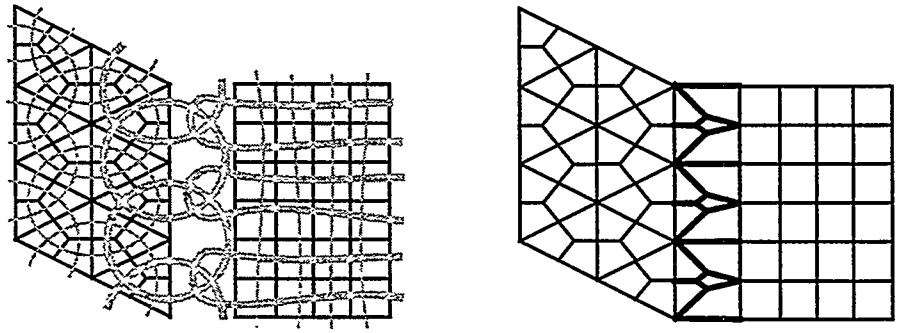


Figure 4. Left, the completed duals are pushed together so that they overlap. Right shows the resulting primal mesh.

2.2 Three-dimensional Geode-Template Construction

The same principles used in constructing the two-dimensional geode-template apply to the three-dimensional case. The main difference is the structure of the completed STCs. Instead of just completing curves, I must complete surfaces. For the hexahedral mesh, its interface mesh is composed of quads. The dual curves of the interface mesh are closed curves. Dicing converts each curve into two identical, parallel copies. I complete the diced hex mesh dual by making a tunnel surface for each parallel pair. Note that at each pre-diced quad, two tunnels pass through each other; see Figure 5. Since the arrangement must be in general position in order to dualize to a hexahedral mesh[5], I must chose which tunnel's roof is higher than the other. The choice is arbitrary; there does not need to be any consistency between neighboring pre-diced quads. But this does introduce an asymmetry within the template.

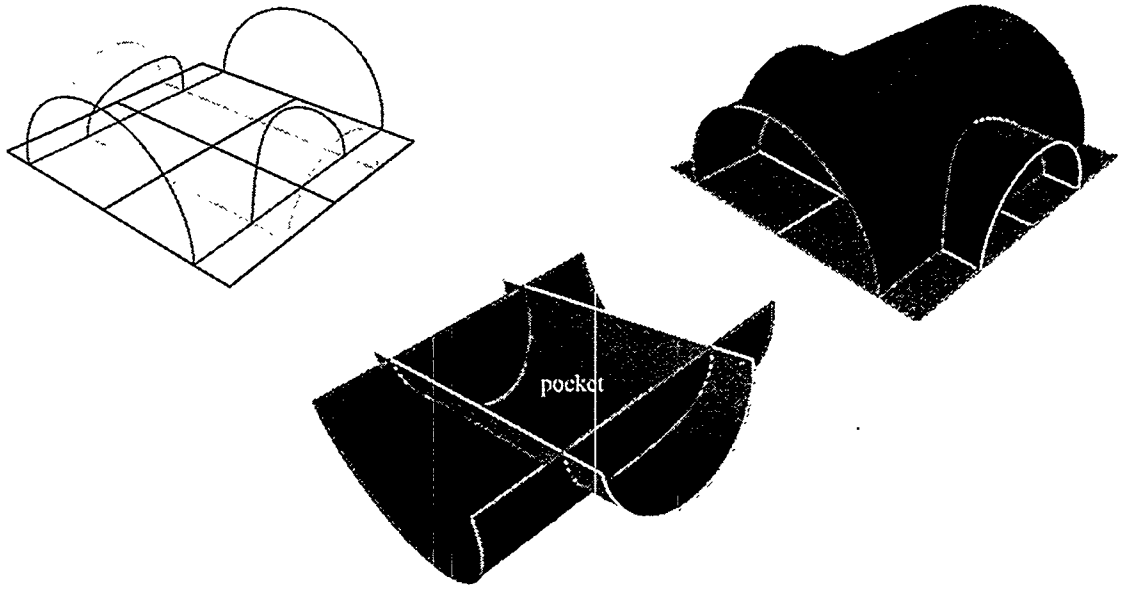


Figure 5. A local view of the completed dual surfaces for the diced hexahedral mesh. Right and left, the bottom four quadrilaterals are the diced hex interface mesh. Center, the quads have been removed and the tunnels turned upside-down. I use cylinders of different radii only to illustrate that one tunnel locally passes below the other; topologically, the tunnels always match up exactly with tunnels of neighboring templates.

For the tetrahedral mesh, its interface mesh is composed of triangles. Dicing produces circle-curves on the interface[6], see Figure 3. I complete these circles into spheres; see Figure 6. Considering a section of the surfaces above a pre-diced

quad, note that two of the completed circles intersect in a curve that starts and ends on two quads of the diced triangular mesh.

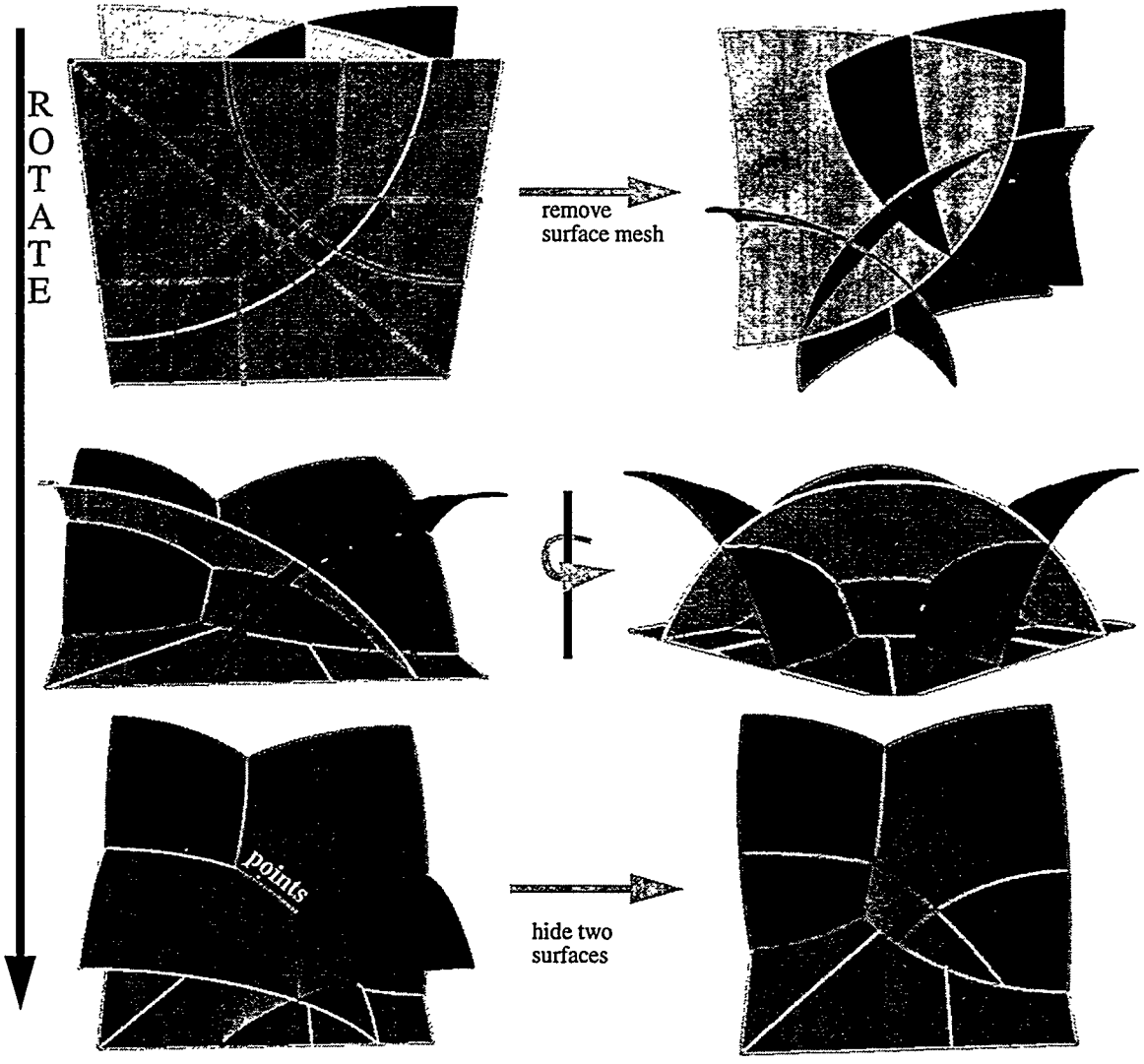


Figure 6. The completed dual surfaces for the diced tetrahedral mesh.

I now push the completed duals together. I can choose how far to overlap the duals. There are two key features of my choice: First, the two points where three diced-tet surfaces intersect, as in Figure 6 bottom left, lie inside both tunnels, the *pocket* in Figure 5 center. Second, each of the curves of intersection between two diced-tet surfaces, except the curve that starts and ends on the interface mesh, exit the side of the template inside one of the arch-like tunnel entrances, so that the side of the three-dimensional template looks like the two-dimensional template as in Figure 4 or Figure 1 center. See Figure 7 for views of the entire arrangement.

I am fortunate that the arrangement dualizes to a well-defined hexahedral mesh, with no degenerate elements, without any fix-ups[5][2] needed. The next section discusses the geode-templates hex structure and quality.

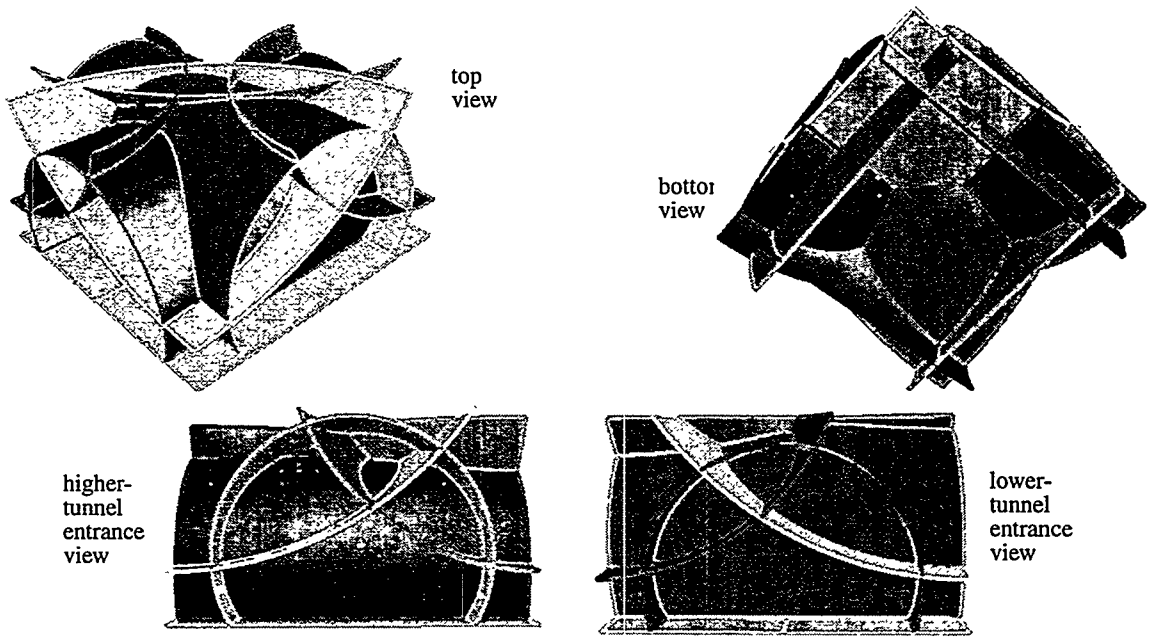


Figure 7. The total arrangement of the pushed-together diced-tet and diced-hex completed duals. Note that looking into the entrance of the higher tunnel is different than looking into the entrance of the lower tunnel; compare the lower left of the figure with the lower right.

3. Geode-Template Hexes

3.1 Hex Structure

Since the mesh is highly-unstructured, the traditional views of layers through z-planes are not very informative. Removing hexes from the outside-in is slightly better, such as Figure 1 center. The most useful static views I have found are hex layers dual to each surface of the STC. Note that the layers from the diced-hex-mesh and the diced-tet-mesh intersect, but the layers are separate. Each tunnel is radially symmetric about a vertical line through the center of the template. To truly understand the hexes of the template, I suggest creating a computer model, or even better a physical toothpick or pipe-cleaner model, based on Appendix A. An interesting feature is that in these models the hexes appear to rotate by 45 degrees when travelling from the diced-triangle top to the diced-quad bottom.

3.2 Mesh Quality

The quality of the geode-template obviously depends on the shape of the buffer layer. The best-quality hexes appear to be achieved with a short geode template. This is serendipitous, since it means that the buffer layer can be thin. Let the *scaled jacobian* at a node of a hex be defined as the triple product of the vectors along the three edges of the node, divided by the product of the lengths of the vectors. The scaled jacobian for a hex is the minimum scaled jacobian among its nodes. For a geode-template inside a rectangular parallelepiped with a square base of length 24 and height 14, and nodes positioned with optimized-jacobian based smoothing[3], the scaled jacobian ranges from 0.26 to 0.53.

A complete quality summary follows in Table 1, "CUBIT quality report for a 24 x 14 x 24 geode-template." Quality measures are defined in Robinson.[8]

Table 1: CUBIT quality report for a 24 x 14 x 24 geode-template.

Quality Measure	Average	Std. Dev.	Minimum	Maximum
Aspect Ratio	1.75	0.55	1.07	3.07
Skew	0.54	0.16	0.17	0.76
Taper	0.30	0.13	0.038	0.76
Element Volume	300	223	97	786
Stretch	0.41	0.13	0.25	0.63
Diagonal Ratio	0.53	0.09	0.38	0.74
Dimension	3.19	0.94	1.97	5.00
Jacobian	6.99	44	26	164
Scaled Jacobian	0.34	0.09	0.26	0.53

4. Examples

This section gives some preliminary examples of meshes created in CUBIT with the geode-algorithm.[10] This algorithm blends the geode-template with MSC's ARIES tet mesher and CUBIT's Plastering[4] hex-dominant mesher. As algorithms for positioning nodes mature I expect mesh-quality to increase.

The first example is from Clay Fulcher; see Figure 8. It consists of a simple geometry, a cube, but with imprinted circles of different sizes on three sides that prevent 2.5-dimensional meshing and make it difficult to decompose. This nugget is surrounded by large, sweepable parts. The geode algorithm successfully produces an all-hex mesh.

In Fulcher's example I place the transition layer directly on the geometry, without Plastering any of the volume. This produces 6864 geode-template hexes, with worst scaled-jacobian 0.017 and worst aspect ratio 11. There are 6140 diced-tet hexes, with worst scaled-jacobian 0.038 and worst aspect ratio 10.

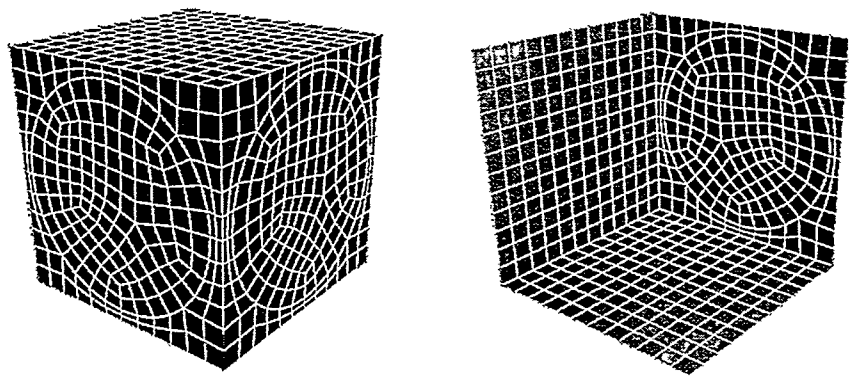


Figure 8. The frontal (left) and cut-away (right) view of Clay Fulcher's problem after dicing. Despite the simple geometry, a highly-unstructured hex-mesh is necessary.

The next example is the square pyramid of Schneiders's open problem; see Figure 11. There are 512 hexes, with worst scaled-jacobian 0.033 and worst aspect ratio 9.4. I chose this example to be cute; I'm filling the pyramid transition element with the geode transition template. The geode-template solution probably does not have any practical value and does not solve the open problem because dicing is required.

Another type of example that would demonstrate the geode-template's utility is the cube-complement of a collection of simple parts. I could also allow Plastering to fill part of Fulcher's model. At the time of this writing I can reliably generate "meshes" for examples such as these, but I can not automatically get good-quality nodal positions. In many cases I have been able to get good quality meshes by positioning nodes by hand, so I do not think the connectivity is fundamentally bad.

5. Extensions and Non-template methods

There are several natural variations on the construction to explore. The first variation is that the surfaces could be completed in a different way. In particular, perhaps Whisker Weaving or another algorithm could complete the STC of the diced hex mesh in another way, so that the surfaces do not need to curve so sharply.

A second variation is that the duals could be pushed farther together. In particular, currently every quad face on the interface gets split into a geode-template. Quality is less than ideal where two interface quads have a small or large dihedral angle. This can occur even when the meshes are relatively structured; see Figure 9. It would be nice to "round off" these corners by pushing a dual surface into the opposite dual, so that the dual surface curves less. This method no longer produces pure templates. Figure 10 shows a two dimensional example of this.

Figure 9. Interface angles affect the quality of the geode-templates. Right, in three-dimensions it may be impossible to create a buffer layer: Suppose interface node V has six attached interface faces as in the figure. Since no point can see both of S_1 and S_2, one of the geode-templates attached to S1 or to S2 must have a non-positive jacobian at V.

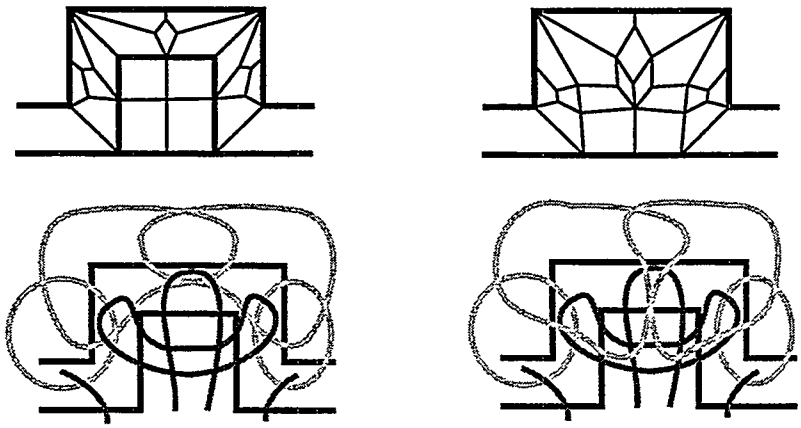


Figure 10. Pushing the arrangements so that they overlap further can improve quality, but modifies the pure templates. Left shows the pure templates. Right shows the result of puffing out two curves so that they are more round. Top shows the primal mesh, bottom shows the dual curves.

A third variation is to change the interface mesh: Split each interface quad into four triangles instead of two. The dual of the triangular mesh is still composed of circles and may be completed into spheres. I speculate that it is straightforward to carry out the geode-template construction steps. The biggest unknown is whether the resulting arrangement will dualize to a well-defined hex mesh. If not, then Mitchell[5] and Folwell[2] show how to fix-up the arrangement, but mesh quality will be poorer.

A fourth variation is to try to transition between two wildly different meshes, say two hex meshes with different element sizes that do not even conform node-wise. Completing the duals is the same as before. However, I would need to develop a new algorithm for pushing the duals together in a general way, and in many cases mesh fix-up as in Mitchell[5] and Folwell[2] would be required.

5.1 Why are Other Templates Hard?

In short, I designed the geode-template to be easy to mesh by keeping the dual curves of the boundary-mesh separate, so that no curve passes through a quad of both a diced-tri and a diced-quad. I also kept each dual curve *simple* (non-self-intersecting). In contrast, consider filling Schneiders's open problem,[9] Figure 11, with hexahedra. The surface mesh is identical to the variation of the geode-template with a pre-diced quad divided into four triangles, but without any sides. This problem, publicized several years ago, was difficult to fill at all, and no good-quality all-hex mesh is known. Figure 11 right shows the dual curves, one of which self-intersects eight times and passes through every quad of the boundary mesh. The octahedron has a boundary-mesh whose dual is just that curve.[1]

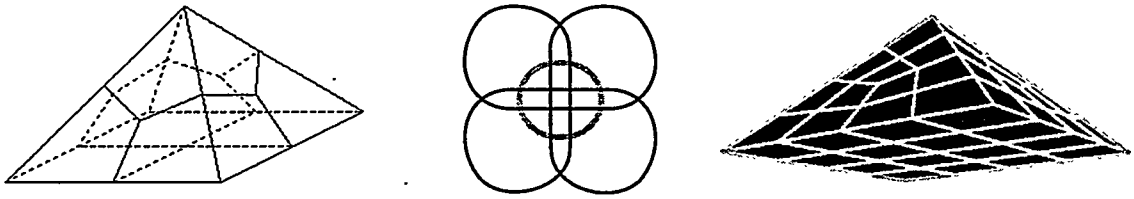


Figure 11. Schneiders's open problem.[9] No good all-hex mesh for the left figure is known. Center shows the two dual curves of the surface mesh laid flat. Right shows an underside-view of the problem after dicing the surface mesh; this version was meshed with the geode-algorithm.

On the CUBIT project, we have tried several pyramid-like templates. In these examples, unless sides like the geode-template's are introduced, the dual curves have a complicated self-intersection structure.

6. Conclusions

I have shown how to topologically conform a diced hexahedral mesh to a diced tetrahedral mesh by inserting an all-hex transition layer. The general method may be extensible to other transition problems. My results are practical, in that the algorithm is simple and the template can have good-quality hexes; at SNL, we are currently working on creating a geometrically-good transition layer for general boundaries.

Some interesting open problems remain. The geode-template indicates that, given the freedom to modify the surface mesh by dicing and the existence of a buffer layer, most surface meshes admit a well-shaped compatible hexahedral mesh. What if the surface mesh can not be diced or modified? Note that dicing gives a surface mesh whose node-edge graph is bipartite, which is sufficient to prove that a conforming hexahedral mesh exists, see Eppstein[1], but there is no guarantee that a good-quality mesh exists. Without dicing, the interface mesh may be composed of an odd number of quadrilaterals, which is impossible to fill with hexahedra.[5] Can the extent of dicing or modifying the surface mesh be limited in practical settings?

The buffer layer arises naturally as the dual of the intersection of the completed STC of the diced-tet-mesh with the completed STC of the diced-hex-mesh. Certain interface geometries do not admit a well-shaped buffer layer. One

interpretation of this is that the completed STC surfaces must curve too much. Could the STCs be pushed so that they overlap more, allowing more slowly curving surfaces?

Acknowledgments

I would like to thank Robert W. Leland and Timothy J. Tautges for suggesting this problem and developing its context. Thanks also go to Robert Schneiders and Christa Polaczek for discussing and publicizing Schneiders's open problem.[9]

References

[1] D. Eppstein, Linear Complexity Hexahedral Mesh Generation, *Proceedings, 12th Annual ACM Symposium on Computational Geometry*, Philadelphia, pp. 58-67, May 1996.

[2] N. Folwell and S. Mitchell, Reliable Whisker Weaving via Curve Contraction, submitted to *7th International Meshing Roundtable.*

[3] P. Knupp, Optimized-jacobian-Based Smoothing, in progress. pknupp@sandia.gov.

[4] R. Meyers and P.Tuchinski, The "Hex-Tet" Hex Dominant Meshing Algorithm as Implemented in CUBIT, submitted to *7th International Meshing Roundtable.*

[5] S. Mitchell, A Characterization of the Quadrilateral Meshes of a Surface Which Admit a Compatible Hexahedral Mesh of the Enclosed Volume, *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, Springer, pp. 465-476, 1996. Also http://endo.sandia.gov/~samitch/exist-abstract.html.

[6] P. Murdoch, The Spatial Twist Continuum: A Dual Representation of the All Hexahedral Finite Element Mesh, Ph.D. dissertation, Mech. Engr., Brigham Young University, December 1995.

[7] P. Murdoch and S. Benzley, The Spatial Twist Continuum, *Proceedings, 4th International Meshing Roundtable,* Sandia National Laboratories, pp. 243-251, October 1995.

[8] J. Robinson, CRE Method of Element Testing and the Jacobian Shape Parameters, Eng. Comput. Vol. 4, No. 2, p113-118, June 1987.

[9] R. Schneiders, An interesting open problem, http://www-users.informatik.rwth-aachen.de/~roberts/open.html.

[10] R. Leland, D. Melander, R. Meyers, S. Mitchell and T. Tautges, The Geode Algorithm: Combining Hex/Tet Plastering, Dicing and Transition Elements for Automatic, All-Hex Mesh Generation, submitted to *7th International Meshing Roundtable.*

[11] P. Tuchinsky, The Hex-Tet Hex Dominant Automesher: An Interim Progress Report, *Proceedings, 6th International Meshing Roundtable*, Park City, Utah, pp 183-13, October 1997.

Appendix A. Mesh Connectivity and Positions

This section describes how to reconstruct the geode-template. The following tables list the node positions and hex connectivity of a 24 x 14 x 24 geode-template centered at the origin.

Node	x	y	z	Node	x	y	z	Node	x	y	z
1	-12.0	7.0	12.0	17	12.0	7.0	0.0	33	3.9	-1.8	-3.6
2	12.0	7.0	-12.0	18	12.0	1.7	5.0	34	-3.9	-1.8	3.7
3	0.0	7.0	0.0	19	12.0	4.5	0.0	35	-4.1	-1.5	-4.1

Table 2: Node positions for a 24 x 14 x 24 geode-template centered at the origin.

Node	x	y	z	Node	x	y	z	Node	x	y	z
4	12.0	7.0	12.0	20	12.0	1.7	-5.0	36	-0.2	4.5	-5.3
5	0.0	7.0	12.0	21	12.0	-7.0	-12.0	37	-8.9	3.9	-6.5
6	12.0	7.0	0.0	22	0.0	-7.0	-12.0	38	-4.3	4.2	-2.3
7	4.6	7.0	4.6	23	5.0	1.7	-12.0	39	0.0	3.5	0.0
8	-12.0	7.0	-12.0	24	0.0	4.5	-12.0	40	4.4	4.3	2.3
9	-12.0	7.0	0.0	25	-5.0	1.7	-12.0	41	8.5	3.9	6.6
10	-12.0	-7.0	-12.0	26	0.0	7.0	-12.0	42	0.2	4.5	5.4
11	-12.0	-7.0	0.0	27	0.0	-7.0	12.0	43	4.1	-1.5	4.2
12	-12.0	1.7	-5.0	28	-5.0	1.7	12.0	44	5.7	0.7	0.7
13	-12.0	4.5	0.0	29	0.0	4.5	12.0	45	2.6	1.4	2.1
14	-12.0	1.7	5.0	30	5.0	1.7	12.0	46	0.0	0.0	0.0
15	-12.0	-7.0	12.0	31	0.0	-7.0	0.0	47	-2.6	1.4	-2.0
16	12.0	-7.0	12.0	32	-4.6	7.0	-4.6	48	-5.6	0.7	-0.6

Table 2: Node positions for a 24 x 14 x 24 geode-template centered at the origin.

Hex	n1	n2	n3	n4	n5	n6	n7	n8	Hex	n1	n2	n3	n4	n5	n6	n7	n8
1	17	20	2	21	31	33	23	22	14	48	13	12	35	34	14	11	31
2	43	18	17	31	44	19	20	33	15	28	27	15	1	34	31	11	14
3	43	30	27	31	18	4	16	17	16	41	5	1	14	42	29	28	34
4	36	24	26	37	33	23	2	20	17	38	32	9	13	39	3	1	14
5	37	26	32	38	20	2	3	39	18	40	7	3	39	41	5	1	14
6	19	6	2	20	40	7	3	39	19	46	39	14	34	45	40	41	42
7	45	40	19	44	46	39	20	33	20	47	38	13	48	46	39	14	34
8	46	39	20	33	47	38	37	36	21	36	24	25	35	37	26	8	12
9	33	23	22	31	36	24	25	35	22	47	38	37	36	48	13	12	35
10	48	47	46	34	35	36	33	31	23	37	26	8	12	38	32	9	13
11	34	46	45	42	31	33	44	43	24	41	5	4	18	40	7	6	19
12	31	22	10	11	35	25	8	12	25	42	41	18	43	45	40	19	44
13	42	29	28	34	43	30	27	31	26	42	29	30	43	41	5	4	18

Table 3: Connectivity of a geode-template: Nodes in hexes.

Triangular Meshing

Strategies for Nonobtuse Boundary Delaunay Triangulations

Michael Murphy* Carl W. Gable†

Abstract. *Delaunay Triangulations with nonobtuse triangles at the boundaries satisfy a minimal requirement for Control Volume meshes. We motivate this quality requirement, discuss it in context with others that have been proposed, and give point placement strategies that generate the fewest or close to the fewest number of Steiner points needed to satisfy it for a particular problem instance. The advantage is that this strategy places a number of Steiner points proportional to the combinatorial size of the input rather than the local feature size, resulting in far fewer points in many cases.*

1 Introduction

Techniques for numerically approximating the solution to partial differential equations typically have at least three distinct phases: *mesh generation*, where the domain is partitioned into a finite number of pieces; *discretization*, which takes the mesh and derives a system of linear equations, $Ax = b$, whose solution can be used to obtain an approximation to a PDE over the domain; and the *solution* phase, where the system of linear equations is solved[1]. The requirements of one phase can place constraints on the output of another. A frequent requirement of the solution phase is to restrict the matrix A to a class of matrices so that linear solution techniques which exploit special properties of this class to obtain fast, accurate, and stable performance can be employed. To satisfy this requirement, the discretization phase, which generates the matrix, often must impose geometric restrictions on the mesh. In this paper, we study the requirements of the *Control Volume* discretization technique. Our goal is to obtain algorithms that satisfy such geometric restrictions in the mesh generation phase.

We first give a geometric interpretation of the Control Volume Method, which will help motivate the need for mesh quality at some minimal level. We then argue why satisfying these minimal quality requirements alone can in some real world settings be more useful than satisfying them in conjunction with stricter quality requirements that have been proposed in the literature. Finally, we present algorithms to guarantee the minimal level of mesh quality necessary to perform computations on Delaunay triangulations of polygonal domains by adding either the fewest or close to the fewest number of Steiner points (*i.e.*, nodes placed to obtain quality triangles) necessary for a particular problem instance; we also address the difficulties that arise in more complicated geometric settings such as Planar Straight Line Graphs (PSLGs) (see [BE92] for a definition) and Piecewise Linear Complexes (PLCs) (see [MTT+96] for a definition) in higher dimensions.

2 The Control Volume Method

2.1 Overview

The Control Volume Method, also known as the "Control Volume Finite Element Method,"[For91], "Box Method," [BR87]) "Finite Volume," and "Integrated Finite Differences,"using "PEBI grids"[HBMC91], is a discretization technique for Delaunay meshes[2]. Regardless of the dimension, it discretizes a Delaunay mesh with respect to its edges. For each edge $e = (i, j)$ in the mesh, the Control Volume method creates an

*Department of Computer Science, University of Maryland—College Park and Los Alamos National Laboratory. Email: murphy@lanl.gov

†Geoanalysis Group, Earth and Environmental Science Division, Los Alamos National Laboratory. Email: gable@lanl.gov

[1]This is a simplification. Indeed, many variations exist depending on the application. For example, for time dependent applications this can be an iterative process—the output from the solver may induce changes in the mesh forcing a new discretization and thus a new system of equations to be solved. However, for our purposes, this view is complete.

[2]Control Volume can also be used for non-Delaunay meshes, but that is beyond the scope of this paper.

entry in the ith row and jth column in the matrix A associated with system of linear equations $Ax = b$ approximating the PDE to be solved (and vice versa —the matrix A is symmetric). The entry, call it $A_{i,j}$ is non–zero (except when there are degeneracies arising from co–circularity) because it is associated with the length (area in three dimensions and volume in higher dimensions) of the Voronoi face, $V_{i,j}$, separating i and j. The entry $A_{i,j}$ is computed by dividing $V_{i,j}$ by the the length of edge e; however, it is intuitive for the purposes of understanding quality requirements to ignore the division by the length of e and focus on the quantity $V_{i,j}$. For an edge in a constrained or conforming triangulation in a planar domain, (which we will work with from now on unless otherwise stated) $V_{i,j}$ can be computed as follows. If e is an internal edge, then because T is a triangulation, there are exactly two triangles incident upon e. The length of the segment formed by connecting the circumcenters of these two triangles c_1, c_2 is $V_{i,j}$. If e is a boundary edge, then there is only one triangle incident upon e and the length of the Voronoi edge is infinite. Thus, the length is truncated at the domain boundary by setting $V_{i,j}$ equal to the length of the line segment connecting the circumcenter of the incident triangle to the midpoint of e. This is illustrated in Figure 1.

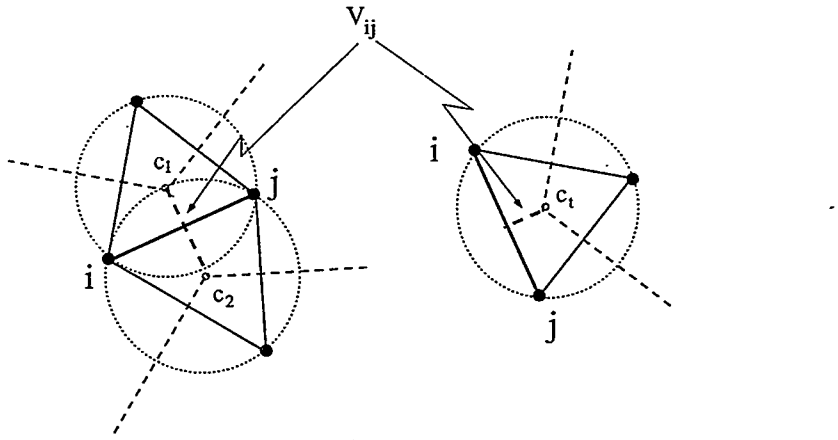


Figure 1: The entry $A_{i,j}$ in the matrix A corresponding to edge $e = (i, j)$ is associated with the length of the edge of the Voronoi face, $V_{i,j}$, shared by edge (i, j). When (i, j) is a boundary edge (right), this length is the length of the line segment with endpoints at the bisector of (i, j) and the circumcenter of the unique triangle incident upon (i, j).

2.2 Minimal Quality Requirements for the Control Volume Technique

Although it is well known that the Delaunay Triangulation is optimal with respect to many interesting criteria[Law77] [Raj94] [Mus97], not all Delaunay meshes are appropriate for the purpose of solving PDEs with this discretization technique[For91]. Indeed, the *quality* of a triangulation has been a major research topic in mesh generation, resulting in many definitions. This diversity stems in part from the variety of applications, differing levels of expectations about what can be obtained from a mesh, and requirements of competing discretization methods and solvers. It is not our intention to attempt the authoritative definition. However, by restricting our attention to the Control Volume setting, a minimal requirement for a Delaunay mesh to be suitable for numerical computation can be precisely stated. That is, from the solver's perspective, it is required that the matrix A in the associated linear system $Ax = b$ be an M–matrix. M–matrices are square matrices whose off-diagonal elements are either zero or negative and whose diagonal elements are strictly positive. M–matrices are also non-singular and have strictly positive inverses. As it turns out, the diagonal element $A_{i,i}$ in the control volume discretization is the sum of the absolute values of the other entries in row i, causing it to be diagonally dominant, implying (with some minor additional assumptions) the latter two criteria via standard theorems in numerical analysis. Consequently, we need only focus on obtaining negative off diagonal entries in the manner explained below. With that, the solution of the resulting system of linear equations using an iterative technique is possible, allowing a tremendous advantages in accuracy, stability, and running time over direct solvers; see [Saa96] for more background on M–matrices and iterative linear solution techniques.

The M–matrix restriction also avoids physically nonsensical results. To see why, consider a general model

problem that could be solved using this discretization technique. The equation given by $F = -\nabla G$, where F is the flux, governs the diffusion of heat if G corresponds to temperature or fluid flow in a porous material if G corresponds to pressure. Energy (fluid) flows from high to low temperature (pressure) so the equation has a minus sign to account for heat (fluid) flowing opposite the gradient. When constructing the geometric coefficients for the discrete form of the equations, the minus sign is associated with the geometric terms. The requirement that all the geometric coefficients are negative insures there is no non-physical transport up gradient.

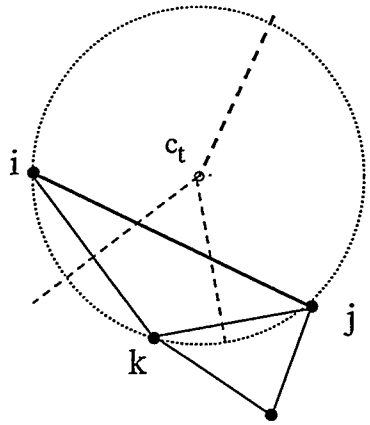


Figure 2: An obtuse boundary triangle in a Delaunay mesh. The edge (i, j) is a boundary. The circumcenter c_t of the triangle (i, j, k) falls outside the domain. Situations like this lead to geometrically nonsensical results and fail to give an M–matrix when using the Control Volume discretization scheme.

To translate the M–matrix requirement of the solver into a geometric restriction for the mesh generator, we must first point out a subtlety of the discretization technique. Since the non-diagonal elements of an M–matrix should be zero or negative, the length of the Voronoi segments associated with edges, used in the matrix entry computation, should be a signed quantity. When computing this signed length, it is necessary to use the orientation of the triangles incident upon it in the following manner. Suppose that edge $e = (i, j)$ is incident upon a triangle $t = (i, j, k)$ with circumcenter c_t. Then we compute the signed distance from c_t to the midpoint of e. If c_t is on the same side of e as k (the other vertex of the triangle), then this distance is negative, by convention. Otherwise, it is non-negative. For an external boundary edge, if the circumcenter is outside the domain then the length of the circumcenter to the bisector must be positive with this convention; an example is shown in Figure 2. However, this violates the M–matrix requirement. For an internal edge in a Delaunay mesh, the sum of the contributions of the two triangles incident upon it will always be non–positive, satisfying the M–matrix requirement, even if one of the circumcenters falls outside the domain. Nevertheless, circumcenters outside the domain in any case are undesirable[3]. Fortunately, as a consequence of the following straightforward theorem, to obtain an M–matrix with the Control Volume discretization we need only focus on triangles incident upon boundary edges.

Theorem 1 *In a Delaunay Triangulation, if the circumcenters of each triangle with an edge incident upon the boundary are contained inside the boundary, then the circumcenter of every triangle in the triangulation is contained inside the boundary.*

A proof is given in[CFH93].

We investigate algorithms that satisfy Theorem 1 by adding points to insure that each triangle incident upon a boundary edge is nonobtuse because a nonobtuse triangle contains its circumcenter.

[3]There are point sets where the circumcenter of every triangle in the Delaunay Triangulation of these sets falls outside the boundary. For example, a finite point set sampled from the "moment curve" in the plane parameterized by (t, t^2).

3 Related Work

Recently, quality requirements imposed by the Control Volume method have been discussed in the theoretical computer science community [MTTW95] (and followed up in [MTT+96] and [MTTW98]). Furthermore, the work of [BMR94] is motivated in part by the problem of generating quality meshes applicable to the Control Volume method. Finally, Rivara and Hitschfeld [HR97] recently have addressed a problem very similar to the one we consider. We review these works below.

3.1 The Radius–Edge Condition

One might ask why we are focusing upon avoiding a "show stopper" by requiring only that the mesh yield a discretization resulting in an M–matrix when algorithms have been given to satisfy loftier quality requirements that not only guarantee convergence but also improve the rate of convergence. Specifically, a recent result in quality Control Volume meshing [MTTW95] states that the Control Volume method will converge at a rate depending on the the largest ratio of the circumradius of each triangle to its shortest edge (abbreviated "radius–edge"). The smaller this ratio, the better. Shewchuk explores in his thesis [She97] how Ruppert's Delaunay Refinement algorithm [Rup93] can be used to achieve such a bounded ratio for every triangle in the triangulation.

To see why this quality measure can be too restrictive for some applications, it is worthwhile to examine some rather extreme yet real-world, applications that we face. One such application is modeling fluid flow in geological systems. A rock layer may extend for many kilometers yet may only be a few meters thick. Thus, high aspect ratios are the norm rather than an occasional nuisance. Another real world situation where very high aspect ratios arise is in semiconductor modeling when the ability to model the effect of thin layers of film placed in the semiconductors is desired. In these situations, more stringent quality triangulation requirements can produce unsatisfactory results because the number of triangles required to satisfy them is too large. For example, in two dimensions, the bounded radius–edge ratio quality measure requires that element size be on the order of the local feature size (see [Rup93] for a definition). However, any algorithm that outputs elements on the order of the local feature size will generate entirely too many elements in these high aspect ratio settings because the local feature size is small everywhere[4].

In contrast, the quality requirements we impose, although far less stringent, require far fewer Steiner points to satisfy them. Indeed, we can obtain "combinatorial results" — those that show that the number of Steiner points depends on the combinatorial size of the initial input (*i.e.*, the number of points and line segments), rather than local feature size, a geometric measure.

3.2 Nonobtuse Triangulation of Polygonal Domains

The work of Bern, Mitchell, and Ruppert [BMR94], which shows how to triangulate a polygonal domain with nonobtuse triangles using $O(n)$ Steiner points (where n is the number of vertices in the polygon) is a prime example of a combinatorial result applicable to Control Volume meshing. Nonobtuse triangles are appropriate for the Control Volume method because they imply that the Voronoi face associated with an edge e, crosses e. They are also interesting for general interpolation problems because they imply that for any point inside a triangle, its nearest neighbor in the set of all mesh vertices to that point is one of the triangle vertices. Meshes with bounded radius–edge ratios, or any triangulation with no small angles, do not have these properties unless the guarantee says that no angle will be less than 45 degrees or the bounded edge ratio is $\leq \sqrt{2}$; guarantees at this level are beyond the current state of the art.

There are some differences in our approaches. As we only require the boundary elements to be nonobtuse, we can use far fewer Steiner points. Bern, Mitchell, and Ruppert report that in practice, their algorithm generates roughly $25n$ Steiner points per instance. In contrast, our algorithms generate the fewest or close to the fewest number of Steiner points needed *for a particular problem instance.* In the worst case, this can be bounded at $3n$, for the reasons that one can never pack more than three obtuse angles around a vertex and that it is possible to place a Steiner point to resolve an obtuse angle without creating a new obtuse angle in the Delaunay Triangulation of the point set. In situations where we are not as concerned about the

[4] It is also worthwhile to note that these high aspect ratio problems tend to frustrate many mesh generation heuristics as well.

quality of interior triangles, or when we wish to have no interior points at all, our approach can be more useful.

3.3 Nonobtuse Boundary Triangulations

Hitschfeld and Rivara[HR97][HR98], have recently addressed a very similar problem. They note that no–small–angle quality Delaunay triangulation algorithms which guarantee that all angles are larger that 30 degrees, can allow obtuse boundary angles. They present an algorithm to be used as a "post–processing step" after such a quality triangulation algorithm on a polygonal domain is used to remove the obtuse boundary triangles; the postprocessing algorithm makes the explicit assumption that all angles in the triangulation not defined by two boundary edges are between 30 and 120 degrees and that the triangulation is constrained Delaunay. We simplify this algorithm so that it places fewer Steiner points and does not make any assumptions about the angles in the input triangulation. However, the price of our simplification is that a no–small–angle triangulation is not maintained. We dub this algorithm **Project–Flip**. Hints of a similar algorithm can be found in[For91].

PROJECT–FLIP
1 **input** : A Constrained or Conforming Delaunay Triangulation of a Polygonal Domain.
2 **while** there exists an obtuse triangle $t = (a, b, c)$ with boundary edge $e = (a, b)$ opposite the obtuse angle **do**
3 Let d be the orthogonal projection of c onto e into the triangulation.
4 Remove t from the triangulation
5 Create two new triangles (a, c, d) and (b, c, d)
6 Restore Delaunay Triangulation via Flip algorithm[Law77]
7 **end-while**

Project-Flip has the following properties:

- It terminates and produces triangles so that the circumcenters of each triangle are contained in the boundary.
- The number of points inserted can be bounded by $O(n)$.
- While it is useful for constrained or conforming Delaunay Triangulations of polygons, it cannot be generalized to arbitrary PSLGs.

The Project-Flip algorithm is not optimal in the sense that it will not place the minimum number of points in a particular instance. This will be shown below. Consequently, we wish to explore methods which are sensitive to the problem instance.

4 Nonobtuse Boundary Triangles with Few Steiner Points

4.1 Nonobtuse Triangles on a Boundary Edge

Let us first consider the following related problem. Suppose we are given a line segment in the plane denoted by edge $e = [a, b]$, $a, b, \in R^2$, and a set of points S all contained in a semicircle of the diametrical circle of e (*i.e.* the circle with diameter e). We wish to refine e into $k+1$ into non-overlapping subintervals $[a, s_1), \ldots, [s_{k-1}, s_k), [s_k, b]$, by placing Steiner points $S_e = \{s_1, \ldots, s_k\}$ along e such that the open diametrical circle of each subinterval is empty in the sense that it contains no point of S in its interior. This would guarantee that no circumcenters in the Delaunay Triangulation of the point set $S \cup \{a, b\} \cup S_e$ would fall on the other side of e. Equivalently, all the angles opposite the subintervals in the Delaunay Triangulation would be nonobtuse. Ideally, the number of Steiner points that we choose, k, should be the fewest number of Steiner points needed for a particular instance.

One trivial algorithm takes the orthogonal projection of all points inside e's diametrical circle onto e and uses these as the Steiner points that induce the subintervals. However, this can add too many points because each time we add a Steiner point to refine e, the area covered by the diametrical circles on each subsegment of e

decreases. Thus, some points that were initially contained in e's diametrical circle may no longer reside inside the diametrical circles of the newly created subintervals and therefore can be omitted from consideration. Indeed, the addition of one Steiner point may be all that is necessary. Although worst case optimal, it is not sensitive to the size of the output required for a particular instance and is thus unsatisfactory.

The Project-Flip algorithm could also be applied to this problem. By first taking the Delaunay Triangulation of the point set (in which e would be a boundary edge) the algorithm would terminate with no obtuse angles. In some cases, it would place far fewer points on e than the straightforward orthogonal projection algorithm described above. However, as shown in Figure 3, it does not place the fewest Steiner points for a particular instance. Moreover, there exists similar examples, identified by Mount[Mou97], which shows that the running time can be $O(n^2)$.

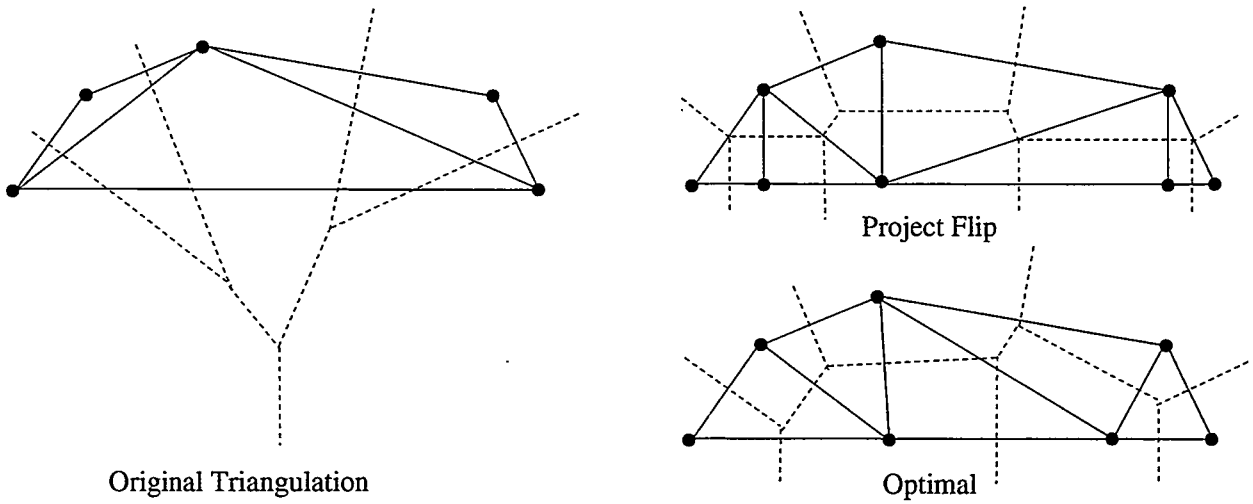


Figure 3: Project-Flip, which in this case has the same output as simple orthogonal projection of all points inside e's diametrical circle onto e, can be suboptimal in terms of the number of Steiner points added to obtain nonobtuse boundary triangles.

It turns out that a simple greedy strategy, illustrated in Figure 4, gives an optimal solution to this easier problem.

GREEDY–REFINE(e,S)
1 **input** : An edge $e = [a, b]$ and a set of points S all contained in a semicircle of the diametrical circle of e.
2 Starting at a find the largest subsegment of e, $e' = [a, a')$ such that the open diametrical disk of e' contains no point of S
3 **if** $a' \neq b$ **then**
4 Refine e by placing a Steiner point at a'.
5 Let S' be the points in the diametrical circle of $[a', b]$
6 GREEDY–REFINE ($[a', b]$,S');
7 **end-if**

Theorem 2 *The greedy algorithm for an edge generates the fewest number of Steiner points required for a particular instance.*

Proof: The greedy algorithm's optimality can be proven by induction on the minimum number of Steiner points k needed for a particular interval. That is, suppose we are given an edge $e = [a, b]$ and a set S such that k Steiner points are required to refine e so that it satisfies the empty diametrical circle criterion. Obviously, $k \leq n$, for as noted, orthogonally projecting each point in S onto e would do the job. The base case, $k = 0$ is trivial.

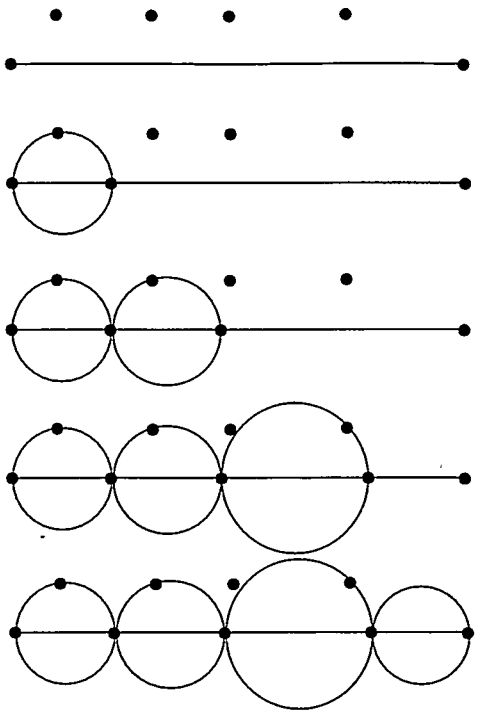


Figure 4: Execution of the greedy algorithm for a line segment, e. It starts at an endpoint and grabs the largest empty circle it can get.

Without loss of generality, transform the plane so that the edge (a, b) coincides with the interval [0,1] along the x-axis. Among all optimal Steiner placements achieving k points, consider the one whose leftmost Steiner point is as far right as possible (*i.e.*, has the largest x-coordinate). More formally let x_0 be the *supremum* of the set of initial optimal Steiner placements. Let x_g be the leftmost endpoint of the greedy triangulation. We claim that $x_g = x_0$. Suppose not. If $x_g < x_0$, then the closed diametrical circle bounded at x_g is interior to the circle bounded at x_0 and hence would be further expanded by the algorithm, a contradiction. On the other hand, if $x_g > x_0$, then we can modify the optimum point placement strategy so that it uses the same number of Steiner points. This is done by shrinking the second optimum disk by moving its leftmost point to x_g. The modified second disk is contained in the original second disk and consequently remains empty. This contradicts the hypothesis on x_0.

4.2 Convex Polygon

We now consider generalizing our problem to a slightly more complex situation. Given a convex polygon P, we wish to decompose each polygonal edge into subsegments that satisfy the point–free diametrical circle property. This problem cannot be solved by applying the above greedy algorithm for edges to each polygonal edge individually, using the vertices of the P inside the diametrical circle as the set of points. This is because *interference* can arise. It is possible to deem an edge's diametrical circle point–free only later to violate the diametrical circle when adding a Steiner point on another edge. This leads to conflicts and suboptimality, illustrated in Figure 5.

A simple modification to the greedy algorithm places the fewest number of Steiner points. For an edge e that has a nonempty diametrical circle, we need to be sensitive to the regions on e in which adding a Steiner point will create problems for other edges. That is, those regions where the diametrical disks of other edges intersect e. We call these the "forbidden intervals" of e. The idea behind this "sensitive greedy strategy" is to start at an endpoint and find the largest segment whose diametrical circle is empty, with the added constraint that an endpoint is never placed inside a forbidden interval. Specifically, if the greedy strategy

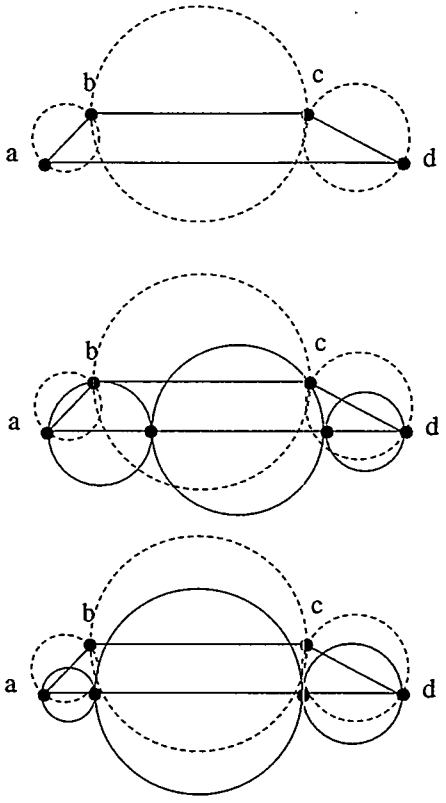


Figure 5: In the top figure edges (a,b), (b,c) and (c,d) have point–free diametrical circles and edge (a,d) is in need of refinement. The middle figure illustrates that the greedy algorithm for edges is deficient in the sense that it can add Steiner points that violate previously point–free diametrical circles. The solution is to never add a Steiner point that violates a diametrical circle of another edge, regardless of whether that diametrical circle is empty or not.

places a point inside a forbidden interval, then place the Steiner point at the start of the forbidden interval.

SENSITIVE–GREEDY–REFINE(P)
1 **input** : A convex polygon P
2 Let E be the list of edges of P without point–free diametrical circle property w.r.t. the vertices of P
3 **while** E is not empty **do**
4 Let $e = [a,b]$ be an edge in E
5 Remove e from E
6 Starting at a find the largest subsegment of e, $e' = [a,a')$ such that:
The open diametrical disk of e' contains no point of S and
a' is not contained in the open diametrical disk of another edge f.
7 Refine e by placing a Steiner point at a'.
8 **if** the diametrical circle of $[a',b]$ is not point–free **then**
9 Add subsegment $[a',b]$ to E
10 **end-if**
11 **end-while**

Theorem 3 *For a convex polygon, the sensitive greedy strategy places the fewest Steiner points to obtain nonobtuse boundary triangles.*

Before proving this, we need the following two lemmas:

Lemma 1 *The diametrical circles of the forbidden regions on an edge are point-free.*

The proof follows immediately from the convexity of the polygon.

Lemma 2 *The forbidden regions on an edge do not overlap.*

Proof Sketch: Let f and g be two edges on a convex polygon P whose open diametrical circles overlap. We consider the lune, the overlapping region, of the two open circles. If f and g share a unique common vertex, then one apex of the lune is at this common vertex. We now analyze the location of the other apex and its relation to e. Let h be the segment connecting the two vertices not common to f and g. Notice that due to convexity, h must either be e or be contained in the halfspace of e containing P. The other apex is on the orthogonal projection of the common vertex onto the line containing h. Consequently, it is either contained in the same halfspace of e as P, if it falls onto h or it is outside the domain if the orthogonal projection does not fall onto h because f and g are boundary edges. In the former case, since we are considering open diametrical circles, this single point is the gap. In the latter case, it implies by convexity that e cannot intersect the lune at all. In case f and g do not share a common vertex, then it is possible to show that the lune is contained in halfspaces of the lines containing the two segments connecting e and f to form a convex quadrilateral. Since both of these segments must be contained in a halfspace of e, it implies that e cannot intersect the lune at all. □

Proof (Of Theorem 3): Let e be an edge which is under refinement. Let f be an edge whose diametrical circle intersects e. Two cases can arise: (1) the diametrical circle of f is empty and (2) the diametrical circle of f is non-empty.

In case (1), placing a point in this forbidden interval will violate f, since f's diametrical circle is empty. Would one benefit from refining in the forbidden region of f with respect to e? The answer is no. Placing a point in the forbidden region of e with respect to f will necessarily induce a Steiner point on f to cope with the one we just added. However, we could spend two Steiner points (one from refining e and the one placed on f to cope with the one placed on e) at least as well, if not better, by just refining e and leaving f alone. Two Steiner points placed on e are enough to skip over the forbidden region of e with respect to f and create an empty diametrical circle on e: place one Steiner point at the beginning of the forbidden region of f and the other as far as possible along e so that it does not violate another forbidden region and creates a subsegment with an empty diametrical circle. By Lemma 1, we know that the second point will be beyond the forbidden region of f with respect to e and consequently will leave f alone.

Again in case (2), one can never benefit from refining in a forbidden region. This is because if f's diametrical circle is not empty and intersects e, then one of the endpoints of e is one of the endpoints of the forbidden region of e with respect of f as well. Since by Lemma 1, the diametrical circle of this forbidden region is empty, we can simply place a Steiner point at the start of the forbidden region. □

4.3 Set of Points

Given a set of points S in the plane, we wish to refine each edge of the convex hull of S so that the Delaunay triangulation of S taken with the augmented points has no obtuse angles opposite the boundaries. One interpretation of this criterion is that the nearest neighbor in S and the augmented points to a point outside the convex hull should be one of the vertices of the convex hull edge closest to that point. To achieve this, we divide the convex hull edges into two groups: those that have empty diametrical circles and those that do not. The strategy is never to violate an empty diametrical circle of another edge when refining an edge, as in the case of the convex polygon. However, the strategy changes when dealing with a non–empty diametrical circle of another edge. The answer is to refine only if it greedy strategy gets snagged on a point internal to the convex hull. If it gets snagged on a boundary point, the strategy is to refine the other violated edge first. This insures that a Steiner point is never needed to be placed to resolve an obtuse angle around a previously added Steiner point. We do not have an optimality proof at this time, although we can be sure that in many situations, this will place fewer Steiner points than Project-Flip.

4.4 Non–convex polygonal domains

The above algorithms can be generalized to non–convex polygons, with or without interior points, by changing the notion of forbidden regions slightly to take into account visiblity. This allows us to obtain constrained Delaunay Triangulations with nonobtuse boundary triangles, but not conforming ones.

5 The Difficulty of More Complicated Domains

5.1 Planar Straight Line Graph

The problem of nonobtuse boundary triangulations is significantly more difficult on a PSLG, which allows for much more complicated input geometries with multimaterial interfaces. The problem is to ensure that for each edge on a given PSLG, its diametrical circle is empty. As a solution to this problem implies the solution to the problem of conforming Delaunay Triangulation, the lower bound of $\Omega(n^2)$ Steiner points given in [ET93] applies. However, it is not even known if the number of Steiner points required to satisfy the edges of a PSLG can be bounded by a polynomial function of the input size. Work with a similar spirit can also be found in the study of triangulations that have no large angles[Mit93][Tan96]. The goal in these studies is to refine a triangulation using a number of Steiner points as close to $\Omega(n^2)$ as possible so that the largest angle in the triangulation is bound by some constant. Of course, the closer to nonobtuse this angle is, the better. We suggest relaxing the problem so that the angles opposite boundary edges are the only ones taken into consideration. However, many of the difficulties faced in these problems still apply. The salient problem is that the interference caused by refining one edge can have global reprocussions. Mitchell's paper[Mit93] contains some very interesting examples of the global problems encountered.

Recently, Rivara and Hitschfeld [HR98] have announced that they can solve this problem on constrained Delaunay triangulated PSLGs if all angles not constrained by two boundary edges are bounded between 30 and 120 degrees. We will refer to these as "quality" PSLGs. However, their technique, which appears very practical, falls prey to some "worst case" scenarios that invalidate their general claims about its performance. Specifically, Proposition 3 (p. 18) which states that a such a quality PSLG with k obtuse boundary triangles needs only $O(k)$ Steiner points to make the PSLG satisfy the nonobtuse boundary property. We present figure 6 as evidence that this is incorrect. In this counterexample, one bad triangle can induce as many as $\Omega(n)$ Steiner points to satisfy the nonobtuse boundary Delaunay property, where n is the total number of triangles. Furthermore, it is not even clear if their algorithm will terminate on all such quality PSLGs. This illustrates the insidious nature of the the global interference problem, and the resulting propogation of Steiner points, which makes this problem so difficult in general.

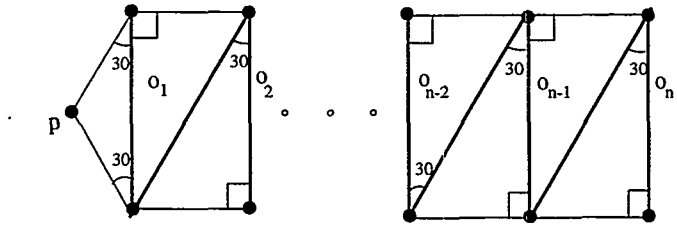


Figure 6: A PSLG satisfying all the angle critera assumed by Rivara and Hitschfeld's method [HR98] that contradicts their Proposition 3 (p. 18). The leftmost triangle has a 120 degree angle at vertex p and is the only obtuse triangle in the triangulation of the PSLG, whose edges are highlighted. Refining edge o_1 to remove the obtuse angle will necessarily induce a total of $\Omega(n)$ Steiner points, one on each of the highlighted edges.

5.2 M–Matrices in Three Dimensions

As is the case in the plane, not all three dimensional Delaunay Triangulations will yield a M–matrix under the Control Volume Method[Let92]. The problem of obtaining an M–matrix in three dimensions can be

geometrically formulated most generally as follows: Given a PLC, ensure that the diametrical sphere of each object contains no other vertex. For a line segment in three dimensions, not part of any facet, the diametrical sphere is simply the sphere whose diameter is the line segment. For a triangular facet, the diametrical sphere is the minimum–diameter sphere containing that triangular facet. This can be obtained by computing the circumcircle of the triangular facet and expanding it into a sphere with the same radius. Using the three–dimensional version of Theorem 1 (also proven in [CFH93]), facets refined to satisfy these conditions will result in an M–matrix using the Control Volume discretization technique in three dimensions. Unfortunately, this implies a solution to Conforming Delaunay Triangulation, an "easier" problem for which a general point placement strategy which terminates correctly for PLCs in dimensions higher than two is not known to exist.

Acknowledgements

The first author thanks Professor David Mount for many useful discussions.

References

[BE92] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 1992.

[BMR94] M. Bern, S. Mitchell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 221–230, 1994.

[BR87] R. Bank and D. Rose. Some error estimates for the box method. *SIAM J. Numer. Analysis*, 24:777–787, 1987.

[CFH93] P. Conti, W. Fichtner, and N. Hitschfeld. Mixed elements trees: A generalization of modified octrees for the generation of meshes for the simulation of complex 3-d semiconductor devices. *IEEE Trans. on CAD/ICAS*, 12(11):1714–1725, November 1993.

[ET93] H. Edelsbrunner and T. S. Tan. An upper bound for conforming Delaunay triangulations. *Discrete Comput. Geom.*, 10(2):197–213, 1993.

[For91] P.A. Forsyth. A control volume finite element approach to NAPL groundwater contamination. *SIAM J. Sci. Stat. Comput.*, 12:1029–1057, 1991.

[HBMC91] Z.E. Heinemann, C.W. Brand, M. Munka, and Y.M. Chen. Modeling reservoir geometry with irregular grids. *SPE Reservoir Engineering*, pages 225–232, 1991.

[HR97] N. Hitschfeld and M. C. Rivara. Non-obtuse boundary Delaunay triangulations. In *6th Intl. Meshing Roundtable*, 1997. http://www.andrew.cmu.edu/user/sowen/imr6.html.

[HR98] N. Hitschfeld and M. C. Rivara. Automatic creation of non-obtuse boundary Delaunay triangulations. Technical Report TR DCC-98-2, Dept. Comput. Sci., University of Chile, Santiago, Chile, 1998. http://www.dcc.uchile.cl/~nancy/.

[Law77] C. L. Lawson. Software for C^1 surface interpolation. In J. R. Rice, editor, *Math. Software III*, pages 161–194, New York, NY, 1977. Academic Press.

[Let92] F. Letniowski. Three-dimensional Delaunay triangulation for finite element approximations to a second-order diffusion operator. *SIAM J. Sci. Stat. Comput.*, 13:765–770, 1992.

[Mit93] Scott A. Mitchell. Refining a triangulation of a planar straight-line graph to eliminate large angles. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, pages 583–591, 1993.

[Mou97] D.M. Mount, 1997. Personal Communications.

[MTT+96] G.L. Miller, D. Talmor, S.-H. Teng, N. Walkington, and H. Wang. Control volume meshes using sphere packing: Generation, refinement, and coarsening. In *Proc. 5th International Meshing Roundtable*, Albuquerque, NM, 1996. Sandia National Laboratories.

[MTTW95] G. L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. A Delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 683–692, 1995.

[MTTW98] G.L. Miller, D. Talmor, S.-H. Teng, and N. Walkington. On the radius edge condition in the control volume method. *SIAM J. Numer. Analysis*, 1998. Submitted for publication. Preprint available at http://www.cs.cmu.edu/afs/cs.cmu.edu/user/tdafna/www/publications.html.

[Mus97] O.R. Musin. Properties of the Delaunay triangulation. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 424–426, 1997.

[Raj94] V.T. Rajan. Optimality of the Delaunay triangulation in R^d. *Disc. and Comput. Geom.*, 12:189–202, 1994.

[Rup93] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 83–92, 1993.

[Saa96] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston,MA, 1996.

[She97] J. R. Shewchuk. *Delaunay refinement mesh generation.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1997. Available at http://www.cs.cmu.edu/~jrs/jrspapers.html.

[Tan96] T.-S. Tan. An optimal bound for high quality conforming triangulations. *Discrete Comput. Geom.*, 15:169–193, 1996.

A priori Delaunay-conformity.

Philippe P. PÉBAY
INRIA, Gamma Project, Rocquencourt
BP 105, 78153 Le Chesnay Cedex, France.
INSA, L2MCS, Bât. 401 - Avenue A. Einstein,
69621 Villeurbanne Cedex, France.
E-mail : Philippe.Pebay@inria.fr

Pascal J. FREY
INRIA, Gamma Project,
Domaine de Voluceau, Rocquencourt,
BP 105, 78153 Le Chesnay Cedex, France
E-mail : Pascal.Frey@inria.fr.

Abstract. *This communication presents a method for redefining* a priori *a field of constraints represented in two dimensions by a set of edges, in three dimensions by a surface triangulation. The aim is to provide a resulting constraint, strongly Delaunay-conforming, (*i.e.*, that will be built by any Delaunay triangulation of the convex hull of the associated set of vertices). We show that the two-dimensional problem can be easily solved, and hence, we give a classification of edges leading to a convergent algorithm. Although the classification extends to faces, an essential property vanishes in three dimensions, excluding a simple extrapolation of the method. Nevertheless, a heuristic algorithm, based on face subdivisions, governed by a geometric estimate and by means of edge swapping is given in three dimensions. Several examples emphasize the proposed method in two and three dimensions.*

Keywords. Triangulation, Delaunay triangulation, constrained triangulation, surface triangulation.

Introduction

Numerical simulations based on finite element methods for arbitrarily shaped three-dimensional domains Ω, require at first a mesh $\mathcal{T}_\Omega$ (*i.e.*, a polytopal covering-up of Ω). Such domains Ω are usually represented through a discretization (*e.g.*, a triangulation) of the domain boundary $\partial\Omega$, that forms the constraint Γ. More precisely, the constraint is defined as a list of edges and possibly faces which usually includes a boundary-constraint (*i.e.*, a curve or surface representation approached by a polygonal or polytopal covering-up). Additional entities (*e.g.*, interfaces) may also be supplied, that have to explicitly appear in the triangulation. The set of vertices $\mathcal{S}$ contains at least the edge and face vertices of Γ.

In unstructured mesh generation, mesh elements provided by a Delaunay-type method have usually nice properties. However, if the input data is composed of a set of edges and faces Γ (*i.e.*, the constraint), of which the vertices form a subset $\mathcal{S}(\Gamma)$ of the set of vertices $\mathcal{S}$, a Delaunay triangulation $\mathcal{T}$ of $\mathrm{Conv}(\mathcal{S})$ does not necessarily contain all the faces of Γ. In other words, a Delaunay triangulation algorithm of $\mathcal{S}$ does not necessarily lead to a mesh $\mathcal{T}_\Omega$ of $\mathrm{Conv}(\Omega)$ [1] complying with the constraint Γ. The envisaged numerical application is only possible if the resulting triangulation $\mathcal{T}_\Omega$ preserves the integrity of Γ (or, at least, contains a topological and geometric equivalent of it). Therefore, the need to retrieve the desired missing items lead to consider two different approaches :

1. *a posteriori* recovery of the topological requirements of the constraint, if the constraint integrity is required ;
2. *a priori* redefinition of the constraint, if a topological and geometrical equivalent is satisfactory, so that it appears in the Delaunay triangulation of the convex hull of the set of vertices.

Related work. The *A posteriori* recovery problems can be considered as settled in two dimensions [Chew-1989], [Cline,Renka-1990]. Nevertheless, the extension to three dimensions is rather tricky, mainly because several properties that are valid in $\mathbb{R}^2$ do no longer hold in $\mathbb{R}^3$. In three dimensions, it is well known that *a posteriori* recovery may require additional internal vertices (the so-called *Steiner points*) and their determination can be computationally very complex [Ruppert,Seidel-1992]. However, algorithms with reasonable complexity have been proposed [George-1997], [Weatherill,Hassan-1994], still leaving some jammed configurations unresolved. Moreover, the resulting triangulation is no longer Delaunay, since it has been obtained through local modifications of an initial Delaunay triangulation [2].

[1] as soon as Ω is not convex.
[2] This point becomes especially relevant for some Delaunay-based skeleton approximation methods.

The *A priori* redefinition of the the constraint consists of finding a new constraint, topologically and geometrically equivalent to the intial one, such that it appears into any Delaunay triangulation of the related set of vertices. This method presents two major advantages : on the one hand, it avoids an initial triangulation; on the other hand, it ensures that a Delaunay triangulation, preserving the integrity of the new constraint, can be generated. In two dimensions, the problem can be solved, through edge subdivisions (a solution, valid for simple cases, can be found in [George,Borouchaki-1997]). The three-dimensional case, however, remains widely open mainly because a geometric result in $\mathbb{R}^2$ cannot be extended to $\mathbb{R}^3$. Thus, a simple extension of the two-dimensional method is not possible.

Finally, an *a posteriori* recovery method, based on boundary redefinition, preserving the Delaunay-conformity as been proposed [Sapidis,Perrucio-1991]. It can be qualified as hybrid insofar as the recovery stage follows the triangulation stage, although the constraint integrity is not preserved. Sor far, to our knowledge, this approach has not been extended to three dimensions.

Considering that, for some problems, *a posteriori* methods are either unsuited (because of non Delaunay-conformity for example) or sub-optimal or even divergent (*e.g.*, jammed configurations in $\mathbb{R}^3$), we decided to investigate an *a priori* method, suitable in two and three dimensions. The approach described in this paper is based on local constraint modifications. At first, the two-dimensional case is analyzed for which a solution is proposed. In three dimensions, having discussed the main obstacles to determine a convergent algorithm, we propose a heuristic algorithm, based on face subdivisions and on edge swapping, governed by a geometric estimate.

Outline. Some preliminary definitions related to Delaunay triangulation, their properties and constrained Delaunay triangulations are briefly recalled in Section 1. Section 2, the classification of the edges of a two-dimensional topological constraint Γ is proposed, leading to the strong Delaunay-conformity theorem. This important result allows to decide whether or not a face belongs to the Delaunay covering-up of the given point set $\mathcal{S}$. An algorithm based on edge subdivision is proposed and its convergence established. A set of examples illustrates the method. The case of three-dimensional constraints is discussed in Section 3. An algorithm is proposed to redefine the constraint. Finally, Section 4, further developments are mentioned.

1 Preliminary definitions

Let consider a finite set of points $\mathcal{S}$ in $\mathbb{R}^d$, $d = 2$ or $d = 3$. We recall that a *conforming mesh* of a subset Ω of $\mathbb{R}^d$ is a d-polytopal covering-up of Ω, so that any face of any d-polytope of $\mathcal{T}$ is either a face of another element or belongs to the boundary. In particular, a *triangulation* $\mathcal{T}$ is a conforming mesh, whose elements are d-simplices.

In this section, we shortly recall the notion of Voronoï diagram, dual of the Delaunay triangulation and we introduce the definitions related to constrained triangulations.

Voronoï diagram. The *Voronoï cell* of any point $P_i \in \mathcal{S}$ is defined as :

$$V_i^{\mathcal{S}} = \{M \in \mathbb{R}^3 : (\forall\, P_j \in \mathcal{S})\, MP_i \leq MP_j\},$$

and let call *Voronoï points, edges* and possibly *faces* of $\mathcal{S}$ the intersections of cells of dimensions 1, 2 and 3, respectively. By definition, any cell $V_i^{\mathcal{S}}$ is not empty and is associated to a point $P_i \in \mathcal{S}$. Naturally, the *Voronoï diagram* of $\mathcal{S}$, refered as $\mathcal{V}(\mathcal{S})$, is the set of Voronoï points, edges and possibly faces of $\mathcal{S}$. For sake of concision, a k-Voronoï-face denotes any k-dimensional entity of the Voronoï diagram. Notice that if $\mathcal{S}$ does not contain any $(d+2)$-uple of cospherical points in d dimensions, then the points are said to be in *regular position.*

Delaunay Triangulation. The Delaunay triangulation can be defined as the geometrical dual, in a sense, of the Voronoï diagram. More precisely, if f is a k-face of the Voronoï diagram of $\mathcal{S}$, such that the only Voronoï cells containing f are the ones associated to the points of the set $\mathcal{F} \subset \mathcal{S}$, then $\mathcal{F}$ is said to be a $(d-k)$-Delaunay-face of the convex hull of $\mathcal{F}$. If a face is Delaunay-conforming in $\mathbb{R}^3$, so are its three edges. However, the converse is not true. The set $\mathcal{D}_{\mathcal{S}}$ formed with $\mathcal{S}$ and all the k-Delaunay-faces defined by $\mathcal{V}(\mathcal{S})$ is called the *Delaunay covering-up* of $\mathcal{S}$. The following result is usually known as the *empty ball property* [Delaunay-1934] :

Theorem 1.1 *Under the previous assumptions, $\mathcal{D}_{\mathcal{S}}$ is a conforming mesh of the convex hull Conv($\mathcal{S}$) of $\mathcal{S}$. A d-simplex K is an element of $\mathcal{D}_{\mathcal{S}}$ if and only if the closed circumscribed ball $\mathcal{B}_K$ of K does not contain any point of $\mathcal{S}$ but the vertices of K.*

Notice that the Delaunay-covering-up $\mathcal{D}_\mathcal{S}$ is a triangulation, the so-called *Delaunay triangulation* of $\mathcal{S}$, if and only if the points of $\mathcal{S}$ are in regular position. The Delaunay triangulation is not unique, since non-simplicial elements of the unique covering-up $\mathcal{D}_\mathcal{S}$ allows several simplicial decompositions.

Theorem 1.2 (Delaunay's general lemma) *Let $\mathcal{T}$ be a triangulation of the convex hull of a finite point set $\mathcal{S}$. If the empty ball criterion is satisfied for any configuration of two adjacent elements of $\mathcal{T}$, then it is true for the whole $\mathcal{T}$.*

Constrained triangulation. Let consider a finite point set $\mathcal{S}$ and a topological constraint Γ (set of edges and possibly triangular faces). The vertices of Γ form a subset $\mathcal{S}(\Gamma)$ of $\mathcal{S}$. More precisely, a finite set of edges and possibly triangular faces Γ of $\mathbb{R}^d$ is called a *field of constraints* if its vertices belong to $\mathcal{S}$, and its edges (resp. faces) have no intersection but their endpoints (resp. boundaries). In addition, any point of $\mathcal{S}$ shall not belong to the interior of any entity of Γ. A triangulation $\mathcal{T}$ *exactly* complies with a field of constraints Γ if any component of Γ fully exists in $\mathcal{T}$. However, the unique Delaunay-covering-up $\mathcal{D}_\mathcal{S}$ of $\mathcal{S}$ does not generally allow to exactly comply with Γ, since $\mathcal{D}_\mathcal{S}$ is a covering-up of the convex hull. Therefore, it is useful to propose a weakened definition for a constrained triangulation, of a more general usage. A triangulation $\mathcal{T}$ *weakly* complies with a field of constraints Γ if any component of Γ exists exactly or by covering-up in $\mathcal{T}$. In other words, weak compliance of Γ is equivalent to exact compliance of a field of constraints Γ', the edges and possibly faces of which are obtained by local modifications of entities of Γ.

2 Two-dimensional field of constraints

Let consider a field of constraints Γ, associated with a finite set of distinct points $\mathcal{S}$. The aim of this section is to establish a condition necessary and sufficient for the existence of an edge in any Delaunay triangulation of a $\mathcal{S}$ containing its endpoints. The proposed method is based on the *a priori* redefinition of the constraint so that any Delaunay-based generator retrieves the new field of constraints, thus ensuring the weak compliance with the initial one. To this end, we first propose of the edges of Γ, in order to identify which edges will be present in any Delaunay triangulation of $\mathrm{Conv}(\mathcal{S})$.

2.1 Classification

In this paragraph, we introduce the Θ_2 function and we discuss hereafter the treatment applied to non-conforming edges. For sake of clarity, we call *small circle* (resp. *small disk*) of a non degenerated edge the circle (resp. disk) having this edge as diameter. It is clear that the small disk of an edge is the smallest disk containing this edge. For any couple (P_i, P_j) of distinct points in $\mathcal{S}$, we denote $\mathcal{C}_{ij}$ (resp. Δ_{ij}) the small circle (resp. small disk) of the edge $[P_iP_j]$. The two open half-planes limited by the line $(P_iP_j) = \mathcal{H}_{ij}$ are denoted as $\mathcal{H}_{ij}^+$ and $\mathcal{H}_{ij}^-$. Provided that $\mathcal{S}_{ij} = \Delta_{ij} \cap \mathcal{S}$, we propose the following definition :

Definition 2.1 *Let P_i and P_j distinct in $\mathcal{S}$ be the two endpoints of an edge f in Γ. The* type *of f, denoted as $\Theta_2(f)$, is defined as :*

- *If $\mathcal{S}_{ij} = \{P_i, P_j\}$, then $\Theta_2(f) = 0$* (Figure 1).
- *If $\mathcal{S}_{ij} \setminus \{P_i, P_j\} = \mathcal{E}$, with $\mathcal{E} = \{Q_1, ..., Q_r\}$ such that $\mathcal{E} \subset \mathcal{H}_{ij}^+$ or $\mathcal{E} \subset \mathcal{H}_{ij}^-$, let Q^* be a point of $\mathcal{E}$ such as the radius of the circumscribed disk Δ_t of the triangle $t = P_iP_jQ^*$ is maximal;*
  - *if $Q^* \in \mathcal{C}_{ij}$ or if $\Delta_t \cap \mathcal{S} = \{P_i, P_j, Q^*\}$, then $\Theta_2(f) = 1.0$* (Figure 2);
  - *if $\mathring{\Delta}_t \cap \mathcal{S} \neq \varnothing$, then $\Theta_2(f) = 1.1$;*
  - *if $Q^* \notin \mathcal{C}_{ij}$, $\mathring{\Delta}_t \cap \mathcal{S} = \varnothing$ and $(\Delta_t \cap \mathcal{S}) \setminus \{P_i, P_j, Q^*\} = \mathcal{F} \neq \varnothing$, then :*
    - *if $\mathcal{F} \cup \{Q^*\} \subset \mathcal{H}_{ij}^+$ or if $\mathcal{F} \cup \{Q^*\} \subset \mathcal{H}_{ij}^-$, then $\Theta_2(f) = 1.2.0$* (Figure 3);
    - *if $(\mathcal{F} \cup \{Q^*\}) \cap \mathcal{H}_{ij}^+ \neq \varnothing$ and $(\mathcal{F} \cup \{Q^*\}) \cap \mathcal{H}_{ij}^- \neq \varnothing$, then $\Theta_2(f) = 1.2.1$.*
- *If $\mathcal{S}_{ij} \cap \mathcal{H}_{ij} = \{P_i, P_j\}$, $\mathcal{S}_{ij} \cap \mathcal{H}_{ij}^+ \neq \varnothing$ and $\mathcal{S}_{ij} \cap \mathcal{H}_{ij}^- \neq \varnothing$, then $\Theta_2(f) = 2$.*

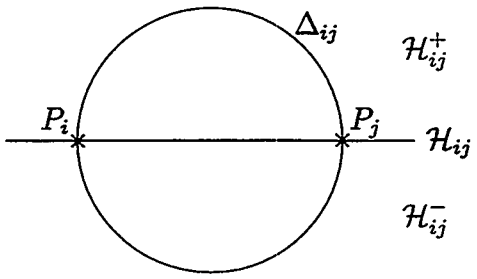


Figure 1: $\Theta_2(f) = 0$

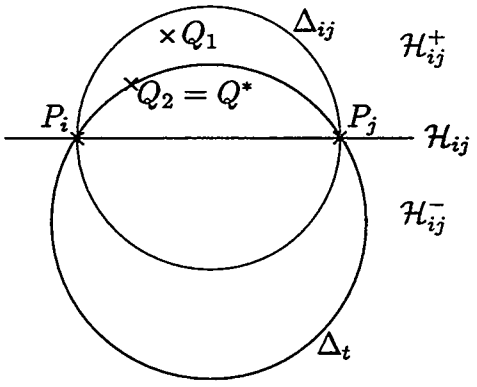


Figure 2: $\Theta_2(f) = 1.0$

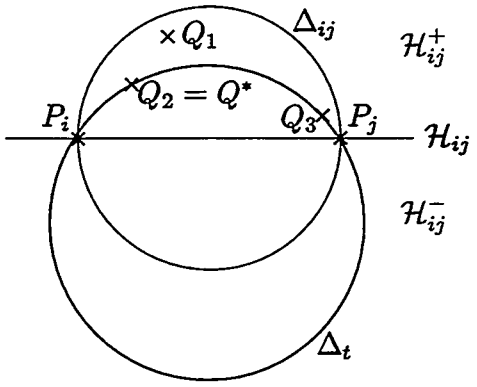


Figure 3: $\Theta_2(f) = 1.2.0$

Discussion. For a given edge f, three different cases can be encountered :

1. $f \in \mathcal{D}_S$: then f belongs to any Delaunay triangulation of Conv($\mathcal{S}$);
2. $f \notin \mathcal{D}_S$: there is no conclusion at this point, since two possibilities have to be discussed :
   (a) if f is not an edge nor a diagonal of any polygon of $\mathcal{D}_S$, then f is non Delaunay-conforming, irrespective to the Delaunay triangulation deduced from $\mathcal{D}_S$;
   (b) if f is a diagonal of a non-triangular polygon Π of $\mathcal{D}_S$, then there exists at least one triangulations of Π containing f and another one not containig f. Thus, f is Delaunay-conforming with respect to at least one among the triangulations deduced from $\mathcal{D}_S$, but not for all of them.

As a consequence of this remark, it appears that *a priori* Delaunay-conformity makes no sense, since this conformity depends on the geometric properties of Γ, as well as on the choice of a particular triangulation. For this reason, we introduce a stronger assumption about Delaunay-conformity, for which a necessary and sufficient condition can be exhibited.

Definition 2.2 *If an edge belongs to the Delaunay-covering-up of Conv($\mathcal{S}$), then it is said to be* strongly Delaunay-conforming.

This definition is equivalent to say that a strongly Delaunay-conforming edge appears in any Delaunay triangulation of Conv($\mathcal{S}$). Following this definition, we give a theorem (the proof can be found in [Pébay-1998b]) based upon the analysis of the Voronoï diagram in the different cases. It ensures that a given edge belongs to any Delaunay triangulation or, on the contrary, is lacking in at least one such triangulation.

Theorem 2.1 *An edge f is strongly Delaunay-conforming if and only if* $\Theta_2(f) \in \{0\,;\, 1.0\,;\, 1.2.0\}$.

2.2 Redefinition of the constraint

Based on the classification, we have seen that some edges of the constraint Γ may be non Delaunay-conforming. Therefore, the proposed method consists in subdividing the entities of Γ so as to obtain a new constraint Γ' that

is strongly Delaunay-conforming and related to a new set of points $\mathcal{S}'$ containing $\mathcal{S}$. Thus, Γ' will be built by any Delaunay triangulation of the convex hull of Γ', *i.e.* Γ will be weakly satisfied. The following algorithm is based on projections, according to the following lemma :

Lemma 2.1 *Let P_i, P_n and P_j be three points such as $P_i \neq P_j$ and $P_n \in]P_iP_j[$. The small circles of $[P_iP_n]$ and $[P_nP_j]$ are inwardly tangent to the small circle of $[P_iP_j]$.*

Corollary 2.1 *Let $[P_iP_j]$ be a non degenerated edge of Γ, with a small disk Δ_{ij} containing at least one point $P_n \in \mathcal{S} \setminus \{P_i, P_j\}$. Denoting as M, Δ and Δ' respectively the orthogonal projection of P_n on $[P_iP_j]$ and the small disks of $[P_iM]$ and $[MP_j]$, then :*

$$Card(\Delta \cap \mathcal{S}) + Card(\Delta' \cap \mathcal{S}) < Card(\Delta_{ij} \cap \mathcal{S})$$

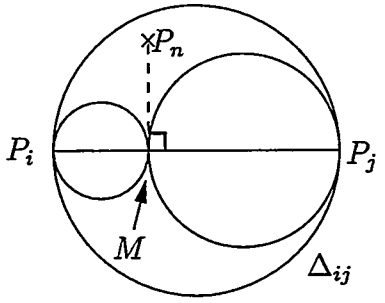


Figure 4: Insertion of a point onto an edge; related small disks.

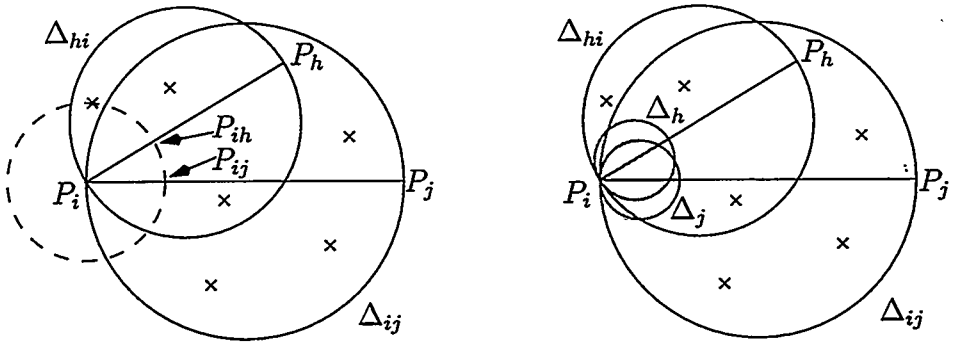


Figure 5: Insertion of two vertices onto the edges of an acute angle; related small disks.

General scheme. Based on this result, we define, for any edge $f = [P_iP_j]$, the small disk of which contains a point $P_n \in \mathcal{S}$, the **projection**(P_i, P_j, P_n) function (Figure 4), that divides f in $[P_iM]$ and $[MP_j]$, where M is the orthogonal projection of P_n onto f. However, the discussion prior to the proof of the convergence makes clear that when two adjacent edges form an acute angle, then splitting the one may affect the other [Pébay-1998b]. More precisely, two such edges having intersecting small disks, one can imagine a progression of alternate subdivisions, thus leading to a divergence problem. For any vertex P_i at which at least two edges form acute angle, we denote as d_i the smallest distance between P_i and we define the **split**(P_i, P_j, d) function (Figure 5), for any $d < P_iP_j$, which splits any $[P_iPj]$ edge in two new edges $[P_iM]$ and $[MP_j]$, such as $P_iM = d$. Applying this function to any acute angle makes the algorithm convergent, but may result in unecessary vertex creations, since this sufficient condition is much too strong. Therefore, we introduced the notion of *acute node* (a vertex at which a cyclic progression of acute angles meet) and showed that using **split** is required only if the constraint contains at least one such acute node. An important consequence is that any manifold constraint can be treated with **projection** alone.

Algorithm 2.1 (Edges subdivision)

1. *Initializations :*
   (a) *Test strong Delaunay-conformity of the edges of* Γ *and form the heap* $\mathcal{T}(\Gamma)$ *of the non-conforming, ordered by decreasing number of points of* $\mathcal{S}$ *located inside their small disks.*
   (b) *Detect acute nodes in* Γ.
2. *While* $(\mathcal{T}(\Gamma) \neq \varnothing)$ *do :*
   - *if* $\mathcal{T}(\Gamma)$ *contains an edge* $f = [P_i P_j]$ *such as at least one its edges* P_* *is the center of an acute node, do :*
     (a) *build the acute node* $\mathcal{N}$ *related to* P^*;
     (b) $d_* = \min MP_*$, $M \in \mathcal{S} \cap \bigcup_{\phi \in \mathcal{N}} \Delta_\phi \setminus \{P_*\}$;
     (c) *set* $0 < d < d_*$;
     (d) *for any* $\phi = [P_* M] \in \mathcal{N}$, $\mathbf{split}(P_*, M, d)$.
   - *else, for the edge* $f = [P_i P_j]$ *from the top of* $\mathcal{T}(\Gamma)$, *do :*
     (a) *pick* $P_n \in \mathcal{S}_{ij} \setminus \{P_i, P_j\}$;
     (b) $\mathbf{projection}(P_i, P_j, P_n)$.
   - *setup* $\mathcal{S}$, Γ *and* $\mathcal{T}(\Gamma)$.

2.3 Examples of constraint redefinition

The convergence of the Algorithm 2.1 has yet been established theoretically, only a few pathological cases are supplied here (see [Pébay-1998b] for more examples).

Figure 6.(i) shows an example of a non strongly Delaunay-conforming field of constraints. In this case, edges $[P_2 P_3]$ and $[P_5 P_6]$ in Γ are missing in the Delaunay triangulation of the convex hull of $\mathcal{S}$. This is a typical problem of two-dimensional constraints, with acute angles and interfering vertices.

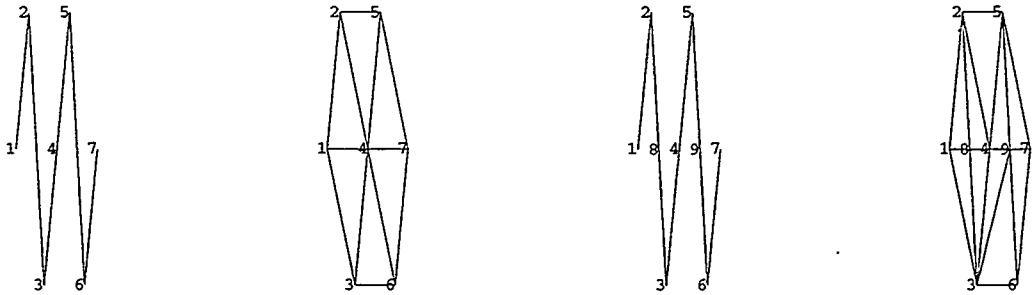


Figure 6: *(i) and (ii) : the original constraint and the Delaunay triangulation of its set of vertices. (iii) and (iv) : the strongly Delaunay-conforming redefined constraint and the Delaunay triangulation of the new set of vertices.*

Figure 6.(iii) presents the *a priori* redefinition of Γ by Algorithm 2.1, which provides a new field of constraints Γ', related to an augmented set of points. The newly created points (P_8 and P_9) ensure weak compliance of Γ, since its missing edges (in the sense of strong compliance) are represented by their subdivisions.

Three other examples are given figures 7, 8 and 9. The first one has academic motivations, since it bears several problematic pathologies : non-manifold constraint with, in addition, many small acute angles. The second one is a boundary-constraint, which is a frequently occuring requirement in practical meshing applications. Finally, the specific problems of acute nodes is emphasized by the third example.

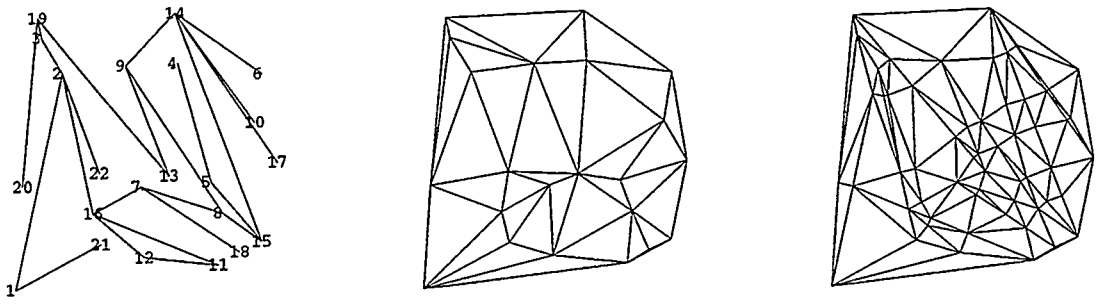


Figure 7: *Example of an* a priori *redefinition. (i) and (ii) : the original constraint is not strongly Delaunay conforming. (iii) The redefined constraint is built by the Delaunay triangulation of the new set of vertices.*

3 Three-dimensional field of constraints

The aim of this section is to introduce a classification and and algorithm suitable to redefine a field of constraints based on the same ideas as in two dimensions. Notice that all the faces are supposed to be triangular and for sake of concision, they will simply be called faces.

3.1 Classification

The previous classification can be extended to deal with a three-dimensional field of constraints Γ related to a set of points $\mathcal{S}$. As mentioned previously, if a face is strongly Delaunay-conforming, so are its three edges, when the converse is not true. Therefore, we aim at classifying the faces of Γ.

Naturally, we introduce the Θ_3 function, giving the type of a face of Γ. For any triple (P_i, P_j, P_k) of distinct points in $\mathcal{S}$, we call *small sphere*, denoted $\partial\mathcal{B}_{ijk}$ (resp. *small ball*, denoted $\mathcal{B}_{ijk}$) of the face $[P_iP_jP_k]$ the sphere (resp. ball), whose big circle [3] is the circumscribed circle of the face. It is clear that the small sphere is the smallest sphere which passes through the three vertices of a face. The two open half-spaces limited by the plane $(P_iP_jP_k) = \mathcal{H}_{ijk}$ are denoted as $\mathcal{H}^+_{ijk}$ and $\mathcal{H}^-_{ijk}$. Assuming that $\mathcal{S}_{ijk} = \mathcal{H}_{ijk} \cap \mathcal{S}$, we define :

Definition 3.1 *Let P_i, P_j and P_k distinct in $\mathcal{S}$ be the three vertices of a face f in Γ. The* type *of f, denoted as $\Theta_3(f)$, is defined as :*

- *If $\mathcal{S}_{ijk} = \{P_i, P_j, P_k\}$, then $\Theta_3(\mathbf{f}) = \mathbf{0}$.*
- *If $\mathcal{S}_{ijk} \setminus \{P_i, P_j, P_k\} = \{Q_1, ..., Q_r\} = \mathcal{E}$ such as $\mathcal{E} \subset \mathcal{H}^+_{ijk}$ or $\mathcal{E} \subset \mathcal{H}^-_{ijk}$, let Q^* be a point of $\mathcal{E}$ such as the radius of the circumscribed ball $\mathcal{B}_t$ of the tetrahedron $t = P_iP_jP_kQ^*$ is maximal;*
  - *if $Q^* \in \partial\mathcal{B}_{ijk}$ or if $\mathcal{B}_t \cap \mathcal{S} = \{P_i, P_j, P_k, Q^*\}$, then $\Theta_3(\mathbf{f}) = \mathbf{1.0}$;*
  - *if $\mathring{B}_t \cap \mathcal{S} \neq \varnothing$, then $\Theta_3(\mathbf{f}) = \mathbf{1.1}$;*
  - *if $Q^* \notin \partial\mathcal{B}_{ijk}$ and $\mathring{B}_t \cap \mathcal{S} = \varnothing$ and $(\mathcal{B}_t \cap \mathcal{S}) \setminus \{P_i, P_j, P_k, Q^*\} = \mathcal{F} \neq \varnothing$, then :*
    - *if $\mathcal{F} \cup \{Q^*\} \subset \mathcal{H}^+_{ijk}$ or if $\mathcal{F} \cup \{Q^*\} \subset \mathcal{H}^-_{ijk}$, then $\Theta_3(\mathbf{f}) = \mathbf{1.2.0}$;*
    - *if $(\mathcal{F} \cup \{Q^*\}) \cap \mathcal{H}^+_{ijk} \neq \varnothing$ et $(\mathcal{F} \cup \{Q^*\}) \cap \mathcal{H}^-_{ijk} \neq \varnothing$, then $\Theta_3(\mathbf{f}) = \mathbf{1.2.1}$.*
- *If $\mathcal{S}_{ijk} \cap \mathcal{H}_{ijk} = \{P_i, P_j, P_k\}$, $\mathcal{S}_{ijk} \cap \mathcal{H}^+_{ijk} \neq \varnothing$ and $\mathcal{S}_{ijk} \cap \mathcal{H}^-_{ijk} \neq \varnothing$, then $\Theta_3(\mathbf{f}) = \mathbf{2}$.*
- *If $(\mathcal{S}_{ijk} \cap \mathcal{H}_{ijk}) \setminus \{P_i, P_j, P_k\} \neq \varnothing$, then $\Theta_3(\mathbf{f}) = \mathbf{3}$.*

[3] That is, any circle with the same radius and center than the sphere.

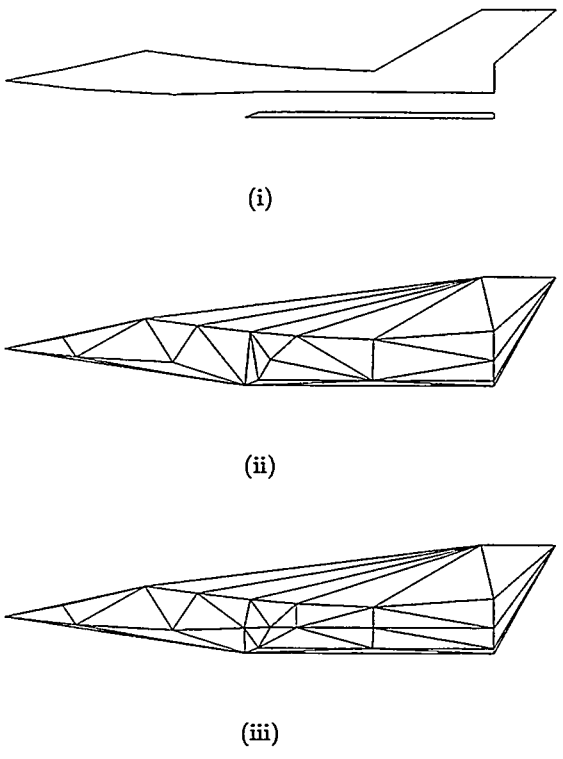


Figure 8: *Example of an* a priori *redefinition of a boundary-constraint. (i) : the constraint, (ii) and (iii) : Delaunay triangulations of the initial and the redefined set of vertices.*

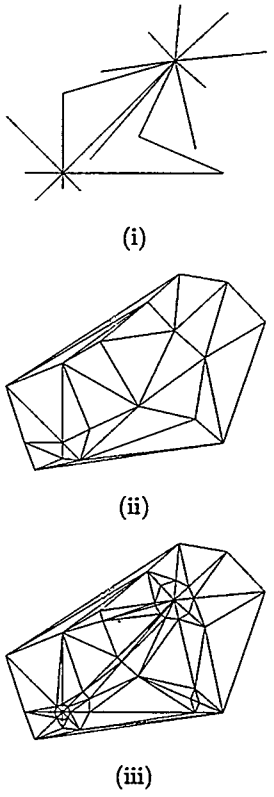


Figure 9: *Example of an* a priori *redefinition of a constraint with two acute nodes. (i) : the constraint, (ii) and (iii) : Delaunay triangulations of the initial and the redefined set of vertices.*

Discussion. As in two dimensions, a given edge f may belong to either all or not all but at least one or even none of the Delaunay triangulations of the convex hull of $\mathcal{S}$. For the same reason, the Definition 2.2 is extended to three dimensions. Provided this definition, the following result holds [Pébay-1998a] :

Theorem 3.1 *A face f is strongly Delaunay-conforming if and only if* $\Theta_3(f) \in \{0\,;\, 1.0\,;\, 1.2.0\}$.

3.2 Redefinition of the constraint

Unfortunately, the Lemma 2.1 does not extend to faces in three dimensions. To the contrary, the small balls of the two faces subdividing any face f are not contained inside the small ball of f [Pébay-1998a]. Therefore, it is not even guaranteed that a convergent algorithm based on simple subdivisions can be found.

In other respects, we like to determine an algorithm using computationally simple operations (thus impacting favorably the speed), as well as restricting the number of newly created faces. For these reasons, a simple algorithm based on face splitting (of the non-conforming ones), using the midpoint of an edge appeared to be a reasonable method (*cf.* Figure 10). Accordingly, three different frames are possible for any face split, having different consequences for its neighbours. Moreover, divergent examples can be exhibited for instance if the frame is either randomly chosen or pre-determined. Hence, the subdivisions have to be governed, this task being devoted to a heuristic geometric estimate. In addition, we allow edge swapping on coplanar adjacent faces, in order to avoid face creation when the problem is limited to two-dimensional strong Delaunay-conformity.

Governing the subdivisions. The Delaunay-conformity being a global property, any subdivision of a face may affect the Delaunay-conformity of other faces. Therefore, it is necessary to design an algorithm that divides one face

at a time, anlyzes the resulting configuration and then proceeds on to the next non-conforming face. Hence, if least two faces are not strongly Delaunay-conforming, the problem of choosing the face to be split arises.

In our approach, we consider the small ball containing the most points of $\mathcal{S}$ and we treat first its associated face, in order to reduce this number. Actually, if it is equal to zero, then any face is 0-type, which is (*cf.* Theorem 3.1) sufficient, but not necessary to strong Delaunay-conformity. Accordingly, the algorithm may stop even before this number vanishes. Nevertheless, focusing on the maximum number of vertices in the small balls allows a global control. From the algorithmic point of view, the procedure uses a heap $\mathcal{T}(\Gamma)$ composed of the faces of Γ, sorted by decreasing number of vertices inside the associated small balls. At each iteration, the face on top of $\mathcal{T}(\Gamma)$ is split, and then $\mathcal{T}(\Gamma)$ is updated.

After a candidate f has been determined, the problem is to choose which one of its edges has to be split (*i.e.* which midpoint will be added to $\mathcal{S}$). Moreover, the simple mesh conformity of Γ, necessary to its strong Delaunay-conformity, requires that any face sharing the split edge must be divided too. Therefore, the effect of the three potential configurations on the adjacent faces have to be taken into account. At this time, we have only studied the case of manifold constraints, which leads to only three *subdivision frames* for a face subdivision. The related function **divide**(f, i), for $i = 1, 2$ or 3, subdivides the face f by inserting the midpoint of edge i, as well as the adjacent face f^i to ensure mesh conformity (*cf.* Figure 10). The newly created faces are denoted as f_1, f_2, f_1^i, and f_2^i.

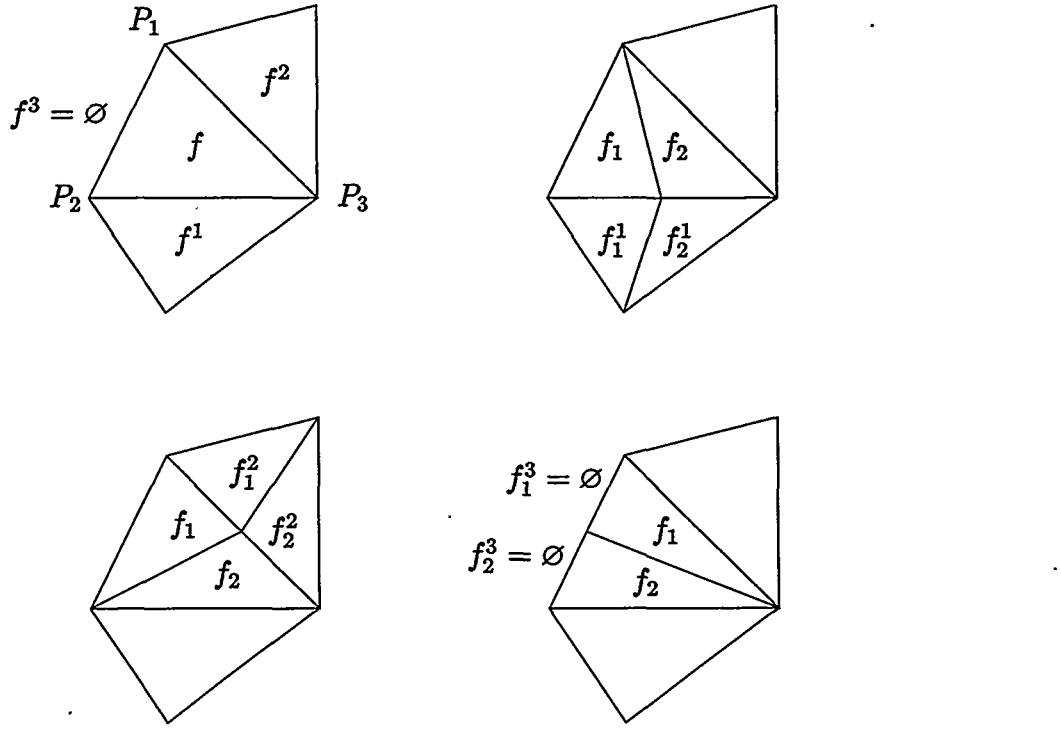


Figure 10: *Example of a face with two neighbours : the three related subdivision frames.*

On the one hand, the choice of the frame is the key point of the convergence because, as we already mentioned it, an unsuitable way of choosing can lead to divergence. On the other hand, the existence of a theoretical result is not proved, although this issue is still under investigation. Therefore, several heuristics have been tested, leading to unsatisfactory results or not results at all except for one approach [Pébay-1998a]. The geometric estimate introduced here uses the fact that any Delaunay triangulation maximizes the minimal angle of the triangles, among all the possible triangulations of the convex hull [Rajan-1994]. Accordingly, it appeared relevant to choose the frame, the minimal angle of which is maximal. Thus, this approach is called *max-min estimate.*

Remark 3.1 *This heuristic does not claim to convergence in general. More specifically, the case of local two-dimensional sub-problems may lead to recursive subdivisions, until creating degenerated faces, with respect to numerical precision. Therefore, we propose a specific treatment for this kind of non-conformity.*

Edge swapping. A local configuration formed by two coplanar faces is the analogous of the case of two triangles in two dimensions. Now, any convex triangulation in $\mathbf{R}^2$ can be transformed in a Delaunay triangulation by means

of edge swapping [George,Borouchaki-1997]. Therefore, if two coplanar faces f and f' share an edge e, and if the Delaunay-covering-up of $\mathrm{Conv}(f \cup f')$ is composed of two adjacent faces f'' and f''', none of them sharing the edge e, then f and f' can be replaced by f'' and f''' (*cf.* Figure 11). When, for any given f in Γ, there exists another face f' for which this operation is possible, we refer it as $f <> f'$, and we define the **swap**(f, f') function, which replaces f and f' by f'' and f''' in Γ.

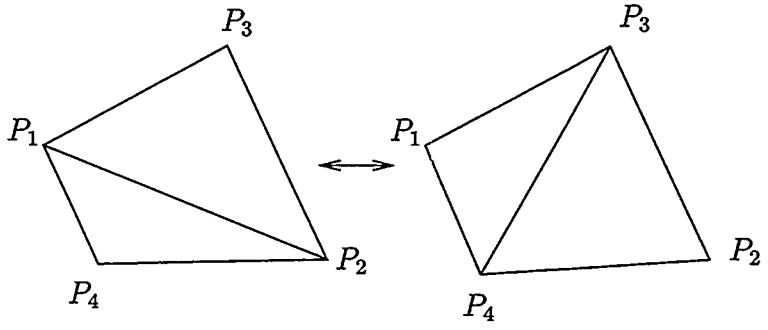


Figure 11: *Example of coplanar faces : edge swapping.*

It is obvious that this local operator can not ensure the strong-conformity of each face. However, being a necessary condition, it will be applied recursively as many as possible.

Algorithm 3.1 (Faces subdivision)

1. *Initialization : Test strong Delaunay-conformity of the faces of* Γ *and form the heap* $\mathcal{T}(\Gamma)$ *of the non-conforming, ordered by decreasing number of points of* $\mathcal{S}$ *located inside their small balls.*
2. *While* $(\mathcal{T}(\Gamma) \neq \varnothing)$ *do :*
   (a) *For the face* f *located on the top of* $\mathcal{T}(\Gamma)$ *do :*
      i. *find the subdivision frame* i *maximising the minimal angle;*
      ii. **divide**(f, i);
      iii. $\mathcal{L} = \{f_1, f_2, f_1^i, f_2^i\}$
      iv. *while* $((\exists \phi \in \mathcal{L})\ (\exists \psi \in \Gamma)\ \ \phi <> \psi)$ *do :*
         A. *add* ψ *to* $\mathcal{L}$;
         B. **swap**(ϕ, ψ);
         C. *update* $\mathcal{L}$.
   (b) *Update* $\mathcal{T}(\Gamma)$.

3.3 Examples of constraint redefinition

Table 1 summarizes the results provided by Algorithm 3.1 applied to a representative set of examples. The set of points $\mathcal{S}$ is initially composed of the N_p vertices of the N_f faces of Γ, a triangular covering-up of the object boundary.

Quality calculations are obtained using the following formula [George,Borouchaki-1997] :

$$\mathcal{Q}_f = \alpha \frac{h_f}{\rho_f}$$

where ρ_f, h_f et α respectively denote the radius of the circumscribed circle of f, its maximal edge length and a constant chosen such as the quality of an equilateral face is 1. Hence, the quality of a face belongs to the interval $[1; +\infty]$; the best, worst and average qualities are respectively denoted as $\mathcal{Q}_+$, $\mathcal{Q}_-$ et $\overline{\mathcal{Q}}$.

N'_p, N'_f, $\mathcal{Q}'_+$, $\mathcal{Q}'_-$ et $\overline{\mathcal{Q}'}$ are the corresponding quantities for the newly generated set of points and field of constraints. The ratio $r = N'_f/N_f$ represents the so-called the *field of constraints size growth coefficient.*

CPU time requirements on a HP-PA 8000 workstation did not exceed 20 seconds in the worst case. Notice that the resulting quality values are globally improved as compared to the initial ones. Regarding the number of newly

	N_p	N_f	$\mathcal{Q}_+$	$\mathcal{Q}_-$	$\overline{\mathcal{Q}}$	N'_p	N'_f	$\mathcal{Q}'_+$	$\mathcal{Q}'_-$	$\overline{\mathcal{Q}'}$	r
A	30	56	1.12	2.11	1.61	44	84	1.04	2.96	1.55	1.50
B	220	436	1.03	6.63	2.07	516	1,028	1.03	6.63	1.73	2.36
C	658	1,312	1.01	7.37	2.19	2,033	4,062	1.00	13.1	1.75	3.10
D	2,506	5,008	1.01	14.3	1.58	2,779	5,554	1.01	14.3	1.63	1.11
E	3,503	7,018	1.00	10.0	1.35	3,763	7,538	1.00	10.0	1.36	1.07
F	2,649	5,290	1.00	12.1	1.89	5,630	11,252	1.00	20.0	1.67	2.13
G	3,605	7,302	1.00	11.4	1.72	5,786	11,664	1.00	9.41	1.69	1.60
H	5,157	10,354	1.00	6.58	1.26	5,289	10,618	1.00	6.46	1.26	1.03
I	10,714	21,476	1.00	5.14	1.17	10,884	21,816	1.00	5.14	1.17	1.02
J	13,183	26,406	1.00	6.63	1.21	13,291	26,622	1.00	6.63	1.21	1.01

Table 1: *Statistics related to boundary constraint subdivision.*

created faces, no relevant conclusion can really be supplied. Actually, it depends on the initial number N_c of strongly Delaunay-conforming faces in Γ.

Initial and final fields of constraints related to the objects C and I are presented in Figures 12 and 13.

3.4 Application to volumetric meshing

In the context of a constrained Delaunay mesh generation software [4], the Algorithm 3.1 allows to redefine the boundary constraint *a priori*, such that the Delaunay triangulation weakly satisfies the initial constraint. No *a posteriori* enforcement operation is necessary and the resulting mesh is Delaunay-conforming.

In Table 2, the number of faces of the initial object is denoted as N_f, among which N_m are missing in the Delaunay triangulation, before being retrieved by local modifications, so that the resulting mesh is no longer Delaunay-conforming.

The N_e tetrahedrons generated by the mesher have for best and worst qualities respectively $\mathcal{Q}_+$ et $\mathcal{Q}_-$, the quality of an element K being given by the formula :

$$\mathcal{Q}_K = \beta \frac{h_K}{\rho_K}$$

where ρ_K, h_K et β respectively denote the radius of the circumscribed ball of K, its maximal edge length and a constant chosen such as the quality of a regular tetrahedron is 1.

After the *a priori* redefinition of the boundary constraint by Algorithm 3.1, the same objects are meshed and the corresponding results are shown in Table 2.

	N_f	N_m	N_e	$\mathcal{Q}_+$	$\mathcal{Q}_-$	N'_f	N'_m	N'_e	$\mathcal{Q}'_+$	$\mathcal{Q}'_-$
A	56	16	62	1.31	4.31	84	0	92	1.24	3.10
B	436	130	628	1.10	9.15	1,028	0	2,354	1.07	9.15
C	1,312	436	2,291	1.15	21.1	4,062	0	19,622	1.03	18.0
D	5,008	74	18,217	1.05	24.9	5,554	0	20,785	1.03	19.3
E	7,018	98	28,135	1.02	11.4	7,538	0	30,347	1.03	15.8
F	5,290	1083	8,469	1.06	17.3	11,252	0	26,386	1.04	16.4
G	7,302	720	9,620	1.03	18.3	11,664	0	23,064	1.05	12.6
H	10,354	52	36,754	1.04	12.2	10,618	0	38,342	1.03	10.7
I	21,476	97	111,556	1.02	5.83	21,816	0	113,492	1.02	5.14
J	26,406	30	131,803	1.03	5.29	26,622	0	133,190	1.03	5.75

Table 2: *Statistics related to meshed objects, before and after redefinition.*

As expected, no face is missing after meshing of the *a priori* redefined constraints, when a variable number of faces have to be retrieved in the initial boundaries. Similarly as for the boundaries, the quality values are globally improved (the histogramm is skewed toward the left).

[4] GHS3D, GAMMA Project, INRIA Rocquencourt.

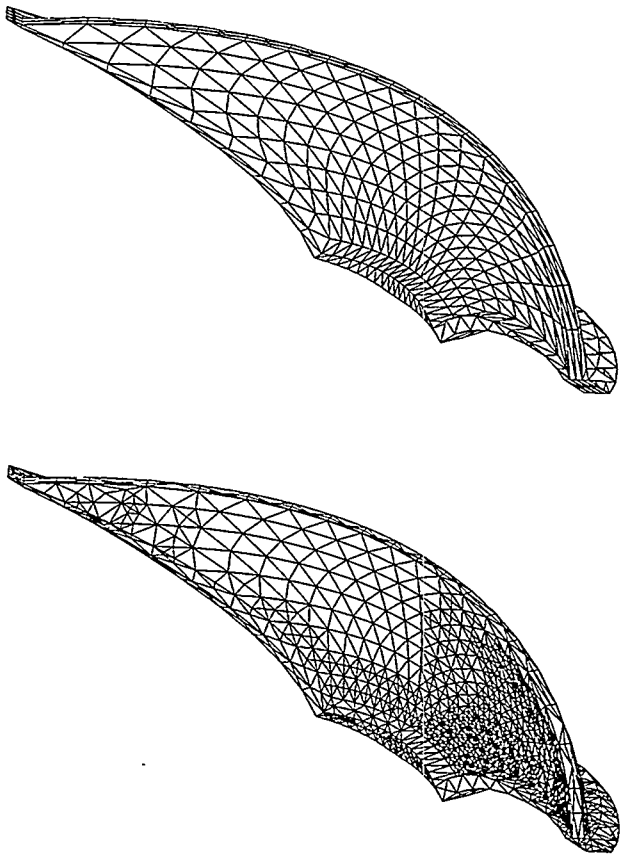


Figure 12: *Example of boundary redefinition : a dam (object C). Left : original feature, right : strongly Delaunay-conforming boundary triangulation.*

4 Conclusion and future work

In this communication, we have shortly recalled some essential properties of the Delaunay triangulations in two and three dimensions. For a given field of constraints (*i.e.* a list of edges and possibly triangular faces) Γ related to a finite set of points $\mathcal{S}$ in $\mathbf{R}^2$ or $\mathbb{R}^3$, we have introduced a geometric classification function allowing to determine *a priori* (*i.e.* prior to the construction of a triangulation) whether or not a face will appear in any Delaunay triangulation of the convex hull of $\mathcal{S}$. We have proposed a convergent algorithm for *a priori* redefinition of a two-dimensional constraint, based on a specific property of circles. Because of specific obstacles in three dimensions, we have proposed a heuristic algorithm, convergent for most of the manifold constraints we have tested.

Among the future developments expected, we shall mention, in no specific order :

- the extension to non-manifold three-dimensional case;
- the proof of a convergent three-dimensional algorithm;
- the medial-axis and mid-surface approximation;
- the potential application to quadrilateral and hexahedral mesh generation.

References

[Chew-1989] L.P. CHEW, Constrained Delaunay triangulations, *Algorithmica* 4, 97-108, 1989.

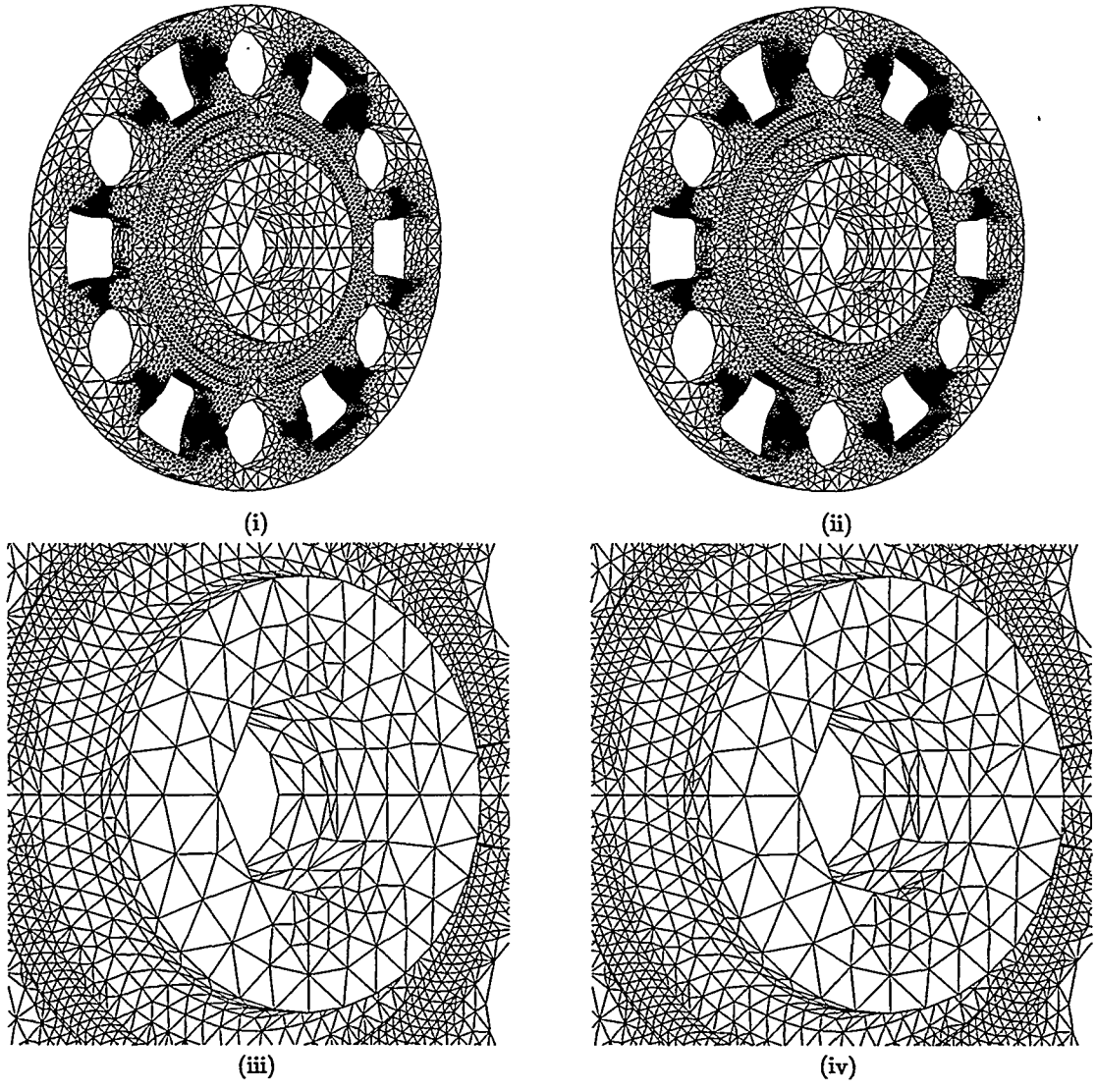


Figure 13: *Example of boundary redefinition : a carrier wheel(object I). (i) : original feature, (ii) : strongly Delaunay-conforming boundary triangulation, (iii) and (iv) : enlargement of (i) and (ii) (data courtesy of ANSYS).*

[Cline,Renka-1990] A.K. CLINE AND R.J. RENKA, Constrained two-dimensional triangulation and the solution of closest node problems in the presence of barriers, *SIAM J. of Num. Anal.* **27**, 1305-1321, 1990.

[Delaunay-1934] B. DELAUNAY, Sur la sphère vide, *Bul. Acad. Sci. URSS, Class. Sci. Nat.*, 793-800, 1934.

[George-1997] P.L. GEORGE, Improvement on Delaunay based tridimensional automatic mesh generator, *Finite Elements in Analysis and Design* **25** (3-4), 297-317, 1997.

[George,Borouchaki-1997] P.L. GEORGE ET H. BOROUCHAKI, Triangulation de Delaunay et maillage, Hermes, 1997.

[Pébay-1998a] PH.P. PÉBAY, Construction d'une triangulation surfacique Delaunay-admissible, Rapport de Recherche RR-3369, INRIA, Rocquencourt, mars 1998.

[Pébay-1998b] PH.P. PÉBAY, Construction d'une triangulation contrainte Delaunay-admissible en dimension 2, Rapport de Recherche, INRIA, Rocquencourt, septembre 1998.

[Pébay-1999] PH.P. PÉBAY, Delaunay-admissibilité et applications, PhD thesis, Université Paris 6, Paris, to be defended.

[Ruppert,Seidel-1992] J. RUPPERT AND R. SEIDEL, On the difficulty of triangulating three dimensional nonconvex polyhedra, *Discrete Computational Geometry* **7**, 227-253, 1992.

[Rajan-1994] V.T. RAJAN, Optimality of the Delaunay Triangulation in $\mathbf{R}^d$, *Discrete Comput. Geom.*, **12** , 189-202, 1994.

[Sapidis,Perrucio-1991] N. SAPIDIS AND R. PERUCCHIO, Delaunay triangulations of arbitrarily shaped planar domains, *Computer Aided Geometric Design* **8**, 412-437, 1991.

[Weatherill,Hassan-1994] N.P. WEATHERILL AND O.HASSAN, Efficient three dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints, *Int. j. numer. methods eng.*, **37**, 2005-2039, 1994.

Mesh graph structure for longest-edge refinement algorithms

Angel Plaza, José P. Suárez, Miguel A. Padrón
Department of Mathematics. University of Las Palmas de Gran Canaria. Spain
{angel,jsuarez,,mpadron}@aries.dma.ulpgc.es

Abstract. In this paper we introduce and discuss a graph structure associated with longest-edge algorithms (algorithms based on the bisection of triangles by the longest-edge) which can be used to reformulate longest-edge algorithms to develop new algorithms and to design efficient data structures for the refinement/derefinement of 2D and 3D triangulations.

Keywords. Longest-edge refinement, bisection, data structures, improvement.

1. Introduction

Modern finite element applications make extensive use of adaptive techniques to optimize the number of unknowns with respect to the accuracy of the numerical solutions. For this purpose, the underlying discretization mesh must be locally refined in regions where improved accuracy is needed. Moreover, multigrid or multilevel methods have shown to be of optimal or nearly optimal complexity for the solution of discrete systems arising from a wide range of partial differential equations. Since these methods are based on discretization hierarchies obtained from succesively refined meshes, they are very appropiate to be embedded in the adaptive framework.

The management of adaptively refined grids and the implementation of adaptive solvers require of specialized refinement and/or derefinement techniques. In particular, the local refinement and/or derefinement of the mesh should not involve a complete reconstruction of the data structures but only the local reconstruction of the grid is implied in the process. In adition, the local reconstruction work should remain proportional to the number of modified elements. To reach these goals, data structures which fit the structure of the problem as well as the refinement scheme are needed.

In the adaptive refinement/derefinement context longest-edge algorithms for triangulations have been extensively studied in the last 15 years (Rivara [12-17], Ferragut [4], Plaza *et al.* [11]). In particular Rivara has studied two algorithms based on longest-edge bisection of triangles have been discussed: the pure longest-edge algorithm (which only performs longest-edge bisection of the current triangles); and the 4-Triangles algorithm where for each target triangle, a longest-edge bisection is firstly performed and the newly vertex is used to subdivide the initial triangles in four. Rivara and Levin have generalized the pure longest-edge agorithm to 3 dimensions [12], while Plaza and Carey [10] developed a new refinement algorithm for tetrahedral grids based on the skeleton: the set comprissed by the faces of the tetrahedra. Their algorithm can be extended to a general dimension N. This algorithm has the advantaje of establishing a finite number of son-elements in the refinement process, and can be seen as the generalization to three dimensions of the 4-T two-dimensional refinement algorithm of Rivara. Other refinement algorithms based on bisection, developed in the last years for three dimensions can be found in [8],[9],[10].

The great amount of information to be managed in 3D case and the obvious complexity to deal with these meshes makes necessary to develop efficient and good algorithms and related data structures in terms of low storage and low computational cost. This fact has encouraged us to first, deal with the two-dimesional case to find out good data structures that improve the existing algorithms and secondly, apply these concepts and results to the three-dimensional case and maybe to a higher dimension as Plaza and Carey stated in [10].

The idea in [10] from Plaza and Carey to represent a mesh as an oriented graph have been taken in this paper as a good way to make easy the process of longest side bisection of Rivara. Besides, this idea

considerably reduces the amount of data to be stored, as it will be shown in this paper. On the other hand, a data structure based on trees is introduced. Trees make it possible to represent the nestedness of the mesh in a hierarchical way, which benefits the multigrid methods and the refinement or derefinement of the mesh. Leinen in [6] made use of trees in a similar way for the two dimensional case and considered the posibility of extending to the three-dimensional case. Bey in [3] also relied on this structure and presented an algorithm for grid refinement which was implemented in the AGM^{3D} software.

In this paper, we present some data structures based on graphs and trees that fit the longest side bisection algorithms of Rivara. In addition, we consider some properties that let us prove the goodness of these data structures and the finiteness for that class of algorithms.

2. Longest edge refinement algorithms.

Definitions:

The *longest edge bisection* of a triangle **t** is the partition of the triangle by the midpoint of its longest edge and the opposite vertex. The *neighbor* of **t** is the neighboring triangle $\mathbf{t}^*$ which shares with **t** the longest edge of **t**. We say that a mesh is *conforming* if any adjacent element (triangle in 2D or tetrahedron in 3D) share an entire face (3D) or an edge (2D) or a common vertex.

2.1 Forward Longest Edge Bisection

In order to make a grid conforming, the local refinement of a given triangle involves refinement of the triangle itself and refinement of some of its neighbors. The algorithms bisect a triangle t (thin line in Fig.1) and its neighbors or the boundary of the domain is reached and so on iteratively until the last two triangles share the same longest edge. Although may be obvius that the refinement propagation stops, we prove later the finiteness of the process. The same idea has to be applied in order to conform the set of non-conforming points generated, (dashed line in Fig. 1) in the inverse of the order in which they were created.

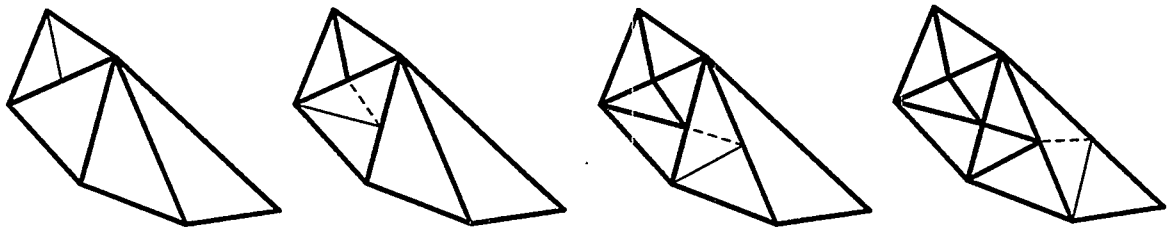


Fig. 1 .Forward Longest Edge Bisection

We shall call this version of the algorithm the Forward Longest Edge Bisection (FLEB).

2.2 Backward Longest Edge Bisection

In [16] Rivara has introduced an improved version of the algorithm called Backward Longest Edge Bisection (BLEB).

Longest-Edge Propagation Path (LEPP) defined by Rivara in [17] is an ordered list of all the triangles $\{t_0,t_1,...,t_n\}$ such that t_i is the neighboor triangle of t_{i-1} by the longest edge of t_{i-1}.

Given a triangle, that last version runs the **LEPP** and it only bisects one or two of the last triangles of the path. The process is repeated as many times as needed until the original triangle is also bisected.

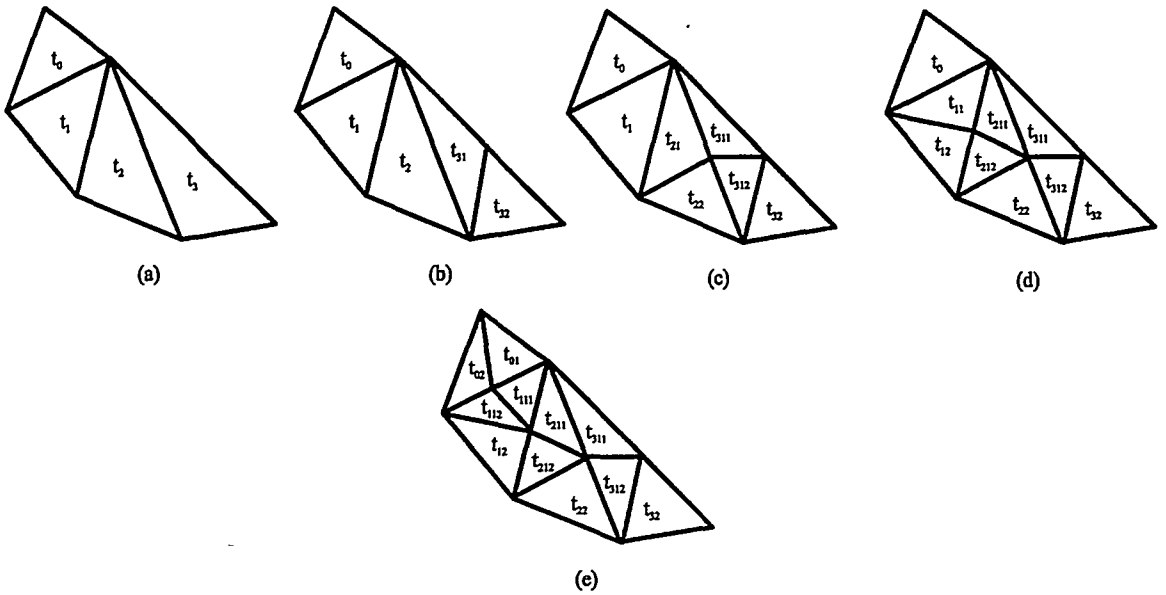


Fig. 2. Backward Longest-Edge Bisection of triangle t_0

In Fig. 2 the Backward Longest-Edge Bisection has been applied to the original triangulation (Fig. 2.a). In the figure are represented the intermediate meshes while bisection is applied. As an explanation see also the **LEPP** of t_0 in each step of the algorithm in the following table.

LEPP(t_0)	Triangles to be bisected	Figure
$\{t_0,t_1,t_2,t_3\}$	$\{t_3\}$	2.(a)
$\{t_0,t_1,t_2,t_{31}\}$	$\{t_2,t_{31}\}$	2.(b)
$\{t_0,t_1,t_{21}\}$	$\{t_1,t_{21}\}$	2.(c)
$\{t_0,t_{11}\}$	$\{t_0,t_{11}\}$	2.(d)

At this point, the following remarks are in order:

Remarks:

1. Both algorithms are equivalent in the sense that they produce the same refined mesh in the end.
2. In both algorithms, the concept of the Longest-Edge Propagation Path (LEPP) introduced above has been repeatedly used. In the first one it is used implicitly and in the second one explicitly.
3. In the first algorithm, FLEB, the assurance of conformity is carried out in each step, however, in the second one, BLEB, nothing must be done because this version guarantees conformity.
4. For both algorithms, a suitable data structure that explicity manages the neighbor-triangle relation should be used.
5. It is expected that both algorithms stop whatever mesh you use. For this purpose, it is sufficient to prove that the LEPP is finite in each step and has no loops.

3. Triangle edges classification and the oriented graph.

The algorithms based on the longest-edge of a triangle discussed above have the advantage that the different possibilities for refining a triangle depend only on determining the longest edge. In general, we can say that these algorithms classify the edges of each triangle in two types, the longest one, **type 1**, and the remaining two edges, **type 2**. Therefore, it is supposed that this classification can be done so that the algorithms perform the right job.

Based on this idea, Plaza and Carey in [10] proposed to represent this classification as an oriented graph in which a vertex of the graph represents an edge of the triangle and an edge of the graph between nodes **a** and **b** represents the relation *"length of edge x in the triangle is less than length of edge y"*. See Fig. 3. In [7], this classification was used to explain the number of different possible configurations obtained by the 3-dimensional refinement algorithm proposed in that paper.

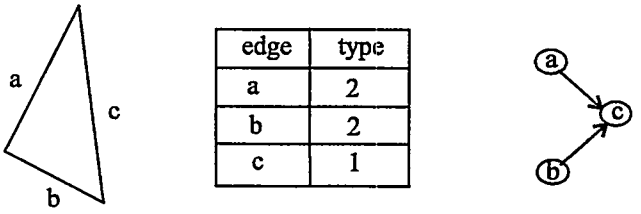


Fig. 3. Triangle, edges types and associated graph

This idea can be beneralized as follows to the whole mesh.

Definition:

Let τ be any mesh having a finite number of triangles. For each triangle t in τ, we sort the edges of t as above and then build the oriented graph G(V,E) associated to the whole mesh, where:

- *$V = \{v_0, v_1, ..., v_n\}$ so that $v_i \in V$ is the vertex on the graph representing an edge of the triangle in the mesh.*
- *$E = \{e_0, e_1, ...e_m\}$ so that $e_j \in E$ is the edge on the graph representing the relation **R** between two vertices belonging to v so that **R** is "length less than". That is, e **R** f if and only if:*
  - *i) e and f belong to the same triangle*
  - *ii) length(e) < length(f)*

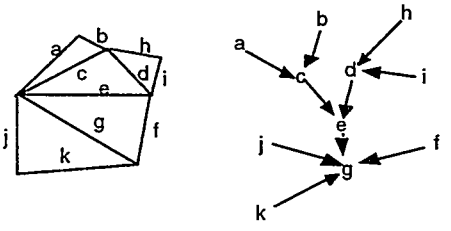
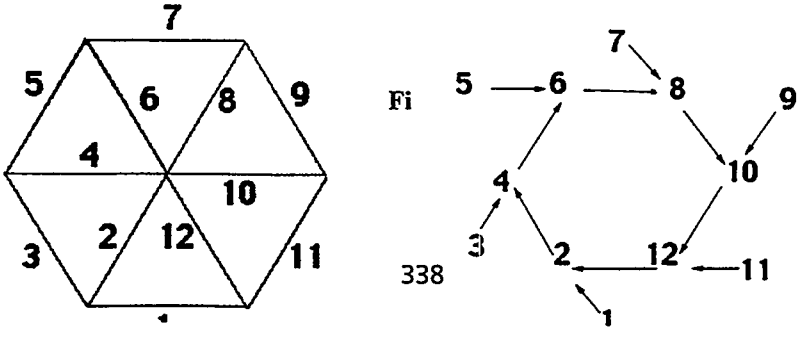


Fig. 4

Note hovewer that, when regular elements appear in the mesh, and since for such elements the longest-edge is not unique, the associated graph can include undesirable loops. A critical situation is illustrated in Fig. 5 where all the triangles are regular and as a consequence the classification of the edges in each triangle can not be carried out.



In order to avoid the ambigousness caused by non-unique longest-edges in the pratical implementation of the longes-edge algorithms the following additional convection is used: whenever the *LEPP(t)* is not unique (due to the existence of elements having non-unique longest-edges) the shortest possible path is selected.

Remark. *Let Ω be any boundary 2-dimensional domain, with poligonal boundary Γ, and let τ be any unstructured and conforming triangulation of Ω having regular triangles. Then, the following conditions have to be taken into account in building the associated graph of τ:*

1. *If a regular triangle $t \in \tau$ is the first to be considered in a LEPP of the graph, we convey any possible relation among the edges.*
2. *Let t_i and t_{i+1} be two neighbor triangles $\in \tau$ to be considered in a LEPP of the graph, if t_{i+1} is regular, we make the edges of t_{i+1} to point to the shared edge in the associated graph.*

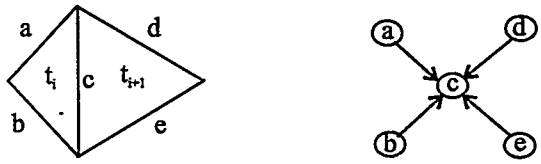


Fig. 6

Proposition 1.- *The previous remark allows us to deal with regular triangles without ambiguity.*
Proposition 2.- *Let G(V,E) be the associated graph of an unstructured and conforming triangulation τ, then G does not hold loops.*
Proof:
There are two situations in which loops can occur. The first one arises in meshes like those referred in Prop. 1, Fig. 5. In this case, the way in which the graph is built assures the non-existence of loops (see Prop. 1). The second situation is as follows:

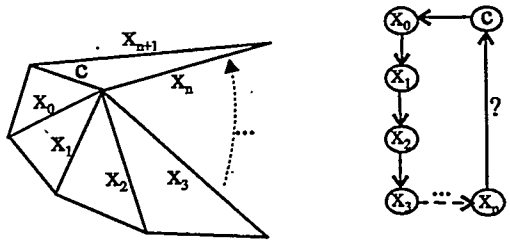


Fig. 7. Triangulation and simplified associated graph in which a possible loop can occur.

In which:

I. $\mathbf{X=\{x_0, x_1, x_2, x_3, x_n\}}$ are the common edges shared by the neighbor triangles $\in$ **LEPP** of $\mathbf{t_0}$ ($\mathbf{t_0}$ is the triangle defined by edges **c** and $\mathbf{x_0}$)

II. $\mathbf{Length(x_{i-1}) < Length(x_i)}$, for i=1 to n

In that situation we must prove that $\mathbf{x_n}$ does not point to **c**, otherwise there is a loop. It can be obvious in Fig.7 that **Length(c)** is less than both $\mathbf{Length(x_n)}$ and $\mathbf{Length(x_{n+1})}$ and so it is impossible that $\mathbf{x_n}$ or $\mathbf{x_{n+1}}$ points to **c**.

Anyway, let us suppose that there is a loop from $\mathbf{x_n}$ to **c** and then we will yield an absurd caused by the wrong supposition.

If so,

III. $\mathbf{Length(c) > Length(x_n)}$

But we know that

IV. $\mathbf{Length(x_0) > Length(c)}$

Then, from III and IV we have:

V. $\mathbf{Length(x_0) > Length(x_n)}$

However, V is not possible because we supposed II. This is due to the supposition in III. Consequently, it must be:

$\mathbf{Length(c) < Length(x_n)}$

and then there are not loops because:

- c and x_{n+1} point to x_n (if x_n is of **type 1**) Fig. 8.a or
- c and x_n point to x_{n+1} (if x_{n+1} is of **type 1**) Fig. 8.b

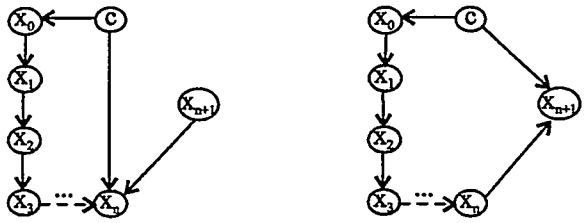


Fig. 8

4. A graph data structure

In what follows a data structure based on the graph representation of the mesh is proposed.

Remark. *In essence, the goal is to design a data structure that provides the best balance between storage and computational cost without increasing significantly the code complexity.*

Although that data structure could make the bisection process easy and have good features related with the implementation, we also need another data structure that let us preserve mesh nestedness, in other words, mesh genealogy. This last requirement benefits refinement or derefinement processes as well as multigrid methods, as we pointed out in the introduction. Moreover, we must to have the last remark in mind. Several solutions have been proposed for that purpose but trees seem suit best, as many authors think: Carey *et al.* in [4], Leinen in [6] or Bey in [3].

4.1 Modified Adjacency List

A standard data structure to model an oriented graph is the *Adjacency List*, especially adequate to the case when few links among the vertices are managed [1], [5]. In such a list, the entire set of vertices in the graph are arranged as a linked list (see column linked list in Fig. 9.c). It can be noted that instead of using a linked list for all edges in the mesh, an array can also be used. This benefits the direct access to a given item, which is not achieved by means of a linked list. However, an additional and fixed cost of storage is incurred on choosing the array structure. Furthermore, the advantage of a dynamic structure based on links and pointers is lost. In this paper we choose the first structure (linked list) but it is an open decision, which depends on the particular implementation.

For each node in the list, a new linked list is associated (see row linked lists in Fig. 9.c). It represents the vertices adjacent to the first one. That last adjaceny list is based on triangle edges, and so, we call it *Edge Adjacency List.*

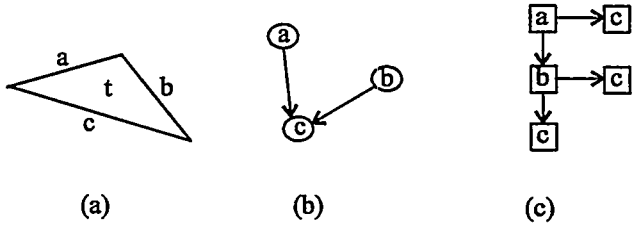


Fig. 9. (a) An arbitrary triangle. (b) Graph associated. (c) Adjacency List

Up to now our data structure as well as the graph are based on edges. However, it is desirable to deal with triangles as implicitly Rivara's algorithms do. Therefore, we introduce a new list but based on triangles. The column linked list holding the vertices in the graph is preserved. For each vertex in the graph having two

incident edges, we link a new node, in a row linked list, representing the triangle defined by them (see Fig. 10). We call this new list *Triangle Adjacency List.*

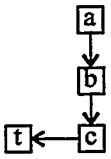


Fig 10. The new list based on triangles. See Fig. 9

It really represents a triangle related with its longest edge, except for the case described in Remarx 2. By means of that new list for each vertex, we are able to access not only the edges in the triangles but the triangles in the mesh in a quick and easy way.

Combining now the two lists in only one yield the *Modified Adjacency List.*

Definition:

Let τ be an unstructured and conforming triangulation. Let $G(V,E)$ be the associated graph representing a classification based on the triangle longest edge except for the case of regular elements. We introduce the Modified Adjacency List, a data structure that make it possible to access both edges and triangles as Rivara's algorithms need.

Graphically:

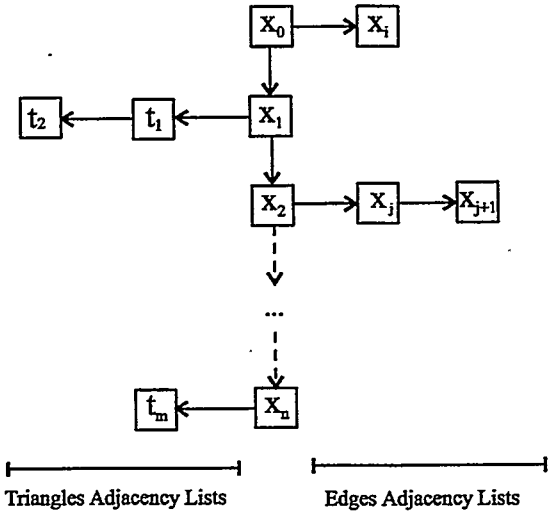


Fig. 11. Modified Adjacency List

Remark:

1. The maximum number of nodes in each of the *Edge Adjacency Lists* is 2.
2. The maximum number of nodes in each of the *Triangle Adjacency Lists is* 2.
3. Given an arbitrary edge, the maximum number of nodes in its *Edge Adjacency List* plus its *Triangle Adjacency List* is 2.

Proposition 3.-*Let n be the number of edges in a triangulation τ, then the storage cost needed to store the Modified Adjacency List is at most 3n.*

Proposition 4.- *The minimum maintenance cost of Modified Adjacency List holds BLSB and affects only to five edges (if two triangles are bisected) or three (if one triangle is bisected) in each refinement step.*

Remark 5.-*The previous proposition ensures the locality of the Modified Adjacency List data structure.*

Proposition 5.- *Let τ be an unstructured and conforming triangulation τ, let e_i be one of the three edges of an arbitrary triangle t_j of τ and let N be the number of triangles in τ, then Rivara's **LEPP**(t_j) can be obtained from the depth run of the Modified Adjacency List from e_i, with a maximum cost of (N+1).*

Proposition 6.- *Rivara's algorithms stop in a finite number of steps.*
Proof:
As mentioned in Remark 1.2, Rivara's algorithms deal with LEPP in order to make triangles bisections. Provided that **LEPP** was always finite, we will ensure the finiteness of algorithms. In fact, **LEPP** is finite because **G(V,E)** does not hold loops (see Proposition 2). An upper bound of items in LEPP is **N**, where **N** is the number of triangles in the mesh.

4.2 Trees to store mesh genealogy.

As pointed out above in section 4, trees appear to be a suitable data structure for representing nestedness in the mesh. The typical order relation on trees is not assumed, but another one that stands for nestedness. Therefore, if a triangle $\mathbf{t_i}$ is subdivided in two, denoted by $\mathbf{t_{11}}$ and $\mathbf{t_{12}}$, we can describe that relation as:

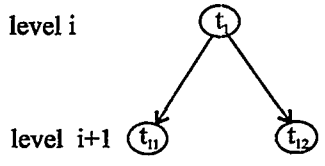


Fig. 12. A tree representing a triangle subdivision.

We now study how trees can be adapted for using them in Rivara's algorithms and generally in algorithms based on bisection. One desirable restriction to be satisfied by trees is that nodes of a given level have to represent conforming triangles and only just the subtriangles created in a given level of the refinement.

Look at Fig. 1 related to FLEB and notice that in each level of refinement, one given triangle can be subdivided in three adittional triangles. However, if we consider BLEB in Fig. 2, two only subtriangles appear in each step of the refinement. This let us to think of one data structure based on a ternary tree for FLEB and a binary tree for BLEB. Binary trees are well known and have been studied by several authors [1].

We concentrate on Rivara BLEB's because it presents the simplest case. The division process of BLEB can be modelled as a binary tree like in Fig. 12.

Definitions:
A *node* on the tree represents one triangle in the mesh and a top down *arrow* represents a relationship father-son. A node is said a *father* from other one named *son* if the last is obtainded by the bisection of the father node. A node is named *root* if it is the first on the tree. A *level* on the tree represents just a refinement level, and it is enumerated like in Fig. 12.

Remark:

1. *Leaf* nodes on the tree represent non-bisected triangles available in the mesh and *nonleaf* nodes stand for subdivided triangles.
2. To model the entire mesh we have to consider as many trees as triangles in the original mesh, each one having a given root standing for the triangles.
3. Storing the mesh in this hierarchical way makes it possible to check the refinement level. In this sense, a well balanced tree means a uniform refined-region.
4. Provided that triangle sizes in the mesh are similar, trees also offer a way to control mesh smoothness.

5. The neighbors of one triangle can be obtained accessing at most 3 different subtrees, which, as pointed out in Remark 5 for the Modified Adjacency List, also ensures the locality of that data structure.

5. Conclusions

We have reviewed longest-edge algorithms for local refinement of triangular computational meshes and we have proposed an associated graph structure to provide an efficient framework for implementation of these algorithms.

We have focused on Rivara's algorithms in 2D, Forward and Backward Longest Edge Bisection. Although a great number of jobs related to algorithms in 2D are known, the data structures involved, which really set up the kernel of algorithms, have received far less attention.

We think that whatever effort made in that direction will help building software related to this gruelling field of numerical treatment of partial differential equations, representation of regions or computer graphics. Moreover, we think that most of ideas shown here in 2D are aplicable to higher dimension.

The idea pointed out by Plaza and Carey in [10] related to a graph representing the edge classification in a triangle is taken here in order to design a new data structure which let us, in an efficient and easy way, run **LEPP** in both Forward and Backard Longest Edge Bisection of Rivara. Furthermore, we consider some properties that let us prove the goodness of these data structures and the finiteness of that class of algorithms.

Finally, we briefly describe another data strcuture based on trees which make it possible to store the hierarchical way in which the grid is refined or derefined.

Special thanks to Maria Cecilia Rivara for her helpful suggestions and kind support in the preparation of this paper.

6. References

[1] A. V. Aho, J.E. Hopcroft y J.D. Ullman, "*Data Structures and Algorithms*", Addison-Wesley, (1983).

[2] J. Bey, "*Tetrahedral grid refinement*", Computing **55**, 355-378 (1995).

[3] G.F. Carey, M. Sharma, K.C. Wang, A. Pardhanani, "*Some aspects of adaptive grid computations*", Computer & Structures, Vol. **30**, N° 1/2, 297-302, (1988).

[4] L. Ferragut, NEPTUNO, "*A system of adaptive finite element method*", Dept. Mat. Aplic. y Met. Inf., ETSI Minas, Madrid, (1987).

[5] F. Harary, "*Graph Theory*", Addison-Wesley, (1972).

[6] P. Leinen, *"Data structures and concepts for adaptive finite element methods"*, Computing **55**, 325-354 (1995).

[7] A. Liu and B. Joe, *"Quality local refinement of tetrahedral meshes based on bisection"*, SIAM J. Sci. Comput., **16**, 1269-1291, (1995).

[8] A. Mukherjee, *"An adaptive finite element code for elliptic boundary value problems in three dimensions with applications in numerical relativity"*, Phd. Thesis, Penn. State University, University Park, PA 16802, (1996).

[9] E. Muthukrishnan, P.S. Shiakolas, R.V. Nambiar and K.L. Lawrence, *"Simple algorithm for Adaptive Refinement of three-dimensional finite element tetrahedral meshes"*, AIAA Journal, **33**, 928-932, (1995).

[10] A. Plaza and G.F. Carey, *"A new refinement algorithm for tetrahedral grids based on skeleton"*, TICAM Report 96-55 , November, (1996).

[11] A. Plaza, R. Montenegro and L. Ferragut, *"An improved derefinement algorithm of nested meshes"*, in Advances in Post and Preprocesing for Finite Element Technology, M. Papadrakakis ed., Civil-Comp Ltd., 175-180, (1994).

[12] M.C. Rivara and C. Levin, *"A 3-D refinement algorithm suitable for adaptive and multi-grid techniques"*, Comm. in App. Num. Meth., **8**, 281-290, (1992).

[13] M.C. Rivara, *"Mesh refinement based on the generalized bisection of simplices"*, SIAM J. Numer. Anal., **2**, 604-613, (1984).

[14] M.C. Rivara, *"A grid generator based on 4-triangles conforming mesh refinement algorithms"*, Int. J. Num. Meth. Eng., **24**, 1343-1354, (1987).

[15] M.C. Rivara, *"Local modification of meshes for adaptive and/or multigrid finite-element methods"*, J. Comp. and Appl. Math., **36**, 79-89, (1991).

[16] M.C. Rivara, *"New mathematical tools and techniques for the refinement and/or improvement of unstructured triangulations"*, 5th. International Meshing Roundtable'96, Sandia National Laboratories, 77-86, (1996).

[17] M.C. Rivara, *"New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations"*, Int. J. Num. Meth. Eng., **40**, 3313-3324, (1997).

Hexahedral Meshing

Hexahedral Mesh Generation using the Embedded Voronoi Graph

Alla Sheffer, Michal Etzion, Ari Rappoport, Michel Bercovier

Institute of Computer Science, The Hebrew University, Jerusalem 91904, Israel.
{sheffa|michals|arir|berco}@cs.huji.ac.il. http://www.cs.huji.ac.il/{~sheffa|~michals|~arir|~berco}.

Abstract

This work presents a new approach for automatic hexahedral meshing, based on the embedded Voronoi graph. The embedded Voronoi graph contains the full symbolic information of the Voronoi diagram and the medial axis of the object, and a geometric approximation to the real geometry. The embedded Voronoi graph is used for decomposing the object, with the guiding principle that resulting sub-volumes are sweepable. Sub-volumes are meshed independently, and the resulting meshes are combined and smoothed to yield the final mesh.

The approach presented here is general and automatic. It handles any volume, even if its medial axis is degenerate. The embedded Voronoi graph provides complete information regarding proximity and adjacency relationships between the entities of the volume. Hence, decomposition faces are determined unambiguously, without any further geometric computations. The sub-volumes computed by the algorithm are guaranteed to be well-defined and disjoint. The size of the decomposition is relatively small since every sub-volume contains a different Voronoi face. Mesh quality seems high since the decomposition avoids generation of sharp angles, and sweep and other basic methods are used to mesh the sub-volumes.

1 Introduction

Automatic generation of 3-D finite element meshes is essential for the automation of the analysis process. Research and development effort on new meshing algorithms has resulted in several automatic algorithms for 3-D meshing (see [7] for a recent review). Most of the research has focused on the generation of unstructured tetrahedral elements. This has resulted in several successful automatic mesh generation algorithms, which can be categorized according to the main approaches used: Delaunay triangulation [10, 28], octree-based methods [23, 2] and advancing front algorithms [8].

In many situations, hexahedral meshing is more attractive than tetrahedral meshing. The additional requirements of hexahedral meshing make the problem more demanding. Numerous approaches have been proposed and investigated: feature-based [13], medial surface subdivision [17, 18], plastering [3], grid-based [19, 24, 26], whisker weaving [27], etc. As with many difficult problems, each of the proposed solutions possesses both positive and negative attributes associated with the technique employed.

Several successful approaches were developed for different types of geometries of varying complexity. These include mappable and sub-mappable volumes [29], volumes meshable by midpoint subdivision [14], swept volumes and uni-axial combinations of swept volumes [4]. The relative success of these methods indicates that a promising approach towards the general problem is to decompose the volume into parts meshable by existing, well established techniques. This is the approach taken in [17, 18, 13, 12].

In [17, 18], the decomposition is guided by the medial axis of the object. The proposed algorithm gives a template based decomposition, building a template subdivision around each entity of the medial axis (its faces, edges and vertices). This results in an initial hexahedral mesh of the model, which can then be refined to the desired density.

The use of the medial surface for the decomposition provides a systematic and generic approach for all possible geometries. In addition, the decomposition process is directed by the medial surface, thus avoiding the computation

of decomposing surfaces and surface intersections. However, the described technique has several drawbacks. First, because the technique builds a template subdivision around each entity of the medial axis, it results in a fine subdivision of the model, even for simple cases. For example, for a brick (containing no medial surface degeneracies) the initial mesh contains 72 elements. Second, the algorithm used for computing the medial axis [20] is difficult to implement and not provenly correct. Third, for degenerate vertices and edges the suggested use of midpoint subdivision is likely to lead to poorly shaped elements.

In [13], a feature based decomposition of the object is generated. Features are recognized based on combinations of convex and concave edge loops. An attractive property of this approach is that it follows the intuition of manual subdivision. The approach is based on the model's generic shape and is independent of the geometric parameters. Decomposition is in some sense minimal, because the process stops once the parts are convex or can be meshed by sweeping. However, the method possesses several drawbacks. Features with interacting geometry pose a difficult recognition problem. In many cases there are ambiguities when it is not clear which decomposition the algorithm should choose, and decomposition surfaces can cut each other. The result depends upon the order of performing the decompositions. The method requires computations of surface-surface intersections. Finally, convex shapes are not decomposed at all, even when both sweep or midpoint are not suitable.

Contribution. This work presents a new approach for automatic hexahedral meshing, based on the embedded Voronoi graph. The embedded Voronoi graph contains the full symbolic information of the Voronoi diagram and the medial axis of the object, and a geometric approximation to the real geometry. The embedded Voronoi graph is used for decomposing the object, with the guiding principle that resulting sub-volumes are sweepable. Sub-volumes are meshed independently, and the resulting meshes are combined and smoothed to yield the final mesh.

The presented approach possesses several advantages: (1) the algorithm for computing the embedded Voronoi graph is provenly correct, stable and easy to implement; (2) the approach is well defined and valid on shapes of any geometry, including shapes whose medial axis is degenerate and complex convex shapes; (3) the decomposition is order independent and prevents intersections between decomposition surfaces; (4) the number of sub-volumes generated is not large; (5) since the directions and entities involved in each decomposition are defined by the medial axis, there are no intersection computations; and (6) since a decomposition is used, as opposed to template, there is only a minimal need for medial axis geometry.

The main drawback of following the medial axis for the decomposition is that it sometimes can result in over-decompostion of the volume and often is not intuitive.

The paper is organized as follows. In Section 2 we briefly discuss the definition of the embedded Voronoi graph. Section 3 gives an overview of our approach. The main step, shape decomposition using the Voronoi graph, is detailed in Section 4. Generation of the mesh itself is described in Section 5. The application of the algorithm is demonstrated on several examples in Section 6. Section 7 discusses the advantages and drawbacks of the algorithm and suggests topics for future work.

2 The Embedded Voronoi Graph

In this section we briefly describe the embedded Voronoi graph, which is an approximation of the Voronoi diagram and the medial axis of the object. It is used here for decomposing the volume into simple parts. Full details on its definition and construction are given in [6].

Let Q be a volume. The *entities* of Q are the vertices, edges and faces of Q. An entity a is *incident* on an entity b iff one of the following is true: (1) a is an end vertex of the edge b, (2) a is a vertex of the face b, or (3) a is a bounding edge of the face b. Two entities a and b are *adjacent* if they are not incident one on the other, and there is another entity incident on both of them. For example a face and an edge that share a vertex. A Voronoi region R_a is the locus of points that are closer to entity a than to any other entity of Q. The boundaries of the Voronoi regions comprise the *Voronoi diagram* of Q, $VD(Q)$. $VD(Q)$ is comprised of *Voronoi faces*, *Voronoi edges*, and *Voronoi vertices*. The *governors* of a Voronoi element are its nearest entities. In the non-degenerate diagram, a Voronoi face

has two governors, a Voronoi edge has three governors, and a Voronoi vertex has four governors. In a degenerate diagram, edges and vertices can possess more governors. When the faces of the volume are linear, its medial axis can be easily obtained from the Voronoi diagram by deleting Voronoi faces and edges leading to concave vertices and edges of the volume. Constructing the medial axis from the Voronoi diagram is more problematic when the faces of the volume are not linear, since in this case the medial axis might contain elements governed by a single entity. Such elements do not belong to the Voronoi diagram.

The Voronoi diagram of a volume is attractive as a computational tool for geometric applications [1, 16, 25]. The Voronoi diagram is useful due to several reasons. First, its shape is closely related to the shape of the object, thus it can serve as a shape analysis tool. Second, it is of lower dimensionality than the object, thus is easier to deal with. Finally, it explicitly carries proximity information. However, the combination of a complex connectivity structure and the high algebraic degree of its geometric elements makes the construction of the Voronoi diagram of a volume a difficult problem.

Etzion et al [6] defined a set of Voronoi skeletons that approximate the Voronoi diagram of a polyhedron, and provided a simple algorithm to construct them. These skeletons provide information regarding both the symbolic structure of the Voronoi diagram and the geometric location of the diagram's elements. The symbolic structure of the Voronoi diagram contains the complete connectivity structure of the Voronoi diagram together with the governing entities of each Voronoi element. This paper uses the embedded Voronoi graph, which is defined below.

1. **Voronoi Graph.** The Voronoi graph contains all the symbolic information present in the Voronoi diagram, without containing any geometry. The Voronoi graph is a graph whose nodes correspond to elements of the Voronoi diagram. Each node corresponds either to a Voronoi vertex, or to a Voronoi edge, or to a Voronoi face. The label of a node is the set of governors of the corresponding element in the Voronoi diagram. An arc exists between two nodes of the graph if and only if there is an incidence relationship between the corresponding elements in the Voronoi diagram.

2. **Approximate Voronoi Graph.** An approximate Voronoi graph approximates the Voronoi graph of Q to a tolerance of ϵ in the sense that a connected subgraph of the Voronoi graph that lies in a region of space of size smaller than ϵ is replaced by a single graph node. For example, a Voronoi edge whose length is smaller than ϵ might be represented together with its two vertices by a single graph node. The label of this node contains the governors of the edge, together with the governors of its vertices.

3. **Embedded Voronoi Graph.** The embedded Voronoi graph is a Voronoi graph that also provides geometric approximation of specific elements of $VD(Q)$. The embedded Voronoi graph of Q with a parameter δ is a Voronoi graph (or approximate Voronoi graph) of Q s.t. some of the nodes of the Voronoi graph carry also a geometric approximation (of the appropriate type) to the corresponding element in $VD(Q)$, to an accuracy of δ. For example, a graph node that corresponds to a Voronoi edge, contains a polyline that approximates the Voronoi edge to an accuracy of δ.

An algorithm for constructing the embedded Voronoi graph of a polyhedron is given in [6]. Initially, a space subdivision whose cells are labeled according to their proximity to polyhedron entities is constructed. Subsequently, information regarding the symbolic structure of $VD(Q)$ is extracted from the subdivision. This information includes the Voronoi elements of $VD(Q)$, their governors, and their adjacency relationships. If it is known that $VD(Q)$ is not degenerate, then the Voronoi graph is constructed. Otherwise a tolerance parameter ϵ is determined, and the approximate Voronoi graph is constructed. In the final step, a geometric approximation of Voronoi elements of interest is extracted. Because the algorithm is based on space subdivision, a geometric approximation of a specific Voronoi element can be computed locally. The most complex geometric operation performed by the algorithm is intersecting two conic curves. The algorithm is easy to implement and robust. Its convergence and correctness are proven in [6].

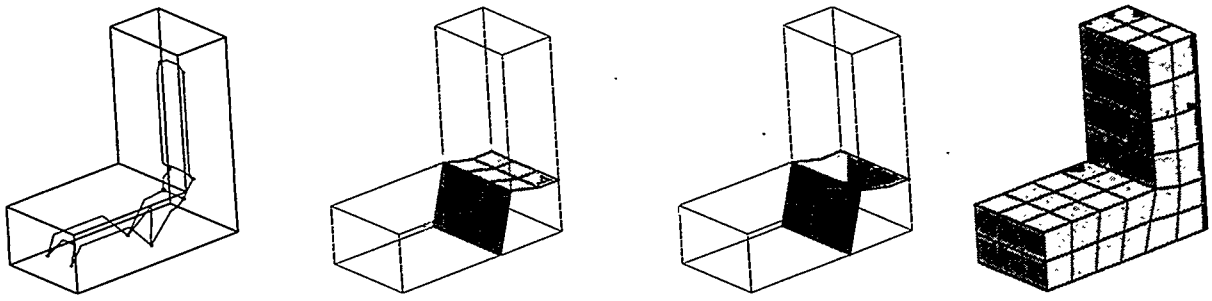
We use the embedded Voronoi graph for decomposing the volume into easily meshable parts. Voronoi faces are used to identify pairs of volume faces that enclose a sweepable volume. Thus, the exact geometry of the Voronoi

faces is not important, only their existence. The Voronoi graph is sufficient for identifying such faces. The volume is actually decomposed by surfaces created by projecting Voronoi vertices and edges on their governors. In order to decide whether a Voronoi element should be projected, only symbolic information is needed, and therefore the Voronoi graph (or the approximate Voronoi graph) is used. This decision is based on the types of the governors of the Voronoi element, and on the adjacency relationships between the governors. If a specific Voronoi element has to be projected, then the geometric location of the element is approximated by the embedded Voronoi graph. The exact location of the Voronoi element is of little importance, since the projection surfaces are not part of the final mesh; their only role is to decompose the polyhedron into simple parts. Thus, the embedded Voronoi graph is suitable for our purposes, and there is no need to compute the exact Voronoi diagram, which is much more difficult.

3 Algorithm Overview

In this section a general overview of the meshing algorithm is presented. The algorithm consists of three main stages: (1) construction of the embedded Voronoi graph of the object, (2) decomposition of the object into simple parts, and (3) actual meshing.

Embedded Voronoi Graph. In the first stage of the algorithm, an embedded Voronoi graph of the input volume is computed (Figure 1(a)). The embedded Voronoi graph displayed in this figure is not symmetric since it gives an *approximation* to the location of the Voronoi elements. This approximation is sufficent for the meshing algorithm. However a finer approximation can be computed if needed. The algorithm for constructing the embedded Voronoi graph operates on linear polyhedra. Therefore, if the input object is not linear, its faces and edges must first be linearly approximated. During this approximation, for each approximation entity its originating entity is stored. After an embedded Voronoi graph is computed, the Voronoi entities whose governors are approximation entities originating from the same model entity are united.



(a) (b) (c) (d)

Figure 1: The stages of the meshing algorithm applied on an `L' shaped volume: (a) the embedded Voronoi graph; (b) the decomposition faces, generated by the projection of Voronoi edges and vertices; (c) the decomposition of the volume into three sub-volumes, showing the decomposition faces after they were merged; (d) the resulting volume mesh (before smoothing).

Object decomposition. This is the major step of the algorithm. The main observation on which this stage is based is as follows. Consider a Voronoi face f governed by two faces a and b of V, where a and b are not adjacent. Consider the volume V_{ab} defined by a, b, and the projection of f to a and b. The volume V_{ab} possesses the following attractive properties:

- V_{ab} is wholly governed by a and b [6], and therefore does not intersect any other entity of the object nor any other element of its Voronoi diagram.

- V_{ab} does not intersect any other volume defined similarly by two faces of the object. Therefore V_{ab} can be meshed independently of other sub-volumes.
- V_{ab} is a sweep from a to b, hence can be meshed by a standard method to mesh sweep volumes.

Denote by f_s a Voronoi face governed by two unconnected faces (the `s' subscript stands for `sweep'), and by V_s a sub-volume containing a Voronoi face of type f_s. Unfortunately, the union of all volumes of type V_s does not cover the whole object V. Denote by V_e a connected volume with no such face (it is a connected component of $V \setminus \cup V_s$).

The embedded Voronoi graph of a volume V thus provides a natural decomposition of the volume into simple parts. V is decomposed into sub-volumes of two types, V_s and V_e. The decomposition is achieved by projecting Voronoi edges that bound faces of type f_s on their respective governors, and creating decomposition faces between the edges and their projections. These faces are then merged to create the faces separating between the sub-volumes. Figure 1(b) shows the decomposition faces resulting from the projections, and Figure 1(c) shows the decomposition faces after they were merged.

In Section 4 the decomposition will be defined such that the following requirements are satisfied:

1. The faces decompose the volume into disjoint sub-volumes, i.e., it is a valid decomposition. This requires the decomposition to be defined at the Voronoi vertices, so that the decompositions induced by the edges that meet at the vertex will be connected correctly.
2. The parts after the decomposition can be meshed by basic algorithms (sweep, mapping or mid-point subdivision): this includes verifying that the decomposition faces are quadrilaterals, and that the angles between adjacent decomposition faces and between the decomposition faces and the original volume faces are not too sharp.
3. The number of resulting sub-volumes is not large.

The decomposition is performed using virtual topology operators, avoiding the complex computations required for actual geometric decomposition and allowing the easy removal of the decomposition surfaces later at the smoothing stage. Virtual topology operators are briefly reviewed in the next section.

Meshing. Finally the volume mesh is constructed by meshing the set of sub-volumes produced by the decomposition. After the sub-volumes are meshed, the decomposition boundaries are removed and partial meshes are united and reassigned to the original volume (Figure 1(d)). A smoothing procedure is then applied on the volume mesh as a whole. The global smoothing relaxes the constraints imposed on the mesh by the geometric positioning of the decomposition faces.

The result of the algorithm is a hexahedral mesh of the volume. The decomposition procedure and the mesh generation that follows are described in detail in the following sections.

4 Volume Decomposition

After the embedded Voronoi graph of the volume is constructed, the volume is decomposed into a set of meshable sub-volumes, by projecting Voronoi edges that bound faces of type f_s on their respective governors and creating decomposition faces between the edges and their projections. Below we discuss which edges and vertices of the Voronoi diagram are of interest for the decomposition and how they should be projected on their respective governors.

The section is organized as follows. In 4.1 a brief overview is given of the virtual topology operators used to perform the editing operations required by the decomposition. Sections 4.2 and 4.3 describe how the projection of the Voronoi edges and vertices is performed, and demonstrate it on all the combinations of Voronoi edge and vertex governors that can occur for non-degenerate Voronoi entities. Section 4.4 explains how the actual volume decomposition is done using virtual topology operators.

4.1 Virtual Topology Operators

To perform the volume decomposition, multiple editing operations are required (projection of Voronoi edges, construction of decomposition faces, splitting of the original model edges, faces and volume, etc.). The geometric computations required by those operations might add a significant overhead to the meshing procedure. This overhead is avoided by using the virtual topology tools reviewed here.

The virtual topology enhancement of the standard boundary representation (B-Rep) was introduced in [22]. It allows a large set of editing operations on the model topology, without changing the actual geometry.

In the standard B-Rep, there is a one-to-one correspondence between topological and geometrical objects, such that any adjustment of the topology requires changing the geometry as well in order to preserve the unique correspondence of topology and geometry.

The virtual topology enhances the B-Rep by uncoupling the topology from the geometry. In order to do this in addition to standard *real* entities, which contain as the description the exact, mathematical definition of the entity's geometry, another type of *virtual* entity is added. The *virtual* topological entities have no exact geometric definition of their own; they rely on other topological entities from which the entity geometric description is to be derived. The structure of this reliance varies depending on the editing operations applied, and the nature of the topology being edited. The virtual topology operators are particularly suitable for model editing performed for mesh generation [21].

4.2 Edge Projection

In order to create the volumes, Voronoi edges are projected on their governing entities in V. For each edge the decision of whether to project it on a specific governor and how to do this projection depends on the type of the governor entity and on the relations between the governors. As observed above, edges that are of interest for the decomposition are those that bound Voronoi faces of type f_s. These are exactly the edges having at least two unconnected face governors. Based on the decomposition goal of creating sub-volumes with one face f_s or none, the decomposition should separate the region governed by f_s from the rest of the volume.

Applying this concept on all types of non-degenerate Voronoi edges (edges governed by three entities in V), we obtain a finite set of cases described below. This classification is an extension of the one given in [17] for Voronoi edges governed only by faces. Types [0], [1] and [2] are defined as in [17], and include Voronoi edges whose governors are all faces. Types [0e] and [1e] are defined for Voronoi edges which have vertex or edge governors as well. The different types of Voronoi edges are defined as follows, and the projection performed for each case is shown in Figure 2.

[0] An edge with three face governors s.t. no pair of governors shares an edge.

[1] An edge with three face governors s.t. a single pair of governors shares an edge. Here a distinction is made between the case [1] where the angle along the shared edge is not sharp, and the case [1s] where the angle is sharp. The distinction between `sharp' and `not sharp' depends on the desired mesh quality.

[2] An edge with three face governors s.t. two pairs of governors share edges.

[1e] An edge with two face governors and an edge governor that is incident on one of the face governors.

[0e] An edge with two face governors and either a vertex governor or an edge governor that is not incident on any of the face governors. Note that this type contains the cases where a face governor is adjacent to the edge governor, but does not contain it.

Other Voronoi edges are of no interest for the decomposition, since they are not incident on Voronoi faces governed by two faces of the volume. An example of such an edge is the edge directly below the concave edge of the `L' shape in Figure 1(a). This edge is governed by the concave edge and the bottom and right faces of the `L'.

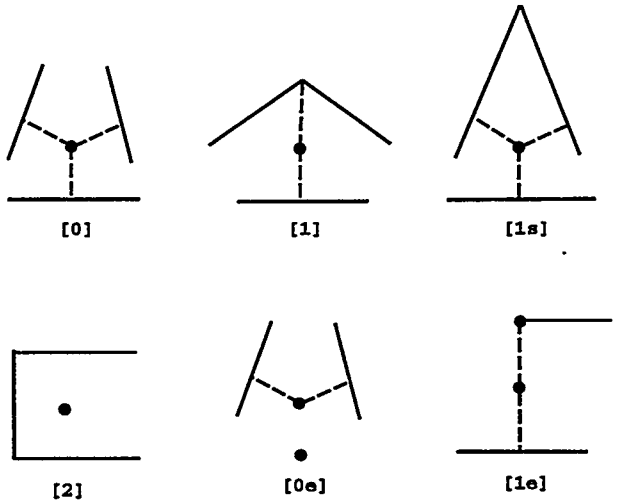


Figure 2: *Projection of the different types of Voronoi edges. The entities of the volume are shown in bold lines, and the decomposition faces added are shown in dashed lines.*

In order to separate the region governed by a pair of unconnected face governors from the rest of the model, the decomposition faces should be added as shown in Figure 2. The intuition behind this projection is the following. The simplest approach would be to project the Voronoi edge on all its governing faces (as done for types [0] and [0e]), but since we want to reduce the number of decompositions and create simpler volumes, the number of decomposition faces can often be reduced. Type [2] is the most common type of a volume boundary region. For example, in Figure 3(a) all the displayed Voronoi edges are of this type. Projection of Voronoi edges of this types will result in a redundant decomposition. Therefore no projection of these edges is done, and as a result, the sub-volumes adjacent to these edges are extended towards the boundary of the volume. In this case additional decomposition faces are sometimes added at the vertices of these edges, to guarantee valid sub-volumes (Section 4.3).

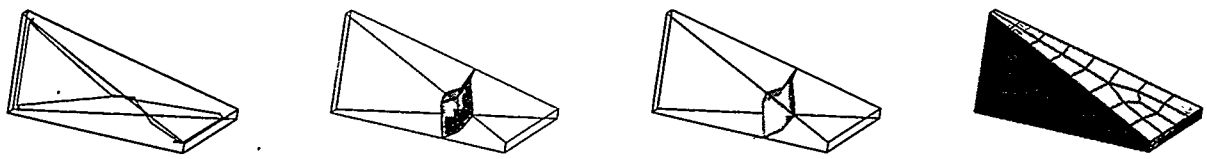


Figure 3: The stages of the meshing algorithm applied on a tapered brick (used in [17]): (a) the embedded Voronoi graph of the volume; (b) the decomposition faces as generated by the edges and vertices projection. In this example the displayed edges are of type [2] and hence the decomposition is performed only at the central vertex, which is of type [2,2,2,2]; (c) the merged decomposition faces and the two resulting sub-volumes; (d) the final mesh before smoothing.

For a Voronoi edge of type [1] to separate the two Voronoi faces of interest (e.g. the Voronoi face governed by the bottom and right faces and the Voronoi face governed by the bottom and left faces), it is sufficient to project the edge on the bottom face and on the common edge of the two upper faces. However, in case of a Voronoi edge of type [1s], this will cause the generation of a low quality mesh and hence the decomposition is performed as shown in Figure 2[1s].

Projection on governors that are not faces does not separate any regions of interest and hence is unnecessary. However, in the case of a Voronoi edge of type [1e], the projection is done on the governing edge and not on the

governor face containing that edge, to simplify the shape of the region governed by the edge and the second face. For example, in Figure 1(a) the type [1e] edges are the one below and the one on the right of the concave edge of the `L' shape. The projection can be seen in Figure 1(b). The projections on the bounding edges of the face (cases [1], [1e]) are viewed as the projections to the relevant governor face.

The same strategy is extended to degenerate edges (i.e. edges with more than three governor entities) as well. Clearly in such cases no finite set of governor combinations exists and the edge projection is determined for each governor face based on its relationship with other governors and their types. The application of the strategy to a degenerate Voronoi edge is shown in Figure 4. The Voronoi edge has four governors: the two vertical (lower) faces and the two edges on top of them. Thus, it is a kind of a [1e] edge, and it is projected on the two governing edges.

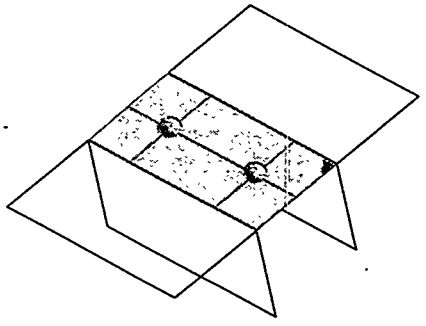


Figure 4: *Decomposition at degenerate Voronoi elements. The dark line shows the degenerate Voronoi edge. The two balls at the endpoints of the edge are the vertices of the edge. This edge appears at the top of the rod of Figure 7. It has four governors: the two vertical (lower) faces, and the two edges on top of them.*

There are two additional types of Voronoi edges that should be considered, although they are not incident on Voronoi faces of type f_s.

[3] An edge with three face governors s.t. three pairs of governors share edges.

[C] An edge whose governors share a common vertex. Such edges are not displayed in the figures showing the embedded Voronoi graph, in order to make the figures less cluttered.

These Voronoi edges are added as special cases to simplify the treatment of Voronoi vertices shared by them and some of the edges addressed above, which are quite common. A Voronoi edge of type [3] indicates a sub-volume that can be meshed by a sweep along the edge. However, in order to simplify the decomposition scheme at the vertices, such edges are treated as edges of type [0], i.e. they are projected on their face governors. A Voronoi edge of type [C] is not projected, but it affects the projection performed at its end vertices as described below.

Clearly, the projection of an edge involves the projection of its end vertices onto the same model entity. In order to decompose the volume using the decomposition along the Voronoi edges, the decomposition has to be extended at the edge end vertices, either by continuing it towards the volume envelope or by combining it with the decompositions defined by other Voronoi edges that share this vertex. The way this is done is explained in the next section.

4.3 Handling of Vertices

In order to decompose the volume along Voronoi edges of interest as described above, the decomposition needs to be defined at the end vertices of the edges.

The vertex treatment includes projecting the vertex on its governor entities, based on the desired projection of the vertex edges on the appropriate governor. Sometimes additional decomposition faces need to be constructed at a vertex to close gaps in the decomposition defined by the edges.

To prevent creation of triangular faces and volumes and avoid unnecessary decompositions, a vertex is projected only on a single location on each governor face. For many edge combinations the projections of the edges match at the vertex with no extra intervention. In the other cases, this involves changing the positions of the end vertex projections as defined by the edges, and sometimes changing the edge projection as well.

When one of the Voronoi edges emanating from the vertex was not projected onto the volume, the decomposition defined by the other edges often needs to be extended towards the volume envelope in order to close gaps.

Below we explain how these operations are applied to the different types of Voronoi vertices. The Voronoi vertices considered are vertices that are incident on Voronoi edges of the types enumerated in Section 4.2. Other vertices are of no interest for the decomposition. The Voronoi vertices are classified according to the Voronoi edges they are incident on. In the non-degenerate case, every Voronoi vertex is incident on four Voronoi edges.

Consider first Voronoi vertices all of whose governors are faces. The edges emanating from these vertices are of one of the following types: [0], [1] ([1s]), [2], [3] or [C]. In this case only a limited number of edge combinations at a vertex exists [17]; the vertex treatment for each of those is shown in Figure 5. The figure shows the decomposition faces at each vertex, including the faces created along each Voronoi edge emanating at the vertex, and the additional faces, added at the vertex, when necessary. Note that the figure is only for visualization of the cases; all the geometry shown is schematic and does not have any relationship to the final mesh.

If all the Voronoi edges emanating from the vertex are of types [0], [3] or [1s], then the projection of the vertex on each governor, as defined by the edge projections, is the same, and no extra adjustments are required. For vertex with edges of type [1], the projection to governor faces needs to be moved to the common edges shared by the governors, to avoid the duplicate projection.

If some of the edges are of type [0] (or [3]), and some are of type [1], then a decision should be taken whether to project the Voronoi vertex according to the [0] (or [3]) edges, or according to the [1] edge. That is, whether to project the vertex on the face (like in [1s] case), or on the edge of the face (like in [1] case). In each specific case (Figure 5) a decision was done as to which option to take s.t. (1) no triangular sub-volumes are created, and (2) no redundant decompositions are done. If the projection at the vertex has to be handled as a [1s] case, then the edge projection will also be performed as [1s].

If the set of edges emanating from a Voronoi vertex includes edges of type [2] and [C], then additional decomposition faces should sometimes be created.

While a Voronoi edge of type [2] is incident on one face of type f_s, an end vertex of such an edge may be incident on other f_s faces, hence additional decomposition at the vertex might be required to separate the regions governed by those faces. If such faces exist (the vertex includes [0],[1], or [3] edges, or four [2] edges), then the decomposition is done as follows. Suppose that an edge of type [2] is governed by faces a, b, c s.t. b is adjacent to a and c. Then, in order to separate the region of the Voronoi face governed by a and c from the rest of the volume, additional decomposition faces are generated between a and b, and between c and b. A decomposition face between a and b is the rectangle $v, \pi_a(v), \pi_{ab}(v), \pi_b(v)$ where v is the Voronoi vertex, and ab is the edge between faces a and b. The Voronoi diagram of the tapered brick in Figure 3 includes a vertex (the central one) of type [2222]. Therefore, to separate the regions governed by the two f_s faces containing it, closing rectangles are generated between the four governors of the vertex.

Voronoi edges of type [C] may also require special treatment at their end vertices. Let a, b, c be the faces governing an edge of type [C]. Since the [C] edge is not projected to its governors, if an edge governed by a and b (which is attached to the same vertex) is projected to a there will be a gap between the decomposition face added and the face c. Such gaps can be handled either by moving the projection points to the shared edge ac, or by creating additional decomposition faces between the vertex and pairs of governors (a and b, a and c, and b and c). Moving is preferable, since it generates fewer decomposition faces. The decision of which approach to take

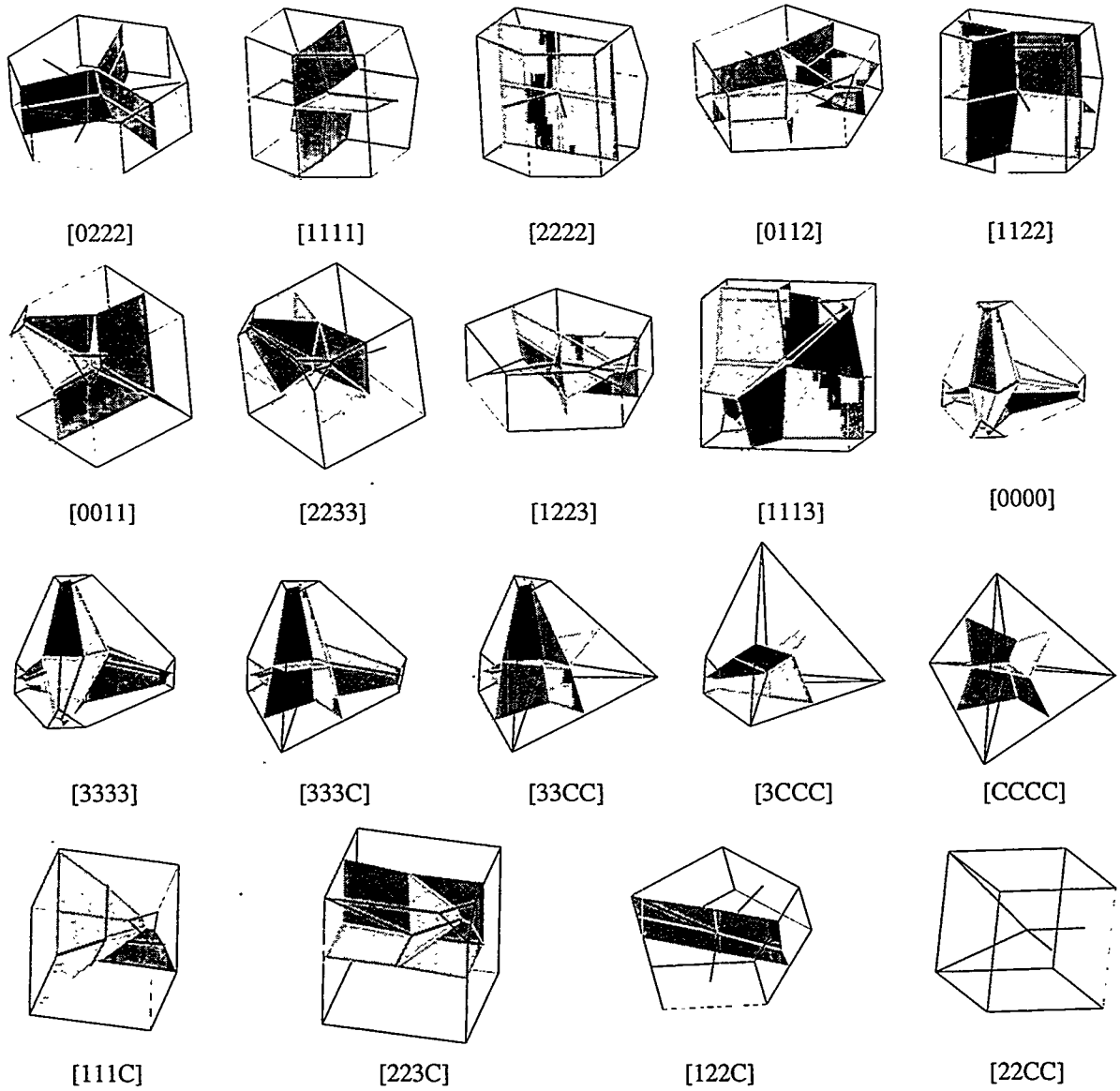


Figure 5: Vertex treatment for non-degenerate vertices with face governors. The decomposition faces are shaded. The Voronoi edges are shown in black. The Voronoi vertex is at the meeting point of the edges. The governor faces boundaries are dark-green. Note: this is only a schematic visualization of the vertices, with no correspondence to the actual sizes of the entities involved.

depends on the type of other edges at the vertex and is shown for each combination in Figure 5.

The treatment of Voronoi vertices with edge or vertex governors is based on the same considerations. We will not detail all types of Voronoi vertices in this case, but rather demonstrate the vertex treatment on a specific example. Consider Figure 1(a) and (b). The internal part of the Voronoi diagram of the volume contains ten Voronoi vertices. Four of them (at the four volume corners) are of type [22CC] and hence no projection or decomposition is performed at them. Two vertices (below the concave edge) are not incident on faces of type f_s and hence no decomposition is performed at them. The remaining four are symmetric and are governed each by three faces and an edge. Consider the vertex v governed by the back-right face (a), the front face (b), the concave edge (d) and the vertical face containing it (c). The Voronoi edge governed by abc is of type [2], the Voronoi edge governed by acd is of type [1e], the Voronoi edge governed by bcd is of type [C], and the edge governed by abd is not incident on a face of type f_s, and hence is not projected anywhere. Based on the edge acd the vertex is projected on a and d. As described for a vertex containing a [2] type edge (abc), an additional projection is performed on b and decomposition faces between a and b and b and c are created (the projection on d is viewed as the projection on c as described above). For the edge bcd to prevent the gaps in projections on b and c/d a rectangle between b and c should be created, but it is already in place.

Voronoi vertices that are degenerate, i.e. vertices with more than four governors, are treated using a similar strategy. An example is shown in Figure 4. The Voronoi edge has four governors: the two vertical faces and the two edges between the vertical and the horizontal faces. Each of its vertices is governed also by the front/back faces of the rod. Therefore each Voronoi vertex has four emanating edges: a degenerate edge projected on the two governing edges, and edges of type [2] and [C]. Therefore rectangles are added between the governing edges and the front/back face.

4.4 The Actual Decomposition

After the set of decomposition faces is generated at the edges and vertices of the embedded Voronoi graph, the original volume needs to be split into a set of volumes using those faces. This part of the decomposition procedure is demonstrated in the examples in Figures 3 and 1 (b) and (c). This is achieved using the following sequence of operations:

- An automatic merging procedure is applied on the set of the decomposition faces, uniting sets of faces that can be united topologically and have no sharp angles between them. An extra constraint that can be added to simplify the meshing is that faces are not merged if the resulting face is not quadrilateral. For example, in Figure 3 (b) and (c) for the [2222] type vertex this results in merging the four faces created at the vertex into a single face.

  After the faces are merged, the face edges are merged as well using a similar procedure.

- A connect operation is now applied on the edges of the new faces and the original model, to connect together coincident edges. For example, in Figure 1(b) and (c) after the generation of the decomposition faces and the merges, there are two edges (from two decomposition faces) coincident to the concave edge of the model. Often this procedure includes also t-vertex connection as edges might coincide only partially.

- After all edges are connected, the decomposition faces need to be incorporated into the volume topology, by splitting the volume edges by decomposition faces vertices lying on them, and then splitting the volume faces by the appropriate edges.

- After all the topological structures are well connected, the volume is split into sub-volumes by the set of faces.

Note that in Figure 1(a) the approximated location of the Voronoi edges and vertices as given by the embedded Voronoi graph is quite far from their actual location. Therefore three of the decomposition faces are very small

(almost collapse into an edge). However the quality of the decomposition, and as a result, the quality of the mesh, are not impaired.

Both the construction of the decomposition faces at the Voronoi edges and vertices and all the operations above are implemented using the appropriate virtual topology operators. This way all the operations are performed only on the model topology and complex geometric computations are avoided. Another advantage is that since the original model geometry remains unchanged, after the set of sub-volumes is meshed the decomposition faces can be removed, and a smoothing procedure can be applied on the whole volume, without the constraints imposed by the geometry of the decomposition faces. The whole procedure can be fully automated.

5 Generation of the Mesh

The mesh of the volume is built by meshing the sub-volumes resulting from the decomposition, and then reassigning the mesh to the original volume and smoothing it. The two stages are described below.

5.1 Meshing the Parts

After the decomposition procedure, two types of sub-volumes are created: volumes of type V_s, i.e., containing a single Voronoi face governed by two unconnected faces (f_i and f_j) of the original volume, and volumes of type V_e, i.e., containing no such surface.

Volumes of type V_s can be meshed by a sweep mesh between f_i and f_j, since all the side faces between the two are quadrilateral:

- The faces generated by the decomposition are always quadrilateral.
- Based on the definition of the decomposition procedure, the only faces (partial faces) of the original volume that can take part in such a volume (besides f_i and f_j) are faces sharing a [2] type edge with f_i and f_j. The treatment of Voronoi vertices incident on a [2] type edge as described above guarantees that those faces are also quadrilateral.

The V_e type volumes can be classified into two types:

1. Volumes resulting from extending the decomposition surfaces at the Voronoi vertices towards the volume faces. Such volumes are always hexahedral, as demonstrated above (Figure 5).
2. Volumes resulting from decomposition along one or more Voronoi edges. From our experience so far such volumes can be meshed by sweep in the direction of the Voronoi edges. However, a formal study of the nature of such volumes must still be done.

In the tapered brick example (Figure 3(d)) the two sub-volumes are meshed by sweeping between the governors of the Voronoi faces present in each. The sub-volume on the left is meshed by sweeping between the front and back faces, and the volume on the right by sweeping between top and bottom faces.

In the L-shape example (Figure 1(d)) there are two V_s type sub-volumes in the two ends of the shape, and a V_e volume at the L-shape corner. The two end sub-volumes are meshed by sweeping between the governor faces, and the corner volume is meshed by sweeping in the direction of the decomposition edges along it.

When meshing the sub-volumes, mesh conformity has to be maintained between the adjacent sub-volumes. This can be achieved by using an interval assignment algorithm on the set of the sub-volume faces [15].

5.2 Smoothing

After the sub-volumes are meshed, the basic problem of generating a hexahedral mesh of the volume is solved. However, the restrictions imposed on the mesh by the geometry of the decomposition faces may affect the mesh

quality. The fact that the computation of the Voronoi diagram geometry is only approximate, also comes into account, resulting sometimes in non-intuitive positioning of the decomposition, and often in non-symmetric decomposition of symmetric volumes (Figure 7(d)). Since the constraints imposed on the mesh by the decomposition are artificial from the user point of view, it would be beneficial to the mesh quality to remove them.

Again, this can be done using virtual operators. The original topology is restored by merging all the entities split by the decomposition: first the sub-volumes are merged into a single volume, then volume faces split by the edges of the decomposition faces are merged back, and finally the volume edges split earlier are merged back. Throughout this procedure the mesh is preserved and is reassigned to the new, merged, entities.

After the original topology is restored, a smoothing procedure can be applied on the mesh of the volume as a whole, thus improving the mesh quality without the decomposition constraints. There is a variety of techniques for performing the smoothing including several `classic' ones as reviewed in [11], and more recent ones such as [5].

6 Results

Most of the algorithm has been implemented. The embedded Voronoi graph part is a stand-alone application running under Unix. The volume decomposition part is implemented using the commercial mesh generator GAMBIT [9], used for sweeping and mapped meshing of the sub-parts. The implementation of virtual topology in GAMBIT was used for the multiple editing and decomposition operations required. The run time of the algorithm is comparable to other mesh methods.

The full implementation of the algorithm is still underway. Currently it lacks the preproccesing and postproccesing stages. The preproccesing stage includes linearization of non-polyhedral volumes. The postprocessing stage includes smoothing the volume mesh.

The algorithm is demonstrated on two complex real life examples. The example models include many of the possible decomposition types described above. The Voronoi diagrams of the models include multiple degenerate edges and vertices, which the algorithm has no difficulty to handle, since the decomposition approach is generic, and is not limited to non-degenerate entities.

The first example is shown in Figure 6. The volume is decomposed into eighteen swept volumes: The six triangular protrusions, the six main faces, and six volumes connecting the main faces. Each of the six triangular protrusions, and the six main faces, contain a single Voronoi face of type f_s, and therefore a sweep is done between the faces of the volumes governing the Voronoi face. The six volumes connecting the main faces are swept along their Voronoi edges. The final mesh is shown in Figure 6(d).

In Figure 7, the volume can not be meshed by sweep or uni-axial combination of sweeps, but requires the use of either decomposition or generic meshing algorithms. The Voronoi diagram of the volume contains many degenerate edges and vertices. The degenerate edge at the top of the part is shown in Figure 4. The volume is decomposed into swept volumes (the six parts of the bottom wheel, and the side parts on the top) and mappable volumes (Figure 7(c)). The final mesh is shown in Figure 7(d). As can be observed, even without the final smoothing the mesh quality is very high.

This example shows a drawback of the use of the medial axis for mesh generation: its sensitivity to scaling. The decomposition faces inside the wheel are redundant, since the wheel as a whole is a sweepable volume. These decomposition faces would have been avoided if the wheel had been narrower. In that case, there would have been a single Voronoi face in the wheel, a face governed by the two sides of the wheel (as occuring in the two sides at the top of the volume). In this case the whole wheel would have been a single sub-volume in the decomposition.

7 Discussion

In this paper we presented a hexahedral mesh generation algorithm. The algorithm uses the embedded Voronoi graph of the volume to decompose the volume into simple parts that can be meshed using basic meshing methods.

The approach presented here is general and automatic. It handles any volume, even if its medial axis is degenerate. The embedded Voronoi graph provides complete information regarding proximity and adjacency relationships between the entities of the volume. Hence, decomposition faces are determined unambiguously, without any further geometric computations. The sub-volumes computed by the algorithm are guaranteed to be well-defined and disjoint. The size of the decomposition is relatively small since every sub-volume contains a different Voronoi face. Mesh quality seems high since the decomposition avoids generation of sharp angles, and sweep and other basic methods are used to mesh the sub-volumes. The decomposition directions depend on the object and not on an arbitrary external coordinate system, as in octree-based methods.

Hexahedral 3-D mesh generation using the medial axis has been presented in [17, 18]. The present work builds upon that work, and uses a classification of Voronoi elements similar to that defined in [17]. The advantages of the present algorithm over the one of [17, 18] are the following.

1. The number of pieces created by the present algorithm is much smaller, because the decomposition here is performed only in order to separate regions governed by faces of type f_s from regions governed by other Voronoi diagram faces. For example, a rectangular volume will not be decomposed at all (since it either contains a single f_s face, or in the degenerate case of a cube no faces at all), as opposed to a subdivision into 72 hex sub-volumes given by [17].

2. The algorithm uses the embedded Voronoi graph instead of the medial axis (or Voronoi diagram) of the volume. Computing the exact medial axis of a linear polyhedron requires solving systems of tri-variate quadratic equations, resulting in algorithms that are not robust, difficult to implement, and difficult to prove correct. The most complex geometric operation performed by the algorithm to construct the embedded Voronoi graph is intersection of two conic curves. The algorithm is simple to implement, and proven correct.

   The information given by the embedded Voronoi graph is sufficient for the present algorithm. The embedded Voronoi graph gives correct symbolic information; there is no need for exact geometric information, since the projection faces emanating from Voronoi elements are not part of the final mesh. Their only role is to decompose the volume into simple parts that are later meshed using basic meshing methods. A smoothing procedure is then applied to the volume mesh as a whole, making the exact location of the decomposition faces even less important. This stands in contrast to [17, 18], where exact information of the Voronoi elements locations and radii is used to define the mesh.

3. The algorithm handles volumes with a degenerate medial axis. The same strategy is applied to degenerate and non-degenerate volumes. The determination of decomposition faces is identical, and thus also the decomposition into sweepable sub-volumes. In [17, 18], in the non-degenerate case each primitive obtained is one of the 13 types defined, and an appropriate mesh procedure can be applied according to the type of the primitive. In the degenerate case the method creates primitives that are not part of the existing set. Therefore a midpoint subdivision is applied to these primitives. However, in the degenerate case there might be many elements meeting at the primitive center, resulting in poorly shaped elements.

The approach presented in this paper has some drawbacks, whose removal should be investigated in the future. The most conspicuous drawback is that the medial axis is sensitive to scaling, a fact which can result in over-decomposition of the volume. For example, in Figure 7 there are two locations where the decomposition faces obtained are redundant: the wheel (which is a sweepable volume as a whole), and the two corners at the top. In both cases a scaling of the volume would eliminate the redundant decomposition faces (perhaps creating new redundant faces in other places.) Another direction which should be investigated in order to minimize over-decomposition is a post-processing stage that merges adjacent sub-volumes into larger sweepable volumes.

The algorithm presented in this paper decomposes the volume into sub-volumes that are later meshed by basic meshing methods. We have shown that most of the sub-volumes are sweepable or hexahedral (Section 5.1). However, there is one set of sub-volumes that has not been proven to be either sweepable or hexahedral. These are the sub-volumes of type V_e that result from decomposition along one or more Voronoi edges (Section 5.1, item 2).

From our experience so far, such volumes can be meshed by sweep in the direction of the Voronoi edges. However, further study should take place to either prove this conjecture or to define further decomposition of such volumes to ensure that the volumes obtained can be meshed by available basic algorithms.

An important extension to the algorithm presented in this paper is handling of non-polyhedral volumes. If the medial axis of the non-polyhedral volume does not include points that have multiple projection points on a single volume entity, then it probably suffices to implement the procedure described in Section 3 (part 1). This includes approximating the entities of the volume by linear entities, computing the embedded Voronoi graph of the linear approximation, and then uniting Voronoi elements that are governed by the same original entities. In order to tackle the complete domain of non-polyhedral volumes, volumes with Voronoi elements with multiple projection points on an entity should be handled as well.

Acknowledgments. Part of this research was supported by the Israeli Ministry of Science, the program for development of scientific and technological infrastructures.

References

[1] Armstrong C.G., Modeling requirements for finite-element analysis, *Computer-Aided Design*, 26(7):573–578, 1994.

[2] Bern, M. ., Eppstein, D., Gilbert, J.R., Provably Good Mesh Generation, Journal of Computer System Sciences, vol 48, pp. 384-409, 1994.

[3] Blacker T.D., Meyers R.J., Seams and wedges in plastering: a 3D hexahedral mesh generation algorithm, Engineering with computers, 9, 1993, pp.83-93.

[4] Blacker, T. D., 1996, The Cooper Tool, 5th International Meshing Roundtable, SAND 95-2130, Sandia National Laboratories.

[5] Canann S. A., Stephenson M. B., Blacker T., Optismoothing: An optimization-driven approach to mesh smoothing, Finite Elements in Analysis and Design, Elsevier Science Publishers, Num 13, pp.185-190, 1993

[6] Etzion, M., Rappoport, A., Computing Voronoi Skeletons of a 3-D Polyhedron by Space Subdivision, Technical Report TR-8-97, Institute of Computer Science, The Hebrew University of Jerusalem, 1997.

[7] Field, D. A., The legacy of automatic mesh generation from solid modeling, Computer Aided Geometric Design, 12(7), November 1995.

[8] Frey, P.J., Borouchaki, H., George P.L., Delaunay Tetrahedralization using an Advancing-Front Approach, 5th International Meshing Roundtable, SAND 95-2130, Sandia National Laboratories. 1996.

[9] GAMBIT 1.0 CFD preprocessor, User's Guide, Fluent Inc., 1998.

[10] George, P.L., Improvements on Delaunay-based three-dimensional automatic mesh generator, Finite Elements in Analysis and Design, 25, 1997.

[11] Ho-Le K., Finite Element Mesh Generation Methods: A Review and Classification, Computer Aided Design, Butterworth & Co. Ltd, Vol 1, Num 20, pp.27-38, Jan/Feb 1988

[12] Holmes, D.I., 1995, Generalized Method of Decomposing Solid Geometry into Hexahedron Finite Elements, proceedings, 4th International Meshing Roundtable, Sandia National Laboratories.

[13] Liu, S. S., Gadh, R., Automatic Hexahedral Mesh Generation by Recursive Convex and Swept Volume Decomposition, 6th International Meshing Roundtable, Sandia National Laboratories, pp. 217-231, 1997.

[14] Li, T.S., McKeag, R.M., Armstrong, C.G., Hexahedral Meshing using Midpont Subdivision and Integer Programming, Computer Meth. App. Mech. Eng., Vol 124, pp. 171-193, 1995.

[15] Mitchel S., High Fidelity Interval Assignment, 6th International Meshing Roundtable, Sandia National Laboratories, pp. 33-44, 1997.

[16] Patrikalakis, N.M., Gursoy, H.N., Shape interrogation by medial axis transform, Advances in Design Automation, Vol. 1: Computer Aided and Computational Design, B. Ravani (ed.), ASME, 1990.

[17] Price, M. A., Sabin, M. A., Armstrong, C. G., 1995, Hexahedral Mesh Generation by Medial Axis Subdivision: Part I. Solids with Convex Edges, Int. J. of Numerical Methods in Eng., Vol.38.

[18] Price, M. A.and Armstrong, C. G., 1997, Hexahedral Mesh Generation by Medial Axis Subdivision: Part II. Solids with Flat and Concave Edges, Int. J. of Numerical Methods in Eng.,, Vol 40, pp. 111-136.

[19] Schneiders, R., 1996, "A Grid Based Algorithm for the Generation of Hexahedral Element Meshes", Engineering with Computers, Vol. 12, No. 3-4, pp. 168-177.

[20] Sheehy D.J., Armstrong C.G., Robinson D.J., 1995, Computing the medial surface of a solid from a domain Delaunay triangulation, ACM Symposium on Solid Modeling Foundations and Applications, Utah, pp. 201-212.

[21] A. Sheffer, T. Blacker and M. Bercovier, Virtual Topology Operators for Meshing, Proc. 6th Intl. Meshing Roundtable, pp.49-66, October 1997. To appear in International Journal of Computational Geometry and Applications.

[22] Sheffer, A., Blacker, T. D., Clements, J., Bercovier, M., Virtual Topology Construction and Applications, Geometric Modeling Theory and practice. pp. 247-259, Springer-Verlag, 1997.

[23] Shephard, M.S. and Georges, M.K., Automatic Three Dimensional Mesh Generation by the Finite Octree Technique, International Journal of Numerical Methods in Engineering, vol 32, pp 7-9-749, 1991.

[24] Smith, R. J., Leschziner, M., A., A Novel Cartesian Grid Method for Complex Aerodynamic CFD Applications, 5th International Conference on Numerical Grid Generation in Computational Field Simulations, Ed B. K. Soni, J. F. Thompson, J. Hauser, P. Eiseman, Mississippi State University, 1996.

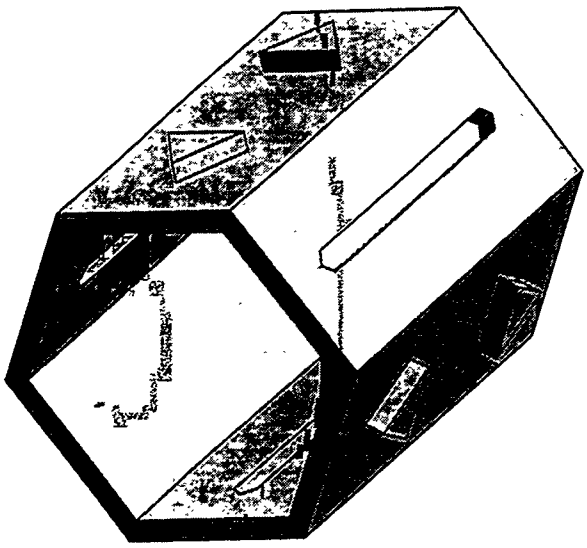
[25] Storti D.W., Turkiyyah G.M., Ganter M.A., Lim C.T., Stal D.M., Skeleton-based modeling operations on solids, *ACM Symposium on Solid Modeling Foundations and Applications*, Atlanta, May 1997, pp. 141–154.

[26] Taghavi, R., Automatic, Parallel and Fault Tolerant Mesh Generation from CAD, Engineering with Computers, Springer-Verlag, Vol 12, pp. 178–185, December 1996.

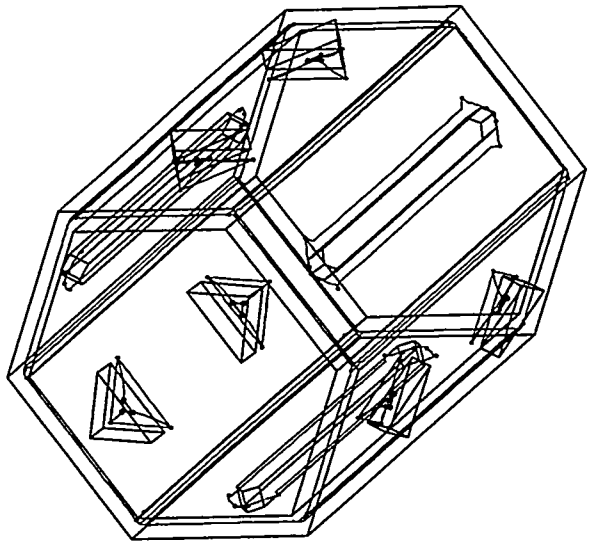
[27] Tautges, T. J., Blacker, T. D., Mitchell, S. A., 1996, The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes, Int. J. of Numerical Methods in Eng., Vol.39 No 19.

[28] Weatherill, N. P. and Hassan, Efficient Three-dimensional Delaunay triangulation with Automatic Point Creation and Imposed Boundary Constraints, Int. J. Numerical Methods in Engineering, vol 37, pp. 2005-2039, 1994.

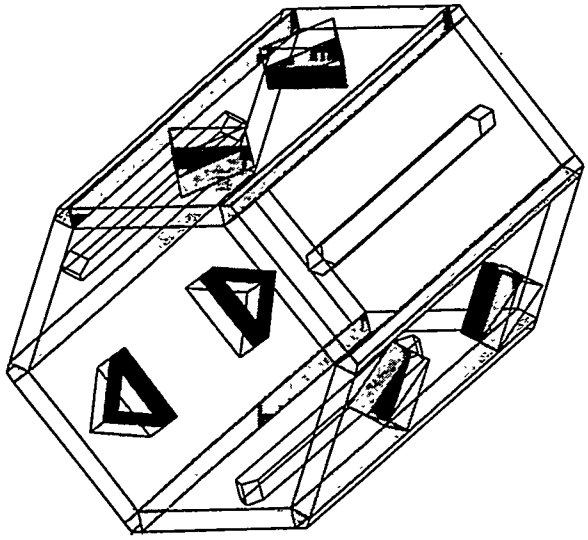
[29] White, D.R., Lai, M., Benzley, S.E., Sjaardema, G.D., 1995, Automates Hexahedral Mesh Generation by Virtual Decomposition, 4th International Meshing Roundtable, SAND95-2130, Sandia National Laboratories.



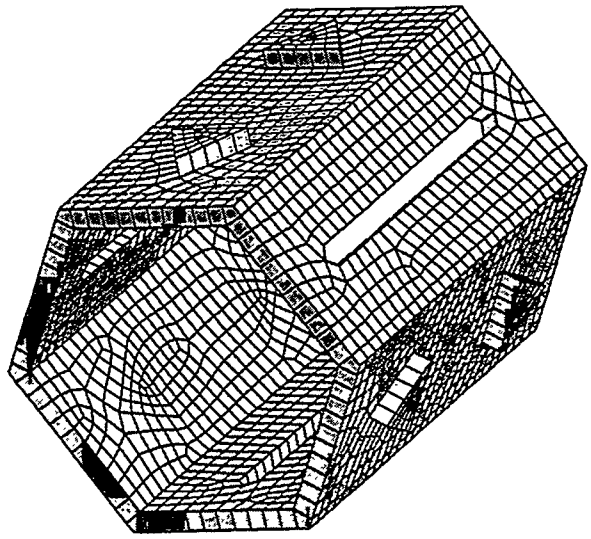
(a)



(b)



(c)



(d)

Figure 6: A meshing example: (a) the initial volume; (b) the embedded Voronoi graph of the model; (c) the decomposition faces generated based on the embedded Voronoi graph; (d) the final mesh before smoothing.

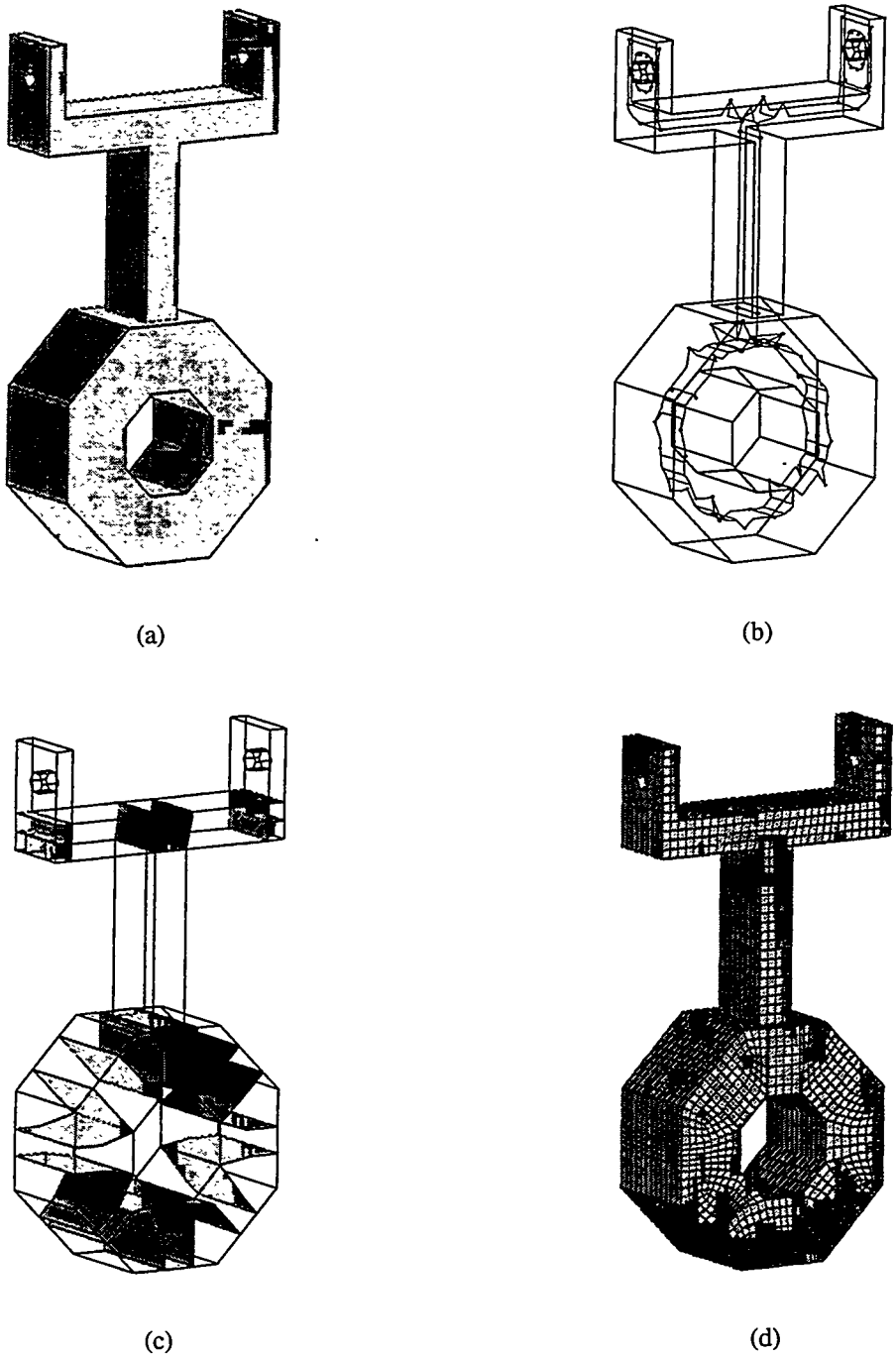


Figure 7: An additional example: (a) the initial volume; (b) the embedded Voronoi graph of the volume; (c) the decomposition faces generated based on the embedded Voronoi graph; (d) the final mesh before smoothing.

Reliable Whisker Weaving via Curve Contraction

Nathan T. Folwell and Scott A. Mitchell[1]

Abstract. Whisker Weaving is an advancing front algorithm for all-hexahedral mesh generation. It uses global information derived from grouping the mesh dual into surfaces, the STC, *to construct the connectivity of the mesh, then positions the nodes afterwards. Currently we are able to reliably generate hexahedral meshes for complicated geometries and surface meshes. However, the surface mesh must be modified locally. Also, in large, highly-unstructured meshes, there are usually isolated regions where hex quality is poor. Reliability has been achieved by using new, provable curve-contraction algorithms to sequence the advancing front process. We have also demonstrated that sheet moving can remove certain types of invalid connectivity.*

keywords. hexahedra, mesh generation, advancing front, topology, curve contraction

1. Introduction

The finite element method (FEM) is effective for studying a wide variety of physics. Before a FEM analysis can be performed, however, a mesh of the model must be generated. Of particular interest to Sandia National Laboratories is structural mechanics simulations, often coupled with electromagnetism and radiation. For these problems, it is important that the mesh be conformal, be composed of hexahedra, and have high quality near the boundary. At Sandia, simulations are often performed on models that are different from typical industrial models; these models often have hundreds of interlocking parts, surrounded by a potting material whose geometry is the complement of the union of the other parts. The parts often have simple geometries, but the potting itself has a complex geometry that structured and semi-structured meshing algorithms have little hope of addressing. Decomposing the potting geometry into simple pieces is helpful but tedious since automatic decomposition tools are not yet mature. Obtaining a conformal mesh is difficult. Even with careful manual decomposition it is difficult to keep the sweep directions of 2.5-dimensional algorithms from colliding. Meshing complex models would be much easier if we had an algorithm for generating unstructured hexahedral meshes from a fixed bounding surface mesh. We envision that most parts would be meshed with a (semi-) structured algorithm, and the general meshing tool would address parts where sweep directions collide, and parts such as the potting that are geometrically complex and have lower quality requirements. Plastering[13] and Whisker Weaving[23] are two general meshing tools that are being developed by the CUBIT project at Sandia to meet these requirements.

Both Plastering and Whisker Weaving are based on the advancing front approach, which is motivated by the need to conform to a fixed surface mesh and for high quality near the boundary. Without both of these simultaneous requirements, alternatives abound. In [1][16][17], the medial axis is used to subdivide volumes into simpler subregions which are then meshed with a structured approach. Mesh quality is good, but the mesh doesn't necessarily conform to a fixed surface mesh. The octree or grid based approaches used in [18], [19] and [20] start with a regular mesh and then adapt the outer elements to the boundary of the volume. A new approach is to recursively insert a layer of hexes separating the volume into two smaller sub-volumes[3]. Another approach in [14] uses the dual to transform the graph of the surface mesh by placing layers of hexes hugging the boundary. The layer is removed from the problem, leaving a surface mesh with one less dual cycle. Eventually the surface mesh of a single hexahedron is obtained. The approaches [3], [14], and the current form of Whisker Weaving all concentrate on building the mesh one layer at a time. The method in [22] relies on pattern recognition, local mesh refinement and coarsening, and variational mesh smoothing techniques. Plastering advances the front based on geometric tests; for complicated geometry, however, it has trouble closing the connectivity on the interior of the mesh. Recent results on stopping with a well-shaped left-over region (*void*) followed by filling the void with tetrahedra appear promising.[12][13] [26] In contrast to Plastering, Whisker Weaving advances the front based on connectivity information inherent to a global grouping of the dual. Geometric criteria near the boundary are of secondary importance. In this interim report, we describe how Whisker

1ntfolwe@sandia.gov and samitch@sandia.gov, http://endo.sandia.gov/~samitch. Parallel Computing Sciences Dept., Sandia National Laboratories, Albuquerque, NM 87185. The authors were supported by the Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy, Office of Energy Research, and work at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AL04-94AL8500. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE.

Weaving is able to reliably generate an all-hexahedral mesh for large and complex geometries. Currently, we must modify the surface mesh. Except for ensuring that the surface mesh has an even number of quadrilaterals, this is non-fundamental; we have an algorithm on paper for fixed surface meshes, but have not yet implemented it. This algorithm will probably produce some degenerate hexes called *knives*. Another drawback is that, despite better algorithms for removing degenerate connectivity (described here) and improving smoothing [8], the quality of the hexes are often unacceptable. For example, often isolated hexes have negative Jacobians. We plan to develop a hex mesh improvement algorithm based on a set of local connectivity-swapping operations and smoothing, as is commonly done for triangular[2], tetrahedral[6] and quadrilateral meshes. [7][21]

Given a surface mesh satisfying mild conditions, the existence of a hexahedral mesh conforming to it has been proven.[5][10][25] We developed our algorithm loosely based on the constructive steps of Mitchell's proof.[10] The proof is based on the Spatial Twist Continuum (*STC*) [15] which is the observation that the dual of a quadrilateral mesh can be grouped into an arrangement of curves, and the dual of a hexahedral mesh can be grouped into an arrangement of surfaces; see Figure 1. Conversely, given certain conditions on the arrangements, a curve arrangement dualizes into a quad mesh and a surface arrangement dualizes into a hex mesh. The problem of extending a quad mesh into a hex mesh then becomes the problem of extending the arrangement of dual curves into an arrangement of surfaces, and then fixing the arrangement to ensure that the surfaces dualize to a hex mesh.

Mitchell's proof [10] states that there exists a way to extend dual curves to dual surfaces by simultaneously manipulating the topological properties of the curves and extending the surfaces into the volume. These topological operations on the curves reduce the number of intersections among the curves. A curve shrinks to a point and disappears from the problem when it no longer intersects any other curve. For this reason, we refer to the algorithm as the *curve contraction* algorithm. This paper describes a new, provably-correct algorithm for carrying out this extension, and our robust, effective implementation. Currently we have implemented the curve-contraction algorithm for *simple* (not self-intersecting) curves and a local pre-processing algorithm that perturbs the surface quad mesh so that the dual curves are simple. Mitchell's proof also describes how to incrementally add surfaces to an arbitrary arrangement so that it dualizes to a hex mesh. This paper describes our implemented algorithm for these fix-ups that produces better quality and fewer hexes than the straightforward translation of the proof.

The remainder of the paper is organized according to the flow of the algorithm. In section 2, we recall the STC. In section 3, we describe how to perturb the surface mesh to remove dual-curve self-intersections. In section 4, we present the main result of this paper, our new algorithm for creating a surface arrangement by contracting curves. In section 5, we describe fix-ups for converting this arrangement to a reasonable mesh. Section 6 gives examples of good-quality meshes of small geometries, and bad-quality meshes of large geometries. Section 7 describes our plan for overcoming Whisker Weaving's current limitations. Conclusions follow in section 8.

2. STC Definitions

Whisker Weaving is based on the Spatial Twist Continuum, STC, the dual of a hex mesh grouped into surfaces or sheets in general position.[15] Each *sheet* is dual to a layer of hexes. The intersection of two sheets is a *chord*, the dual to a column of hexes. While the mesh is being formed, the dangling end of a chord is a *whisker* and is dual to a quadrilateral face on the meshing front. The intersection of three sheets is a *vertex*, the dual of a hex. See Figure 1. A quadrilateral surface mesh also has an STC, an arrangement of curves or *loops* in general position. A loop is defined by recursively passing from one mesh edge of a quad to the edge opposite it, until encountering the first edge or the boundary of the surface mesh. For our purposes, the surface mesh has no gaps and completely encloses the volume, so all loops will be closed. The intersection of two loops is a *point* dual to a quadrilateral.

Note that the primal and dual contain the same information, but grouping the dual into the curves and surfaces of the STC illuminates global information about how a surface mesh constrains the possible interior volume meshes. We have also found that the STC is a more flexible datastructure than the primal; the STC can generically describe certain configurations that don't dualized to a well-defined hex mesh. These configurations are useful as intermediate results, as they can be fixed later.

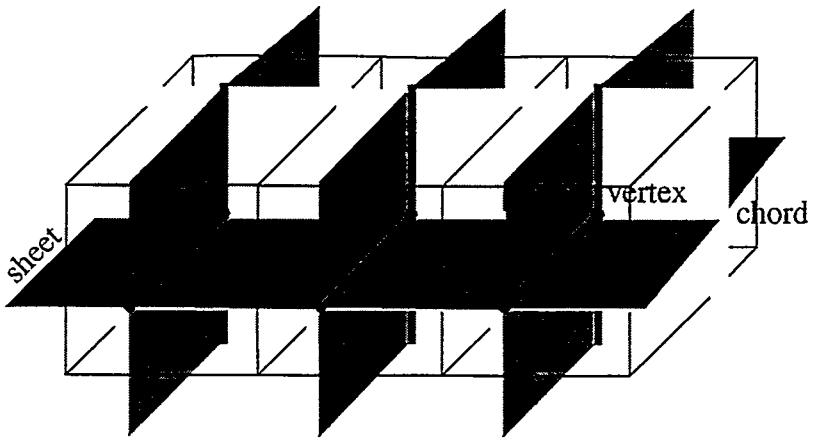


Figure 1. Three hexes and their STC.

3. Modifying the Surface Mesh to Remove Self-Intersections

We now describe how to locally modify the surface mesh so that its *loops* (dual curves) are simple. This condition is required by the current implementation of our curve contraction algorithm, section 4, but will be removed soon. The secondary goals of surface modification are to maintain good quality quadrilaterals and to change the surface mesh as little as possible.

The fact that a loop has a self-intersection is a global property that one usually wouldn't notice by looking in a small neighborhood of the self-intersection. Self-intersections arise because of unstructured surface meshes and non-rectilinear geometry, and are quite common.

The basic operation is collapsing a quadrilateral of self-intersection into two edges; see Figure 2. A *pillow* (ring of quadrilaterals) is occasionally inserted to improve the edge-valence of nodes or to meet geometric constraints; see Figure 3. Note that neither of these operations affect the topology of any other loop, so each loop may be handled in turn. The algorithm is a greedy heuristic, collapsing the most favorable face of self-intersection in the most favorable way and recursing on the resultant loop or loops. The algorithm always succeeds in removing self-intersections, but in models with thin features the surface mesh quality is sometimes poor. We suspect that minimizing the number of collapses and similar problems are NP-Complete. Another basic operation that may be useful is opening a face, but we have not explored this.

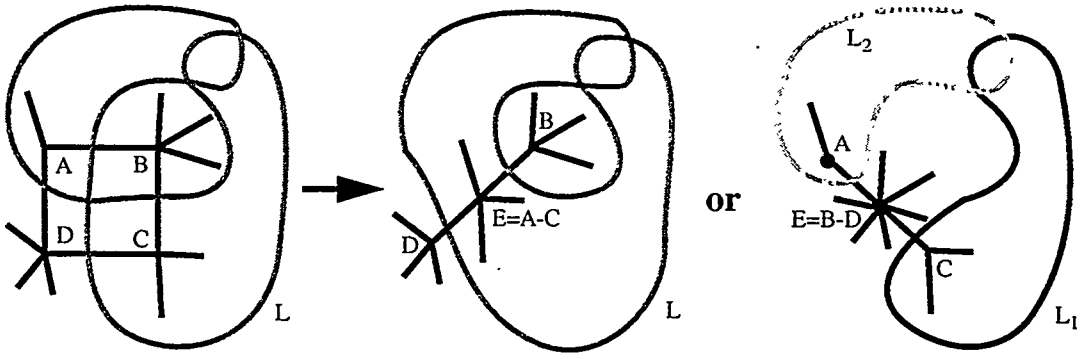


Figure 2. Two choices for collapsing: The edge-valence may differ, and the loop either remains one or splits into two.

3.1 Basic face-collapsing operation

Consider collapsing a quad into two edges by merging nodes A and C into E as in Figure 2. Let $V(X)$ denote the *edge-valence* of $\overline{X}$, the number of edges containing X. The edge-valence of B and D decreases by one, and $V(E) = V(A) + V(C) - 2$. If the valence of node X becomes 2, then we insert a pillow around each of the nodes that are face-opposite X; see "Pillowing Doublets..."[11] and Figure 4 center. Similarly, regardless of edge-valence, we pillow if the interior angle between edges is nearly obtuse; see Figure 3 right. Also, if both A and C lie on curves or vertices and not a surface, then we must place a pillow around the face before collapsing it; see Figure 4.

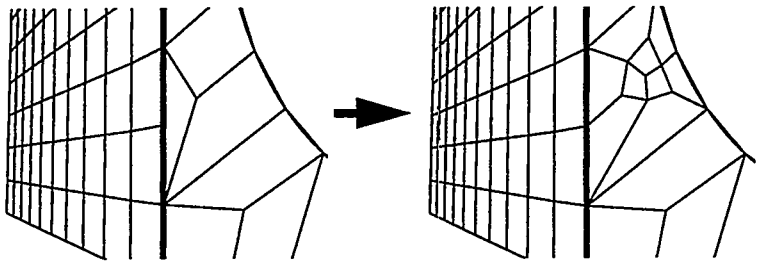


Figure 3. If an angle is nearly obtuse, we pillow all quads attached to the node opposite the large angle. The angle might be large due to poor smoothing or geometric constraints.

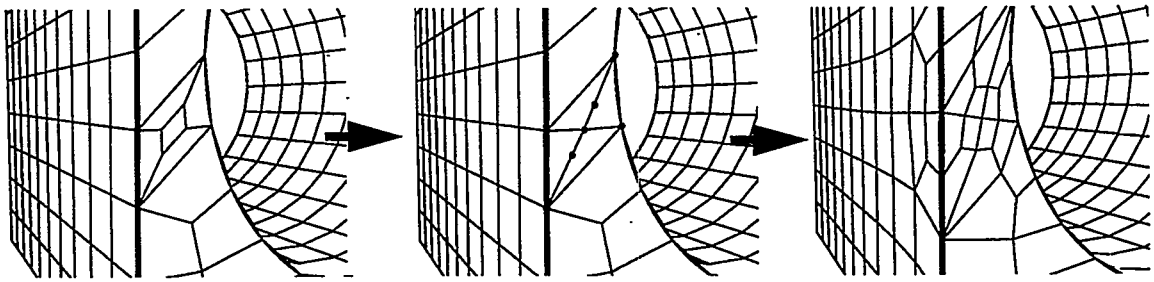


Figure 4. If both merging nodes of a collapsing face don't lie on a geometric surface, e.g. each lies on a curve, the face must be pillowed (left) before collapsing (center). Finally, the nodes opposite the resulting 2-valent nodes are pillowed (right).

3.2 Choosing which faces to collapse

Note that each face of self-intersection can be collapsed in two different ways. One way splits the loop L into two, L_1 and L_2, and the other way retains one loop; see Figure 2. If the loops splits into two, then points where L_1 and L_2 intersect are no longer self-intersections and need not be collapsed. Because of this we typically only have to collapse a small number of faces, which seems to grow as the cube root of the total number of self-intersections.

We favor collapsing faces such that no pillowing is required; among faces that all need or all don't need pillowing, we favor collapsing a face if the resultant nodes have edge-valence closer to 4; see [2]. Also, since collapsing a face such that the loop splits into two avoids having to collapse certain other faces, we favor a face if collapsing those other faces would have resulted in edge-valence far from 4. We conclude this section with a difficult example, Figure 5.

4. Creating Surface Arrangements by Contracting Simple Curves

Whisker Weaving is based on constructing an arrangement of *sheets* (STC surfaces) dual to layers of hexes, by iteratively performing basic weaving operations which are local. Previously, Whisker Weaving relied on a *state list* [23] similar to a first-in first-out queue to determine the order in which weaving operations should be tried. Herein we describe an alternative approach that has proven to be more reliable for larger problems. It is based on the observation that contracting a curve to a point on a shrinking sphere sweeps out a surface behind it inside the ball.[10] For Whisker Weaving, the initial curves are loops of the surface-mesh STC, and the swept out surfaces are the *sheets* of the hex-

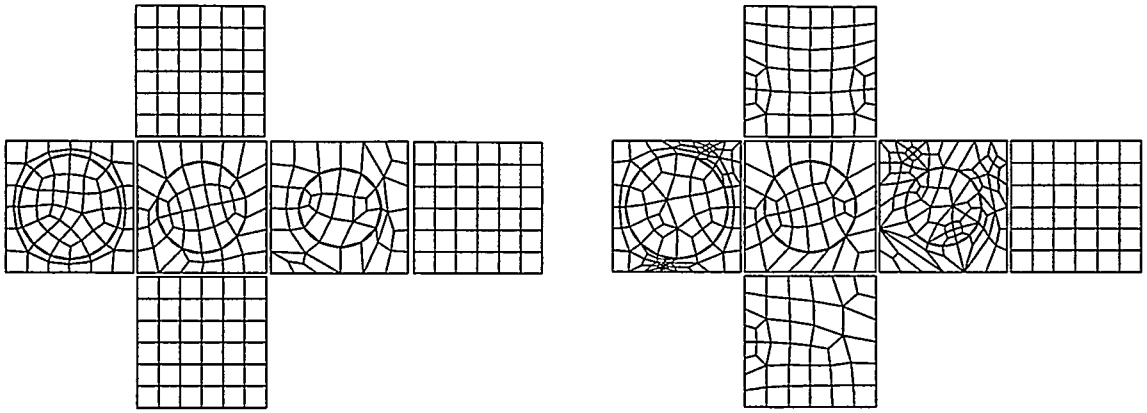


Figure 5. The unfolded surface mesh of a cube with imprinted circles. Left is the initial surface mesh, and right is after collapsing loop self-intersections and pillowing. In the initial surface mesh there are a total of six surface loops. Two of these six loops pass through every quad, which is typical of unstructured meshes. The thin regions around the circles make this is a difficult example for our algorithm. Of the 272 initial quads, 90 are self-intersections. Collapsing just 16 quads removed all self-intersections, and pillowing added 108 quads.

mesh STC. A shrinking curve is dual to the advancing front; as a curve is shrunk, or passed over by another shrinking curve, hex-duals are created the as front is advanced. Here we are only interested in the topological description of a sheet, namely the chords and vertices of intersection with other sheets, and not its geometric embedding.

Whisker Weaving chooses a curve to shrink and decides whether to shrink it to the left or right, then shrinks it completely. This is repeated until all curves are shrunk. Shrinking a curve is accomplished by *collapsing cells* on its boundary until no cell remains. Collapsing a cell means moving the curve so that no point of the cell is *inside* (to the right of) the curve any longer. Each cell is collapsed by a combination of three weaving operations. Both the state list and the curve contraction implementation rely on local connectivity rules not only to determine if a desired weaving operation will result in a reasonable mesh, but to automatically seam together parts of the front (the *join* operation below) and to resolve certain other degeneracies. In Figure 6, we have provided a simple example illustrating the flavor of the algorithm. A *cross* passes a curve over an intersection point. A *join* removes the empty overlap between two curves. *Cross* and *join* are defined more fully in section 4.1.

4.1 Curve Contraction Algorithm

Weaving Operations. We use three basic weaving operations in our curve shrinking algorithm; see Figure 7. In the following description we use the terminology introduced in [23] and outlined in section 2. The first operation, used for collapsing a cell of size two, is a *join*. In the STC, it corresponds to joining the whiskers (dangling ends) of two chords together to make one chord. In the primal, it corresponds to seaming two faces of neighboring hexes together into one face. The second operation, used for collapsing a cell of size three, is a *cross*. In the STC, it corresponds to extending a sheet so that it crosses the chord of intersection of two other sheets. In the primal, a cross corresponds to adding a hex containing three faces of the meshing front which pairwise share edges. The third operation, used for collapsing cells of size four or more, is a *blind chord formation*. In the STC it corresponds to extending a sheet so that it crosses a locally parallel sheet along a third sheet intersecting the previous two. The chord of intersection between these sheets does not start at a surface mesh face and is completely interior to the volume so it is called *blind*. In the primal, it simply corresponds to adding a hex that contains only two faces of the meshing front that share an edge.

Cell Collapsing. Given an oriented curve segment from point a to point b, we can use the basic operations to deform the segment in such a way that one of the cells neighboring the segment is collapsed.The convention we shall use is that of always collapsing to the right of the oriented segment. Recall that collapsing a cell means moving the curve so that no point of the cell is still to the right of the curve. Repeated cell collapsing will shrink a curve. Define a cell to be the segments of the points a, b and $1,...,k$ as in Figure 8 left. Collapsing the cell with points a, b, $1,...,k$ is accomplished by $k-1$ blind chord formations followed by a cross operation as in Figure 8 right. The running time of

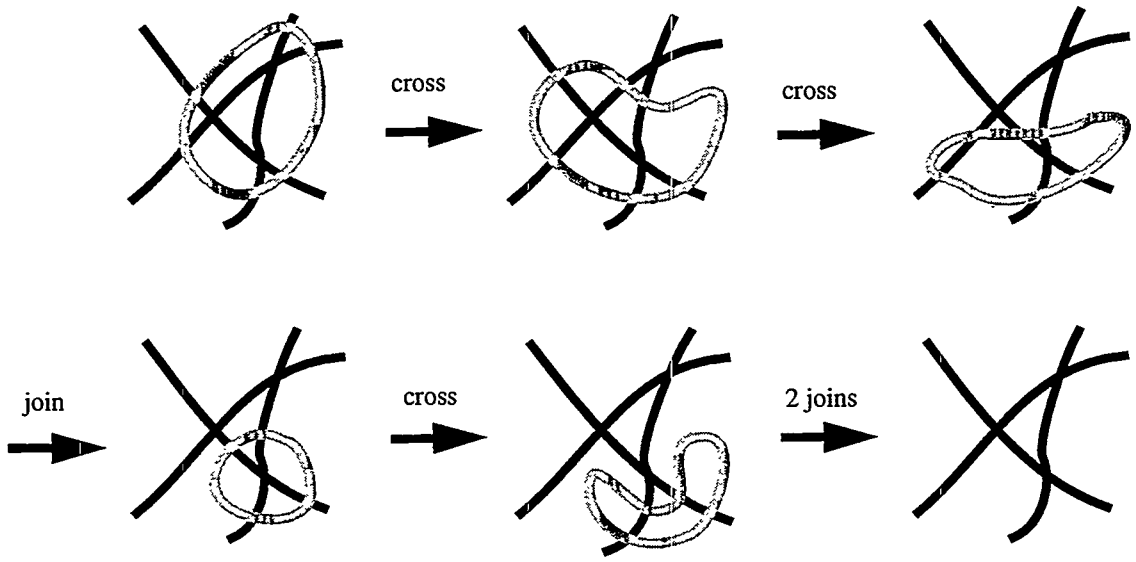


Figure 6. A simple example illustrating the curve collapsing algorithm. The grey curve is oriented clockwise.

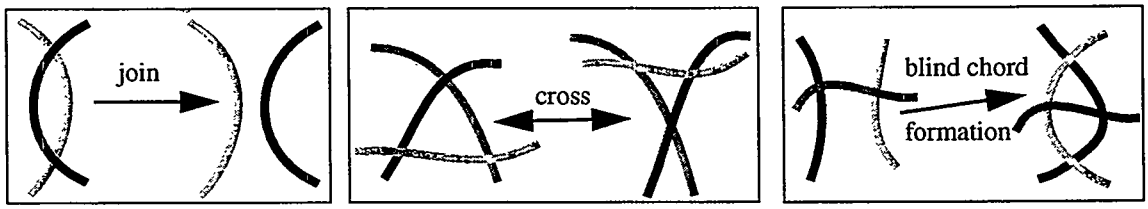


Figure 7. The three basic weaving operations used in curve contraction: join, cross and blind chord formation.

collapsing is linear in the size of the cell; each weaving operation is constant time, and collapsing takes k operations for a k+2 sided cell and 1 operation for a 2 sided cell. (1 sided cells do not arise with simple curves.)

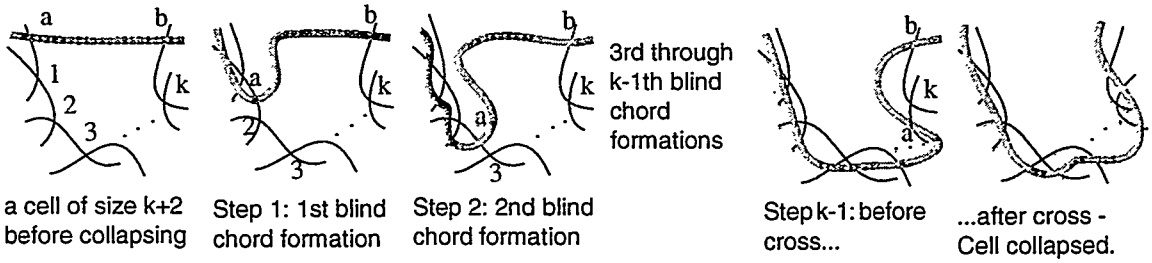


Figure 8. An example of collapsing a k+2 sided cell. Starting on the left of the figure and working to the right, we perform k-1 blind chord formations and finish with a cross operation.

Shrinking a curve. An oriented curve can be shrunk by a sequence of cell collapses. Shrinking a curve will succeed regardless of the order in which cells are collapsed. However, in practice, better meshes result if we shrink the smaller cells first. Our approach is to begin by first traversing once about the curve to find a cell C with the fewest number of sides S. We then collapse C. We back-up one cell, then continue traversing forward. Whenever we encounter a cell of size S or smaller we collapse it. If we traverse all of the way around the curve without collapsing a cell, then S is increased. Eventually all cells are collapsed and the curve shrinks to a point that does not intersect any other curve.

Recall Figure 6 provides a simple example. The reason that we back-up is that collapsing a cell reduces the size of the cells to either side; see Figure 9.

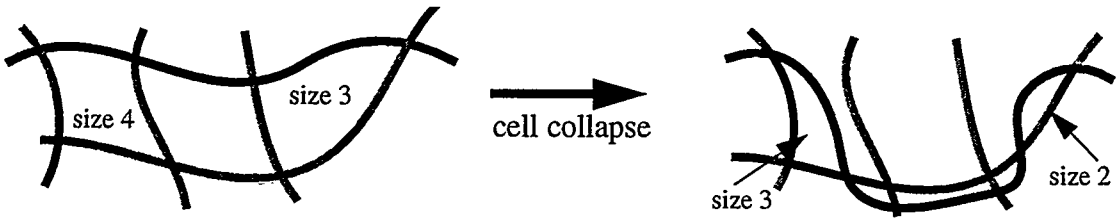


Figure 9. Collapsing a cell reduces the size of the neighboring cells by one.

To avoid forming self-intersections, we will not collapse a cell if the shrinking curve forms two or more of its sides; see Figure 10 for a simple example.

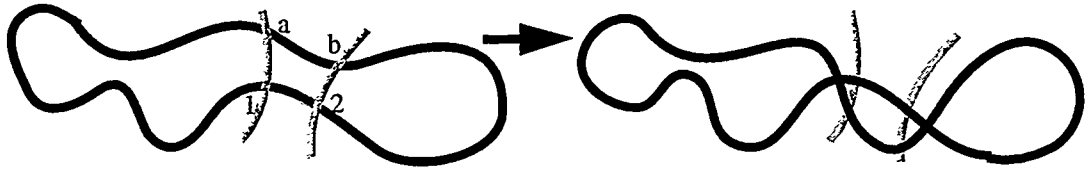


Figure 10. An example of a cell where the shrinking curve forms two sides. Collapsing the cell with points a,b,1,2 would cause a self-intersection as shown on the right, and does not lead to progress. Hence, we do not shrink such cells.

The curve shrinking algorithm provably eliminates a curve from the front. (The local rules may prevent our implementation from achieving this, however. Section 4.2 describes how we overcome this.) Collapsing a cell of size k yields one of two forms of progress. First, if k is 3 or greater, then collapsing the cell reduces the number of intersection points inside the curve by k-2. Second, if k is 2, then collapsing the cell performs a join which reduces the number of points on the curve by 2.

As implemented, the running time of shrinking a single curve is $O(n S_{max})$ where n is the number of intersection points on or to the right of the curve, and S_{max} is the maximum cell size; for each cell size, we may have to traverse once about the curve, which is always $O(n)$. Also, each point to the right of the curve is moved to the left of the curve by a weaving operation in constant time. In practice S_{max} is small, usually less than 20. A single curve could consist of all the points on the front, so n could be large, but by amortizing over all curves this is not a problem. If we shrunk cells as they were encountered regardless of size, then the algorithm would take time $O(n)$. Either way, the number of hexes created is equal to the number of points to the right of the curve, up to modification by the local rules.

Choosing which curve to collapse. Repeatedly applying the curve shrinking algorithm to each curve on the front creates a complete STC. On paper, the algorithm will succeed regardless of the order in which curves are shrunk. In practice, the best order is by increasing *weight* (defined below). The weights of all curves is computed, then the curves are shrunk in that order. The running time of the curve shrinking algorithm as implemented is $O(NS_{max} + cn)$, where c is the number of curves, n is the number of points on the front, and $N = O(cn)$ is the number of hexes generated.

Each oriented curve has two weights, corresponding to shrinking it to the *left* (from the equator towards the north pole) or *right* (from the equator towards the south pole). Let n_{rp} denote the number of points to the right of the curve, n_{lp} the number of points to the left of the curve and n_{pc} the number of points on the curve. The left-weight is defined as $w_l = n_{lp} + n_{pc}$ and the right-weight as $w_r = n_{rp} + n_{pc}$. We currently know no better way to compute these weights than the straightforward method which takes $O(cn)$ steps, where c is the number of curves and n is the number of points on the

front. Note that for a rectangular parallelepiped with a regular surface mesh, shrinking in order of our weights produces a regular hexahedral mesh. In practice, our shrinking order produces a small mesh compared to other shrinking orders.

4.2 Blending Curve Shrinking with Local Rules

The curve shrinking algorithm forms a weave without directly considering the geometry or connectivity of the hexes formed. Within Whisker Weaving there is a set of local rules[24] that prevent hexes from being formed that violate certain geometric and connectivity constraints, and also automatically perform basic weaving operations to fix certain configurations. For example, before forming a hex next to the boundary of the model, one local rule requires that there must be a certain angle between the corresponding two or three faces. We have blended these local rules into the decision structure of the curve shrinking algorithm to increase mesh quality.

In the blended algorithm, curve shrinking chooses a weaving operation, and passes it to the local rules. If the operation is consistent with the local rules, it is performed. Otherwise, it is kicked back to the curve shrinking algorithm; the current cell being collapsed is returned to its initial state and curve shrinking moves on to another cell. Also, local rules automatically remove degenerate hexes such as two hexes sharing two edges, and join chords; this can cause radical changes in the curve being shrunk. This turns the provable algorithm into a reliable heuristic that produces a better arrangement of surfaces than the pure algorithm.

There are two sets of local rules. The first set is geometry rules, which only applies to hexes being formed on the geometric boundary of the model. The second set is connectivity rules, which applies universally, and prevents two hexes from sharing three faces and the like. We first attempt to weave with both sets of rules on. If we get stuck, we turn the geometry rules off, remove the partially completed weave, and start weaving over. If we still get stuck, we turn the connectivity rules off and restart. It is usually necessary to turn the geometry rules off in order to complete the weave, but rarely necessary to turn the connectivity rules off.

A structured mesh has eight hexes attached to each node. In a Whisker Weaving mesh, one cause of poor mesh quality is *high valence nodes* - nodes with high numbers of hexes attached. We treat high valence nodes much the same way as we treat local degeneracies, by a type of local rule that attempts to avoid forming them. While choosing which cell to collapse, we check the size of the cell and the size of the polyhedra behind the cell. This corresponds to taking a node on the front and checking the number of attached faces on the front and the number of attached hexes behind the front. If the sum of these two values is above a certain threshold, we collapse that cell next. This buries the node behind the front, preventing the number of hexes attached from increasing.

5. Converting a Surface Arrangement to a Well-Defined Hex Mesh.

5.1 Removing Degeneracies by Pillowing

Mitchell's [10] existence proof enumerates the possible degeneracies that would keep an arrangement of surfaces from dualizing to a well-defined hexahedral mesh. There are three types of degeneracies possible. The first is *through-cells*: the arrangement does not resolve distinct portions of the surface mesh. For example, normally a 3-cell contains at most one 2-cell of the surface mesh. If it contains one, then the dual node of that 3-cell is actually the surface mesh node of that 2-cell. If it contains more than one, it is dual to all of those nodes. One interpretation is that in order to realize the mesh, we must first collapse the surface mesh nodes; see Figure 11 for a two-dimensional example.

The second degeneracy is *non-simplicial meets*: the intersection of two cells is non-simplicial. That is, the two cells share two or more maximal sub-cells. E.g. two 3-cells sharing two distinct 2-cells is dual to two nodes being connected by two distinct edges.

The third degeneracy is *non-distinct sup-cells*: the cells containing a cell are not distinct. A vertex should be in eight distinct 3-cells, twelve distinct 2-cells, and six distinct 1-cells (STC-edges). Similarly for STC-edges and 2-cells. E.g. if a vertex lies in only seven 3-cells, then the dual hex has only 7 distinct nodes.

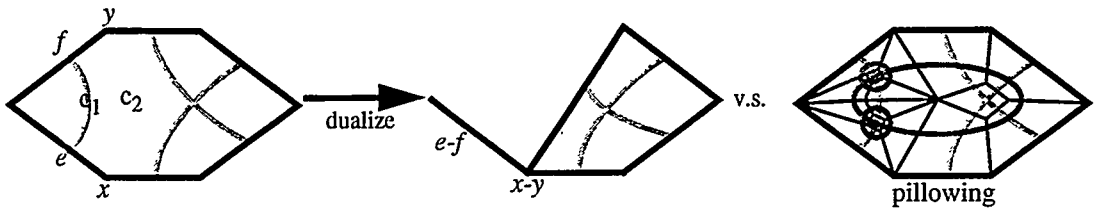


Figure 11. Left, an STC with through-cells dualizes to a mesh that conforms to a collapsed version of the bounding mesh: the 2-cell c_2 is dual to both nodes x and y, and 1-cell c_1 is dual to both edges e and f. Right, adding pillow-sheets removes this, but adds many elements.

Mitchell [10] also gives a provable construction for removing these degeneracies. However, the straight-forward implementation of that would lead to many more hexes than is necessary, and poorer quality. Our algorithm is as follows. First, we remove through-cells. We remove through-2-cells and any adjoining through-3-cells and through-1-cells by *sheet moving*; see section 5.2. Rarely we have a through-3-cell that is not part of any through-2-cell. These are removed by inserting a *pillow-sheet*, a new surface that surrounds all non-surface vertices of the through-3-cell, as in Figure 11 right.

Second, geometric checks are done on the STC-vertices connected to the surface mesh. If any vertex is dual to a hex with two or more surface-mesh quads that make a large dihedral angle, we insert a pillow-sheet that puts a boundary layer around the surface mesh. This pillowing is almost always needed if the geometry rules were turned off during weaving. We have experimented with adding a pillow-sheet surrounding just a neighborhood of the vertex, but in practice these sheets are nearly as large as the complete boundary layer and have poorer quality.

Third, we remove non-simplicial meets by pillowing the non-simplicial intersection. For each vertex, we traverse the attached 1-cells, checking that each pair has only the vertex in common. If a pair has two common vertices, we add each vertex to a list of vertices to pillow. Each vertex must be in separate pillows, since a pillow around both vertices will not remove the degeneracy. Similarly, we traverse the attached 2-cells. If any 2-cell pair has more than an edge in common, then we collect the vertices of intersection for later pillowing. We collect maximal components that are edge-connected by edges shared by both 2-cells; each components must be in a separate pillow. If any attached 3-cells have more than a 2-cell in common, we gather their intersection for later pillowing. After all these checks for the vertex, we pillow maximal sets of vertices that need not be kept separate, then proceed to the next vertex. Occasionally this fails, and we fall back on the provable algorithm of pillowing non-simplicial meets one-by-one as they are encountered.

Fourth, we remove non-distinct sup-cells. This is rarely necessary, usually just in cases where we had to pillow non-simplicial meets one by one. For each vertex, we gather its 3-cells. We pillow vertices appearing more than once in a single 3-cell.

5.2 Sheet moving for through-cells

A through-2-cell can be removed by inserting a pillow sheet surrounding all of its non-surface vertices. However, when there is a *boundary-through-3-cell*, moving the 2-cell so that it is cut by other sheets removes the degeneracy and produces fewer hexes and a better quality mesh; see Figure 12.

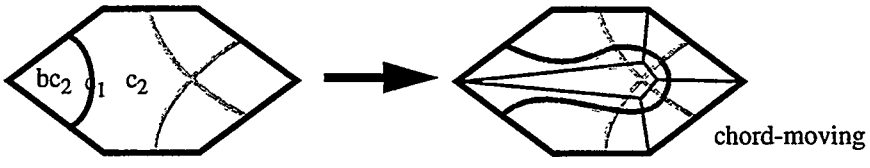


Figure 12. A two-dimensional example of sheet-moving (chord-moving) to remove through-cells: bc_2 is a boundary-through-2-cell. We move c_1 so that it goes around the vertices of c_2, where c_2 is the non-boundary through-2-cell adjacent to c_1.

Consider the 3-cells on either side of the through-2-cell. Typically, one of these cells is a *boundary-through-3-cell*: all of its vertices are either on the surface mesh or on the through-2-cell. If not, then we must resort to pillowing. Otherwise, we move the sheet containing the through-2-cell so that it surrounds the vertices of the other, non-boundary through-3-cell. In practice, there is usually a series of through-2-cells on a sheet, and we move all of these simultaneously. Occasionally a through-2-cell will pass over another. We can detect which is farther from the surface mesh, and move that one first.

6. Examples

The following small examples are good-quality, totally automatic Whisker Weaving meshes of real-world parts. While small, these parts exhibit true 3D character; to mesh these parts with a sweeping-type algorithm would require decomposing the geometry by hand, which is quite difficult in some cases.

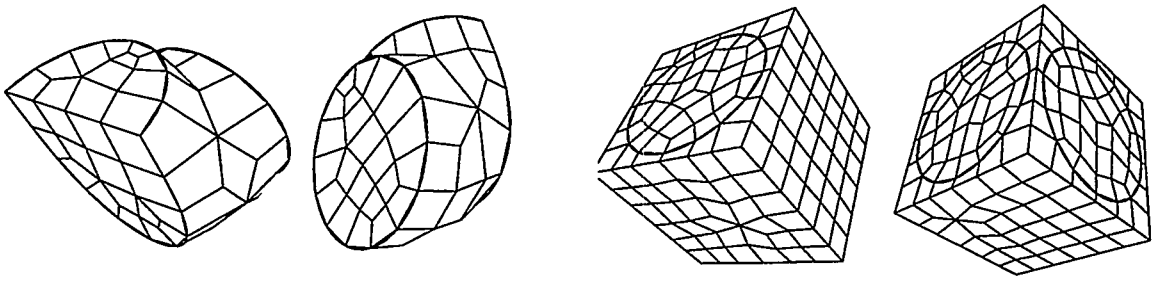


Figure 13. Left: Two views of the bent-pipe example. It has 130 hexes and the scaled jacobian ranges from 0.54 to 0.93. In the original model, the flat faces of the pipe are next to sweepable parts. Right: Two views of a complicated nugget in a large model, courtesy of Clay Fulcher. It has 499 elements. The scaled jacobian ranges from 0.12 to 0.99.

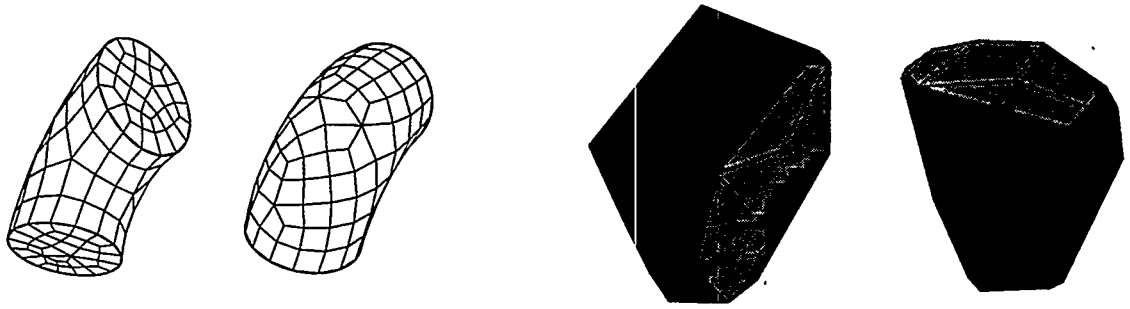


Figure 14. Left: Two views of the macaroni. It has 320 elements and the scaled jacobian runs from 0.26 to 0.95. Right: Two views of a single component of a metal grain. It has 173 elements. The scaled jacobian runs from 0.18 to 0.8.

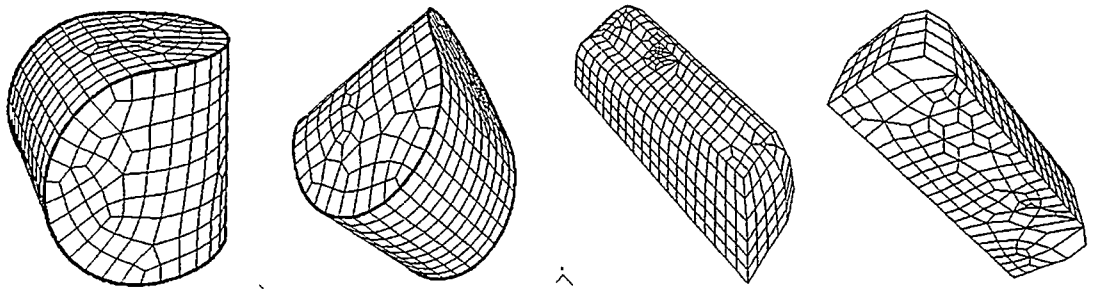


Figure 15. Left: two views of a non-sweepable geometry. It has 1320 elements and the scaled jacobian runs from 0.18 to 0.98. Right: Two views of the bathtub courtesy of Ford. It has 1041 elements and the scaled jacobian runs from 0.001 to 0.99.

The following are Whisker Weaving meshes of large, complex surface meshes and geometries. Typically 2% to 5% of the hexes have negative jacobians at nodes, making them unusable for most FEMs. Despite this, we feel these examples demonstrate proof-of-concept

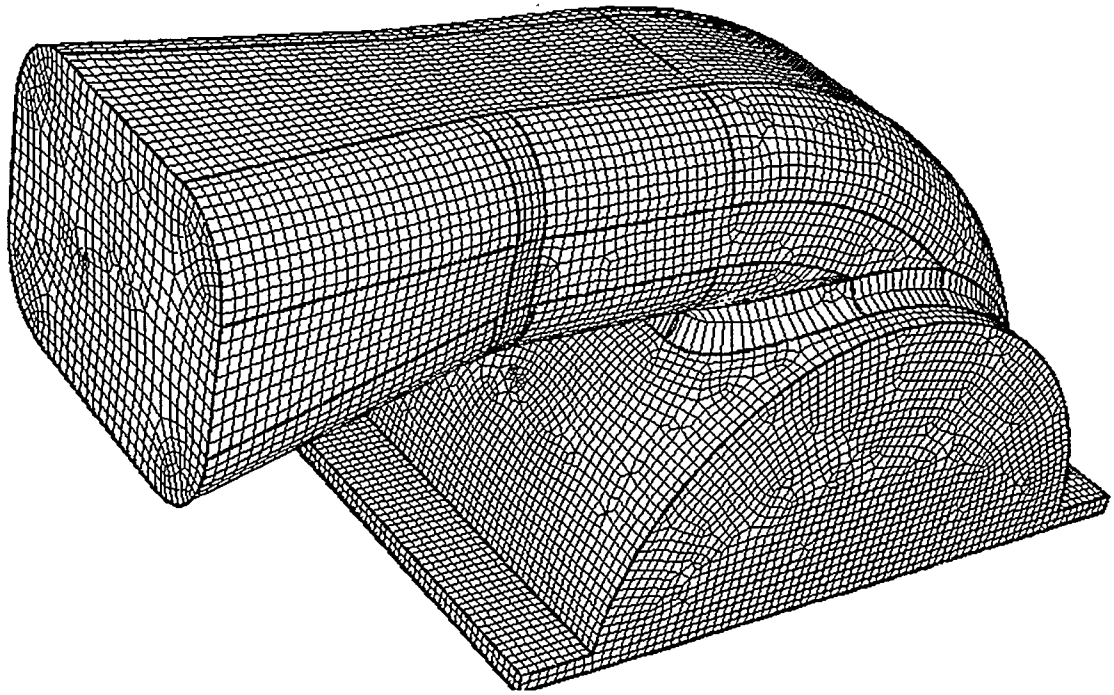


Figure 16. A view of a blower geometry complements of Rick Garcia. It has 105,999 elements and 575 negative jacobians.

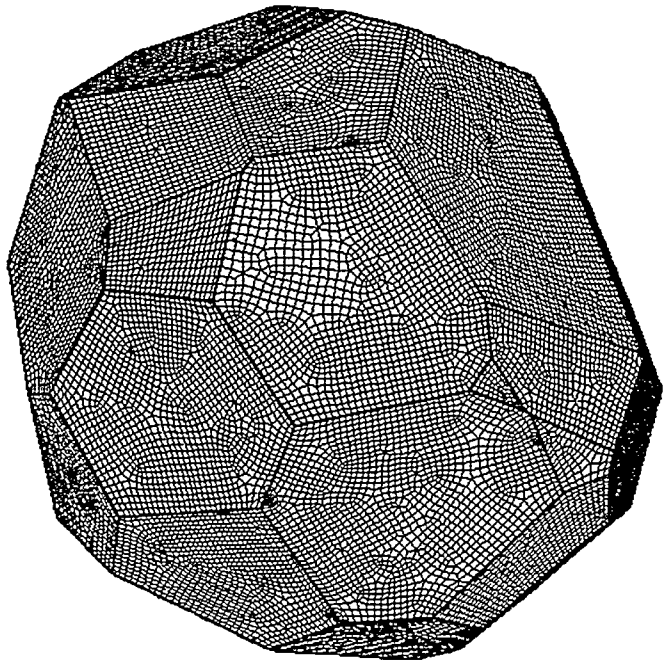


Figure 17. A single view of another component of a metal grain. It has 240,724 elements and 33 negative jacobians.

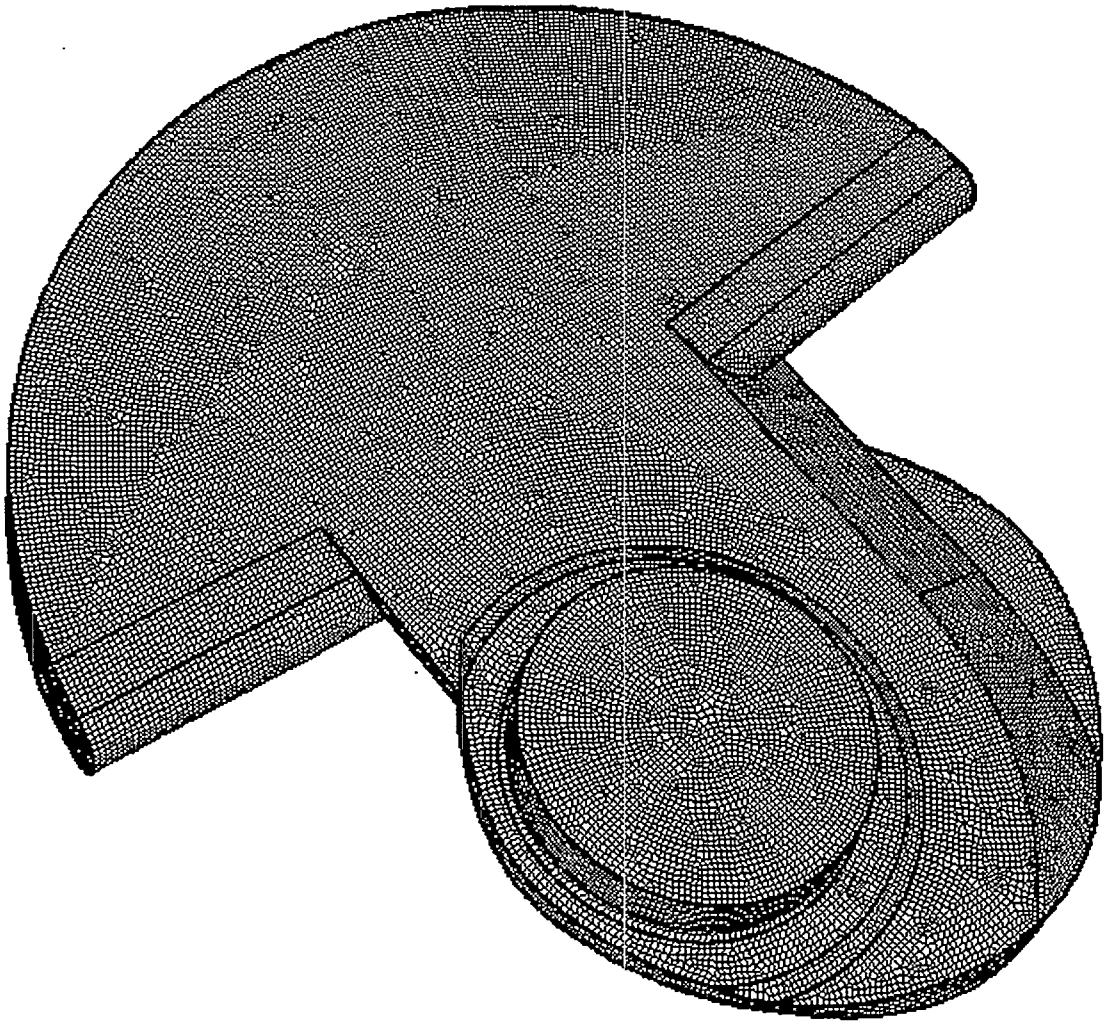


Figure 18. A view of a throw arm for a crank shaft, courtesy of Ford. It has 622,209 hexes and 15,652 negative jacobians.

7. Extensions: Contracting Self-Intersecting Curves, Non-ball Geometry.

We have algorithms for handling the self-intersection case using basic weaving operations. These are currently only worked out on paper, but appear to require not much more effort to implement than the simple curve shrinking algorithm. Progress is measured by reducing the number of self-intersections. After all self-intersections are removed, the simple curve contraction algorithm takes over. Note that the problem of simplifying self-intersecting polygonal curves has been considered in a computational setting.[9][27] However, these approaches concentrate on the geometry of the shrinking. In our case, we are interested in the topological events of the shrinking, e.g. keeping the number of events to a minimum, and, except for the initial surface-mesh loop, we have no geometry for the sheets.

We also have various algorithms on paper for reducing non-ball geometries to a small collection of meshable ball-type geometries. These algorithms have various degrees of efficiency, sophistication and provable properties. Some are geometric and some are purely topological. The common theme is to cut the handles of the geometry until only balls remain.

We are also focused on overcoming the quality problem. We wish to post-process the mesh by optimizing the position of nodes in tandem with improving connectivity regularity, as is commonly done in triangular, tetrahedral, and quadrilateral meshing. This requires research in mesh positioning, namely weighted and boundary-term smoothing. This also requires research on hex mesh connectivity swapping. Currently, poor quality concentrates around nodes with too many or too few hexes attached, where we want to swap hexes between nodes. Few swapping operations are known for hexahedral meshes, unlike for tetrahedral meshes. We plan to develop swap operations using the global information provided by the STC. The key difficulties are finding swapping operations that are local and do not affect the surface mesh, and finding a complete set of operations. We have had some preliminary success by moving dual surfaces for a special case which occurs near the bounding surface mesh; see section 5.2.

8. Conclusions

We have presented algorithms for contracting curves. These guide Whisker Weaving's basic operations and provide provable hexahedral mesh generation on paper. We have implemented a local pre-processing algorithm that perturbs the surface quad mesh so that the dual curves are simple. We have implemented the simple-curve version of curve-contraction, blended it with local geometry and connectivity rules, and implemented heuristics to improve the STC surface arrangement so that it dualizes to a hexahedral mesh. The result is that, given the freedom to perturb the surface mesh, we can reliably generate a hexahedral mesh conforming to the surface mesh. The hex meshes often have unacceptable quality in isolated regions, and our current research is focussed on overcoming this problem.

We are also currently extending our implementation to weave from a fixed surface mesh whose dual curves may be non-simple. The extension appears straight-forward. Unfortunately, surface-mesh quads where dual loops self-intersect are closely related to the formation of degenerate hexes called *knives* inside the volume. Knifes appear to be acceptable FEM elements, [4] but it would be better to avoid them. On paper knives are avoidable if the surface mesh has an even number of quads, but it is unclear if there are practical ways to avoid or remove all knives.

References

[1] Armstrong, C.G., D.J. Robinson, R.M. McKeag, T.S. Li, S.J. Bridgett, R.J. Donaghy and C.A. McGleenan, Medials for Meshing and More, *Proceedings, 4th International Meshing Roundtable*, Sandia National Laboratories, pp.277-288, October 1995.

[2] S. Canann, S. Muthukrishnan and R. Phillips, Topological Refinement Procedures for Triangular Finite Element Meshes, *Engineering with Computers*, Springer-Verlag, Vol 12, pp.243-255, December 1996.

[3] N. Calvo and S. Idelsohn, All-Hexahedral Element Meshing by Generating the Dual Mesh, *Computational Mechanics: New Trends and Applications,* CIMNE, Barcelona, Spain 1998.

[4] B. Clark and S. Benzley, Development and Evaluation of a Degenerate Seven-Node Hexahedron Finite Element, *Proceedings, 5th International Meshing Roundtable,* Pittsburgh, Pennsylvania, pp. 321-331, October 1996.

[5] D. Eppstein, Linear Complexity Hexahedral Mesh Generation, *Proceedings, 12th Annual ACM Symposium on Computational Geometry*, Philadelphia, pp. 58-67, May 1996.

[6] L. Freitag and C. Ollivier-Gooch, Tetrahedral Mesh Improvement Using Swapping and Smoothing, *International Journal for Numerical Methods in Engineering*, Wiley, Vol 40, pp. 3979-4002, 1997.

[7] P. Kinney, CleanUp: Improving Quadrilateral Finite Element Meshes, *Proceedings, 6th International Meshing Roundtable*, Park City, Utah, pp 449-461, October 1997.

[8] P. Knupp, Optimizing the 3-D Jacobian, work in progress.

[9] K. Melhorn and C. Yap, Constructive Hopf's Theorem: or How to Untangle Closed Planar Curves, *Springer Lecture Notes in Computer Science*, Vol 317, pp. 410-423,

[10] S. Mitchell, A Characterization of the Quadrilateral Meshes of a Surface Which Admit a Compatible Hexahedral Mesh of the Enclosed Volume, *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, Springer, pp. 465-476, 1996.

[11] S. Mitchell and T. Tautges, Pillowing Doublets: Refining a Mesh to Ensure that Faces Share at Most One Edge, *Proceeding, 4th International Meshing Roundtable*, Sandia National Laboratories, pp. 231-240, October 1995.

[12] S. Mitchell, The All-hex Geode Transition Template for Conforming a Diced Tetrahedral Mesh to any Diced Quad Mesh, submitted to *7th International Meshing Roundtable.*

[13] R. Meyers and P.Tuchinski, The "Hex-Tet" Hex Dominant Meshing Algorithm as Implemented in CUBIT, submitted to *7th International Meshing Roundtable.*

[14] M. Muller-Hannemann, Hexahedral Mesh Generation by Successive Dual Cycle Elimination, submitted to *7th International Meshing Roundtable.*

[15] P. Murdoch and S. Benzley, The Spatial Twist Continuum, *Proceedings, 4th International Meshing Roundtable,* Sandia National Laboratories, pp. 243-251, October 1995.

[16] M. Price an C. Armstrong, Hexahedral Mesh Generation by Medial Surface Subdivision: Part 1. Solids with Convex Edges, *International Journal for Numerical Methods in Engineering,* Vol. 38, 3335-3359, 1995.

[17] M. Price an C. Armstrong, Hexahedral Mesh Generation by Medial Surface Subdivision: Part 1. Solids with Convex Edges, *International Journal for Numerical Methods in Engineering,* Vol. 40, 111-136, 1997.

[18] R. Schneiders, R. Schindler and F. Weiler, Octree-based Generation of Hexahedral Element Meshes. *Proceedings 5th International Meshing Roundtable*, Pittsburgh, 1996.

[19] R. Schneiders, A Grid-based Algorithm for the Generation of Hexahedral Elemenet Meshes. *Engineering with Computers* vol 12, 1996, pp. 168-177.

[20] M. Shepard and M. Georges, Automatics Three-dimensional Mesh Generation by the Finite Octree Technique, *International Journal for Numerical Methods in Engineering*, Vol 32, pp. 709-749, 1991.

[21] M. Staten and S. Canann, Post Refinement Element Shape Improvement for Quadrilateral Meshes, AMD-Vol.220 *Trends in Unstructured Mesh Generation*, ASME, pp.9-16, July 1997.

[22] R. Taghavi, Automatic, Parallel and Fault Tolerant Mesh Generation from CAD, *Engineering with Computers* vol 12:178-185, 1996.

[23] T. Tautges, T. Blacker and S. Mitchell, The Whisker Weaving Algorithm: A Connectivity-based Method for Constructing All-hexahedral Finite Element Meshes, *International Journal for Numerical Methods in Engineering,* Vol. 39, pp. 3327-3349, 1996.

[24] T. Tautges, S. Mitchell, Whisker Weaving: Invalid Connectivity Resolutions and Primal Construction Algorithm, *Proceeding, 4th International Meshing Roundtable*, Sandia National Laboratories, pp. 115-127, October 1995.

[25] W. Thurston, Hexahedral decomposition of polyhedra, Posting to sci.math., 25 Oct., 1993.

[26] P. Tuchinsky, The Hex-Tet Hex Dominant Automesher: An Interim Progress Report, *Proceedings, 6th International Meshing Roundtable*, Park City, Utah, pp 183-13, October 1997.

[27] G. Vegter, Kink-Free Deformations of Polygons, *Proceedings of the 5th Annual Symposium on Computational Geometry*, pp 61-68, June 1989.

Hexahedral Mesh Generation by Successive Dual Cycle Elimination

Matthias Müller–Hannemann*
Technische Universität Berlin

Abstract

We propose a new method for constructing all-hexahedral finite element meshes. The core of our method is to build up a compatible combinatorial cell complex of hexahedra for a solid body which is topologically a ball and for which a quadrilateral surface mesh of a certain structure is prescribed. The step-wise creation of the hex complex is guided by the cycle structure of the combinatorial dual of the surface mesh. Our method transforms the graph of the surface mesh iteratively by changing the dual cycle structure until we get the surface mesh of a single hexahedron. Starting with a single hexahedron and reversing the order of the graph transformations, each transformation step can be interpreted as adding one or more hexahedra to the so far created hex complex.

Given an arbitrary solid body, we first decompose it into simpler subdomains equivalent to topological balls by adding virtual 2-manifolds. Second, we determine a compatible quadrilateral surface mesh for all created subdomains. Then, in the main part we can use the core routine to build up a hex complex for each subdomain independently. The embedding and smoothing of the combinatorial mesh(es) finishes the mesh generation process.

First results obtained for complex geometries are encouraging.

Keywords. Hexahedral mesh generation, quadrilateral surface meshing, combinatorial dual, hex complex, planar graphs, cycle elimination

1 Introduction

Global competition has led to an increasing demand to reduce the development time for new products. One step in this direction would be a more efficient computer simulation of the technical properties of prototypes for such products. The finite element method has been successfully applied by engineers for many years in simulations. But such a method needs a tool as a prerequisite which converts a CAD model into a finite element mesh model suitable for a numerical analysis.

Therefore, various algorithms for the generation of meshes have been developed, mostly decomposing surfaces into triangles and solid bodies into tetrahedra, for surveys see [BE95, BP97]. In many applications, however, quadrilateral and hexahedral meshes have numerical advantages. The potential savings gained from an all-hexahedral meshing tool compared to an analysis based on tetrahedral meshing may be enormous (with estimations in the range of 75% time and cost reductions [TC97]). On the other hand, the generation of hexahedral meshes turns out to be much more complex than for tetrahedral meshes. Recent years showed many research efforts and brought up several approaches, but up to now, hexahedral mesh generation for an arbitrary 3D solid is still a challenge.

*Technische Universität Berlin, Fachbereich Mathematik, Sekr. MA 6-1, Straße des 17. Juni 136, D 10623 Berlin, Germany, e-mail: mhannema@math.tu-berlin.de; URL: http://www.math.tu-berlin.de/~mhannema.
The author was partially supported by the special program "Efficient Algorithms for Discrete Problems and Their Applications" of the Deutsche Forschungsgemeinschaft (DFG) under grant Mo 446/2-3.

In this paper, we propose a new method for all-hexahedral mesh generation which mainly exploits combinatorial properties of such a mesh.

Geometric vs. combinatorial meshes. We distinguish between geometric and combinatorial meshes. A *geometric mesh* is a partition of some given domain into subdomains, in our context into hexahedra, i. e. regions combinatorially equivalent to cubes. In contrast, a *combinatorial hexahedral mesh*, is only a decomposition of the given domain into an abstract (cell) complex of combinatorial cubes but without an explicit embedding into space.

Thurston [Thu93] and Mitchell [Mit96] independently characterized the quadrilateral surface meshes which can be extended to hexahedral meshes. They showed that for a volume which is topologically a ball and which is equipped with an all-quadrilateral surface mesh, there exists a combinatorial hexahedral mesh without further boundary subdivision if and only if the number of quadrilaterals is even. Furthermore, Eppstein [Epp96] used this existence result and proved that a linear number of hexahedra (in the number of quadrilaterals) are sufficient in such cases. Unfortunately, all these results are non-constructive and it remains unclear whether they can be extended to constructive methods for geometric meshes. There are quite simple solids with natural looking quadrilateral surface meshes, for example the quadratic pyramid problem of Schneiders [Sch95], where only rather complicated combinatorial meshes are known, but no geometric mesh with an acceptable quality is available.

Previous work. We briefly review approaches to hexahedral mesh generation. For a more complete survey, online information and data bases on meshing literature see [Sch] and [Owe].

Most commercial systems rely on a mapping approach where the domain must be divided into simple shapes which are meshed separately. For example, *isoparametric mapping* [CO82] is a method for generating hexahedral meshes which is robust for block-type geometries, but does not work well for more complicated general volumes. Various techniques based on object feature recognition have been developed to automize the subdivision of an object into simpler parts. *Virtual decomposition* [WMBS95] separates the volume into mappable subvolumes by the creation of virtual surfaces (2-manifolds) inside the volume. Another method uses the *medial axis* of a surface and *midpoint subdivision* [PAS95, PA97]. Compatibility between adjacent subregions and mesh density is then modelled within an integer programming formulation [LMA95]. Unfortunately, solving such integer programming problems is NP-hard, even for quadrilateral surface meshes [MMW97], and such models often rely on a very restricted set of meshing primitives, so-called *templates*.

Grid-based [Sch96] and *octree-based* [SRW96] methods start with a perfect grid which is then adapted to the object's boundary by an isomorphism technique. The adaption step is difficult and often leads to badly shaped hexahedra near the boundary.

Meshing from the bounding surface inward is preferred for three reasons. First, for many analysis purposes, it is essential to have the same surface mesh on the common boundary of adjacent solids. Second, good mesh quality is often more important near the boundary of a solid than deep inside a volume. Third, dividing the domain into many smaller regions allows parallel mesh generation which may be crucial for some large-scale applications.

Plastering [Can92, BM93] is an advancing front based method which starts from a quadrilateral surface mesh. It maintains throughout the algorithm the *meshing front*, that is a set of quadrilateral faces which represent the boundary of the region(s) yet to be meshed. The plasterer selects iteratively one or more quadrilaterals from the front, attaches a new hexahedron to them, and updates the front until the volume is completely meshed.

Whisker weaving [TBM96, TM95] also meshes from a quadrilateral surface mesh inward. But in contrast to plastering it first builds the combinatorial dual of a mesh and constructs the primal mesh and its embedding only afterwards. This method is based on the so-called *spatial twist continuum (STC)*. The STC is an interpretation of the geometric dual of a hexahedral mesh as an arrangement of surfaces, the *sheets*. More precisely, the mesh dual is the cell complex induced by the intersection of the sheets. A fundamental data structure for an STC is a *sheet diagram* which represents the crossings of one sheet with other sheets. Whisker weaving starts with incomplete sheet diagrams based on the surface mesh. It seeks to complete the sheet

diagrams by a set of rules which determine the local connectivity of the mesh. The creation of invalid mesh connectivity has been observed in the plastering and whisker weaving algorithms. Heuristic strategies have been developed to resolve such invalidities [TM95].

We finally mention that some methods relax the meshing problem by allowing mixed elements, that is they try to mesh with mostly hexahedra, but include tetrahedra [TC97], or pyramids and wedges [Min97], as well.

A new approach. We suppose that a solid body is described by polygonal surface patches. For the reasons given above, the new approach presented in this paper uses an all-quadrilateral surface mesh as a starting point. But before doing the surface meshing, we first decompose a complex geometry into simpler subregions by adding internal 2-manifolds. Our method only requires that these subregions are topological balls (to avoid difficulties with holes and voids). It is not necessary, although desirable, that these regions are "almost convex." Asking for a subdivision into convex regions in the usual mathematical sense would be too strict, as otherwise there would be no subdivision into finitely many regions for concave surfaces. So, roughly speaking, we mean by *almost convex* that a region should not deviate from a convex region by too much.

The insertion of additional internal 2-manifolds creates *branchings*, i. e. edges which belong to more than two polygonal patches. All-quadrilateral surface meshing in the presence of branchings can be achieved by first solving a system of linear equations over $GF(2)$, and then using a network flows [MH97] or an advancing front based method like paving [BS91].

Because of the difficulties indicated by the pyramid example, our approach does not attempt to extend any quadrilateral surface mesh to a hexahedral volume mesh. It seems that self-intersecting cycles in the dual of the surface mesh are the main source for these difficulties. It is a disadvantage of advancing front based methods that they usually generate many such cycles. In contrast, our network flow based mesher [MH97] tends to produce very regular meshes with only few self-intersecting cycles. To get rid of the remaining self-intersecting cycles we use two strategies. First, we use additional heuristics to avoid them in the surface meshing. Second, we introduce a little trick to modify a mesh such that such cycles do not disappear, but can be coped with. This trick will be explained in Section 4. The important feature of this trick is that one can still use any quadrilateral surface mesher (provided it can handle branchings consistently and mesh the virtual internal surfaces without an explicit geometric embedding).

After this preparation, the core of our method is to build up a compatible combinatorial cell complex of hexahedra for a solid body which is topologically a ball and for which a quadrilateral surface mesh is prescribed.

The step-wise creation of the hex complex is guided by the cycle structure of the combinatorial dual of the surface mesh. Our method transforms the graph of the surface mesh iteratively by changing the dual cycle structure until we get the surface mesh of a single hexahedron. Starting with a single hexahedron and reversing the order of the graph transformations, each transformation step can be interpreted as adding one or more hexahedra to the so far created hex complex. The embedding and smoothing of the combinatorial mesh(es) finishes the mesh generation process.

Overview. We first introduce some basic terminology in Section 2. In the main part of this paper, in Section 3, we present our new approach for the meshing of topological balls. The general case will be dealt with in Section 4. In Section 5 we summarize the main features of our approach and give directions for future work.

2 Basic Terminology

We need some basic graph theory [NC88]. *Drawing* a graph in a given space means representing nodes as points and edges as curves. A graph can be *embedded* into some space if it can be drawn such that no two edges intersect except at a common node. A graph is *planar* if it has an embedding in the plane, or equivalently, an embedding on a sphere in three-dimensional space. The curves representing the edges of a

Figure 1: Hexahedron with curved quadrilateral facets (left), a planar surface graph embedding and its combinatorial dual (right).

planar, embedded graph partition the plane or the sphere, respectively, into connected components, called *faces*. For a planar graph G, the *(geometric) dual* G^* is constructed as follows. A node v_i^* is placed in each face F_i of G; corresponding to each (primal) edge e of G we draw a dual edge e^* which crosses e but no other edge of G and joins the nodes v_i^* which lie in the faces F_i adjoining e.

Let G be a graph, not necessarily simple, that is, it can have loops and parallel edges. Such a graph is *connected* if there is a path between any two distinct nodes of G. A graph G with at least 4 edges is 3-*connected* if it is simple and cannot be disconnected by removing 1 or 2 nodes from G. The reason for this version of the definitions (here, we follow Ziegler [Zie98]) is that it is invariant under planar duality. That is, if we have a planar embedding of a graph G and construct its dual graph G^*, then G is 3-connected if and only if G^* is 3-connected.

A *(convex) polyhedron* is the intersection of finitely many halfspaces in some $\mathbb{R}^d$, and it is a *polytope* if it is bounded. The famous theorem of Steinitz relates planar graphs and polytopes.

Theorem 2.1 (Steinitz' theorem) *G is the graph of a 3-dimensional polytope if and only if it is simple, planar and 3-connected.*

Hex complex. The bodies we want to mesh have more general, curved surfaces. Therefore we use the term *(geometric) cell* to mean a bounded region in 3-dimensional space, bounded by a finite number of 2-manifolds. In the following, cells of different dimension will appear: 0-dimensional cells, i. e. single points, called *vertices*; 1-dimensional cells, i. e. segments of curves between two vertices, the *edges*, and 2-dimensional cells in the form of *quadrilateral facets*, i. e. smooth 2-manifolds bounded by a cycle of four distinct edges. A *hexahedron* is a 3-dimensional cell which is a combinatorial cube. It is bounded by 6 distinct quadrilateral facets, 12 distinct edges, and 8 distinct vertices. The quadrilaterals pairwise share edges as depicted in Fig. 1.

Ideally, hexahedra are polyhedra, but in this abstract setting we do not require that the edges are straight line segments and that the quadrilateral facets are planar.

A geometric cell complex of hexahedra, called *hex complex* for short, is a finite, non-empty collection $\mathcal{C}$ of distinct (openly disjoint) hexahedra and all their lower dimensional cells such that the intersection of any two members of $\mathcal{C}$ is a cell of each or the empty set.

A hex complex is *compatible* with the quadrilateral surface mesh of a body if this surface mesh is the union of all quadrilateral facets which belong to exactly one hexahedron of the complex. A surface compatible hex complex is a *non-degenerated geometric mesh* if all of its hexahedra have a positive volume and the dihedral angle between any two adjacent quadrilaterals of a hexahedron is strictly smaller than 180 degrees.

Two hexahedra are *neighbored* if they share a quadrilateral. By definition, a hexahedron of a geometric cell complex has at most one neighbored hexahedron for each quadrilateral face (*unique neighbor property*).

If cells of a hex complex are not embedded but abstract entities, we regard a cell as composed by its lower-dimensional cells. For an abstract cell C, we define the *cell lattice* as the the set of all lower-dimensional cells including the cell C itself and the empty cell, partially ordered by inclusion. Abstract cells are *combinatorially equivalent* if their cell lattices, are isomorphic.

For a hex complex given by abstract cells, an *abstract hex complex*, we have to demand explicitly the unique neighbor property (which is no longer implied by definition) to avoid degeneracies. Hence, an abstract hex complex is *non-degenerated* if it has the unique neighbor property. An abstract hex complex yields a surface

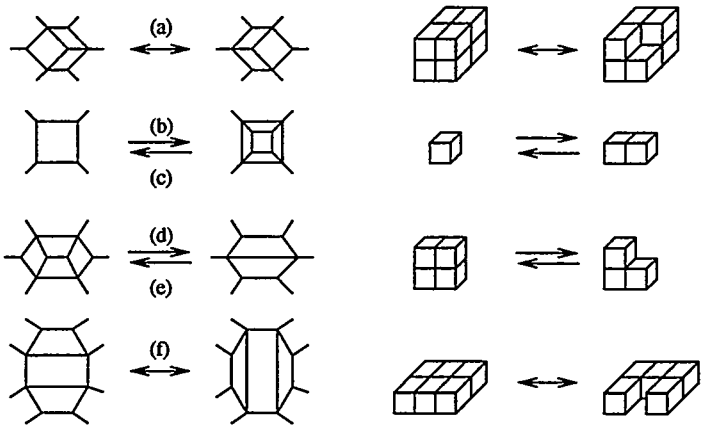


Figure 2: Local graph transformations (a) – (f) (left side) and examples of their application (right side).

compatible combinatorial mesh if each quadrilateral is contained in exactly one hexahedron, if it belongs to the surface, and in exactly two hexahedra, otherwise.

3 Meshing Topological Balls

Throughout this section we restrict our discussion to the case that we are given a solid body which is topologically a ball, implying that the surface is topologically a sphere. We further assume that an all-quadrilateral surface mesh has been determined for this body. Even more, we assume that the surface mesh is a simple, planar and 3-connected graph (which is reasonable in view of Steinitz' theorem). Hence, although the geometry of the surfaces we consider is usually more general than that of a polytope, the combinatorial structure is not.The key idea of our approach is to keep these properties invariant.

Shelling and graph transformations. Suppose that a hex complex is already known for some body. Then we could try to *shell* this complex, that is to decompose the complex by taking away a hexahedron one after another. Shellability of cell complexes has been widely studied in a more general, but slightly different way [Zie98]. What we are looking for is a *topology preserving shelling* of a hex complex. By that we mean that the remaining hex complex is still topologically a ball after the deletion of a hexahedron. More precisely, such a shelling maintains the invariant that the surface mesh is a simple, planar and 3-connected graph.

Consider the six "local" operations (a) - (f) shown in Fig. 2 on subgraphs of the surface graph. They represent the transformation of the surface graph for all possible ways to take away a single hexahedron from a hex complex without changing the topology. (To be applicable, a subgraph must have at least the edges shown in Fig. 2 to ensure that each node has minimum degree 3 after the transformation.) Observe that each operation can be reversed and then interpreted as adding a hexahedron to a hex complex.

The basic properties of these transformations are summarized in the next lemma.

Lemma 3.1 *Let G be a simple, planar and 3-connected graph whose faces are all quadrilaterals. Then any application of one of the six operations in Fig. 2 preserves the following invariants for the resulting graph G': G' remains simple, planar, 3-connected, all its faces are quadrilaterals, and the parity of the number of quadrilaterals leaves unchanged.*

In other words, topology preserving shelling of a hex complex can be seen as applying a series of graph operations on a planar graph. At first glance, this does not help too much as we usually do not know a hex complex compatible to the surface we want to mesh, and so cannot determine which operation we should apply and in what order. So we are faced with the problem of "shelling an unknown complex." This seems to

be intractable for general surface meshes, but we will characterize below important classes of surface meshes where this concept is useful.

Perfect cycle elimination schemes. Let G be the graph of a quadrilateral mesh and G^* its combinatorial dual. We say that two adjacent dual edges are *opposite to each other* if and only if they correspond to opposite sides of a quadrilateral, i. e. if they are not neighbored in the cyclic adjacency list of their common node. Hence, the four adjacent edges to each dual node can be partitioned into two pairs of opposite edges.

The dual graph $G^* = (V^*, E^*)$ can be decomposed in a canonical way into a collection of edge-disjoint cycles, say into $C_1, \ldots, C_k$, by putting a pair of adjacent edges e_1^*, e_2^* into the same cycle if they are opposite to each other. In other words, for each quadrilateral the edges which are dual to opposite sides are contained in the same cycle. Observe that by transitivity two edges may belong to the same cycle even if they are neighbored in the cyclic adjacency list of some node. Hence, these dual cycles can be non-simple or *self-intersecting.*

Note that this set of dual cycles is well-defined and unique (in the sense that two cycles are equivalent if they have the same set of edges) and can be easily determined in linear time. We call this set the *canonical dual cycles* of G^*, and by a dual cycle we will henceforth always mean a canonical dual cycle.

Let us suppose that the edges of a dual cycle are ordered within a cyclic list such that two edges are consecutive if and only if they are adjacent and opposite to each other with respect to their common node.

The cyclic order of the edge list induces an orientation of the cycle. With respect to such an orientation, a simple dual cycle separates the dual vertices $V^* \setminus V(C^*)$ into vertices $V^*_{C,\ell}$ on the "left hand side" of C and vertices $V^*_{C,r}$ on the "right hand side."

Given a planar graph G, its dual G^*, and a dual cycle C of G^*, the elimination of C transforms G to G' and G^* to G'^* in the following way. The new graph G' is obtained from G by contracting each primal edge corresponding to a dual edge contained in C, and removing parallel edges afterwards. The graph G'^* is then defined as the combinatorial dual of G'. An equivalent way to describe the elimination of a dual cycle is to remove all edges of C from G^*, and to replace for each node v^* of C the two remaining incident edges, say (u^*, v^*) and (w^*, v^*), by the new dual edge (u^*, w^*), and finally remove all vertices of C. In this second definition, dualization of G'^* gives the new primal graph G'. A *feasible elimination* of a dual cycle C from G^* requires that

1. C is a simple cycle,
2. at least one of the two subgraphs of G^* induced by $V^*_{C,\ell}$ and $V^*_{C,r}$, respectively, is connected, and
3. G'^* is 3-connected.

Note that 3-connectivity can be checked in linear time [HT73]. So we can test in linear time whether a cycle can be eliminated in a feasible way or not. The concept of a feasible elimination is motivated by the following lemma which connects such eliminations to shelling and graph operations.

Lemma 3.2 *Let G be the graph of a quadrilateral mesh and G^* be its combinatorial dual. Let C be a dual cycle which can be eliminated in a feasible way from G^*. If $V^*_{C,\ell}$ ($V^*_{C,r}$) is connected then there is a sequence of exactly $|V^*_{C,\ell}|$ ($|V^*_{C,r}|$) graph operations which transforms G to G'.*

We only note that this sequence can be found efficiently by a topology preserving shelling of the planar primal graph induced by the union of the quadrilaterals contained in $V^*_{C,\ell}$.

To get an intuitive idea of the use of cycle eliminations for the meshing we change the viewpoint. Suppose that G' is the surface mesh of some hex complex. Then we can reverse the order of the graph transformations and construct a hex complex for the graph G by adding one hexahedron to each quadrilateral corresponding to the set $V^*_{C,\ell}$. So elimination of a cycle amounts to the addition of a complete layer of hexahedra formed by this set, that is in STC terminology to the addition of a complete sheet. The three conditions for feasible eliminations are necessary to give the desired result.

A *perfect cycle elimination scheme* of a dual graph G^* of a quadrilateral surface mesh is an order $C_1, C_2, \ldots, C_k$ of its k canonical cycles such that the first $k-3$ cycles with respect to this order can be eliminated one after another in a feasible way, and such that the remaining 3 cycles form the dual surface graph of a single hexahedron. In particular, this means that the last 3 cycles are simple and cross pairwise exactly twice (two cycles *cross* if they share a common node).

The core algorithm. Whenever we know a perfect cycle elimination scheme for some graph G^*, we can iteratively apply Lemma 3.2 in reversed elimination order to give an explicit construction of a hex complex compatible to the prescribed surface mesh. This yields an algorithm for combinatorial meshes of topological balls which runs in two phases: In the first phase we determine a cycle elimination scheme, and in the second phase we build up a hex complex by adding sheets in reversed order of the elimination. Three interesting questions arise:

1. Which surface meshes have a perfect elimination scheme?
2. Which elimination order should be preferred?
3. What can be done if one gets stuck at some point, that is, no (further) cycle can be eliminated?

We give (partial) answers to these questions in the following. By definition, existence of a perfect cycle elimination scheme requires that all dual cycles are simple. Simplicity of a dual cycle implies even length, and all cycles being simple implies an even number of quadrilaterals. The latter is sufficient for the existence of a combinatorial mesh, but not for the existence of a perfect elimination scheme in general. To see a small counter-example, consider a 3-connected planar dual graph consisting of three canonical cycles which cross each other pairwise exactly four times.

On the positive side, there are important classes which always have a perfect elimination scheme. Zonotopes are special polytopes which can be defined in many equivalent ways [Zie98], for example as the image of a higher-dimensional cube under an affine projection, or as the Minkowski sum of a set of line segments. Zonotopes where all surface facets are quadrilaterals constitute a class of polytopes for which every cycle order leads to a perfect elimination scheme (the "zones" of a zonotope correspond to the dual cycles). Certainly, it is sufficient that the surface graph of some mesh is combinatorially equivalent to that of a zonotope. The important combinatorial property of such graphs is that the cycles cross each other pairwise exactly twice.

This can be used in the following extension. We say that a cycle C_1 is a *parallel neighbor* of another cycle C_2 in the graph G^* if their node sets are disjoint but for each node v^* of C_1 there is a node w^* of C_2 such that the edge (v^*, w^*) belongs to G^*. A helpful observation is that we can feasibly eliminate a cycle if it has two parallel neighbor cycles. If the dual cycles are partitioned into equivalence classes of parallel neighbors, then a graph has a perfect elimination scheme if representing cycles of these equivalence classes cross each other pairwise exactly twice.

Cycle selection. Now we sketch our answer to the second question. In principle, every perfect cycle elimination scheme allows us to build up a valid combinatorial mesh. In general, different elimination orders, however, lead to meshes of a different structure and size. Also observe that two different geometric meshes can have the same combinatorial mesh, see Fig. 3. Hence, it is important to choose the order of the cycles in the elimination scheme carefully. To guide the elimination we use additional information about the geometry of the surface mesh. For that purpose, we determine for each dual edge the dihedral angle between the two quadrilateral faces which correspond to its endpoints. This, in turn, gives us a an initial classification for each primal edge as "sharp" or "plane" edges. A primal edge of the surface graph is a *sharp edge* if the dihedral angle is significantly smaller than 180 degrees. For a simple dual cycle C, we use the term *neighboring primal cycles* to mean the two connected primal cycles induced by the union of all primal edges of the quadrilaterals corresponding to nodes of C which do not cross C. We assign a *side elimination weight* to each dual cycle according to the number of sharp edges of the neighboring primal cycles, normalized by the cycle length (number of edges). The weights can be used to define a preference order for dual cycles. A *first rule* is that

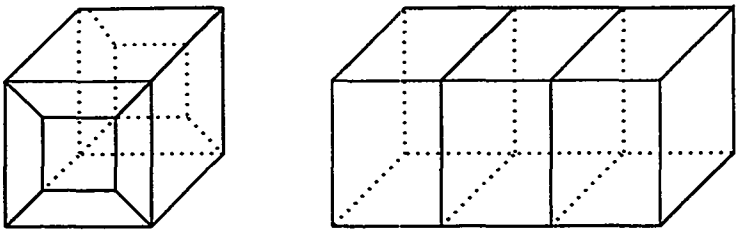


Figure 3: Examples of two solids with "obvious" decompositions into five and three hexahedra. Note that the surface meshes are combinatorially equivalent. Therefore, additional geometric information is necessary to yield the appropriate decompositions. For the left instance, a double cycle elimination is appropriate.

a cycle should be preferred in the selection if it has a higher weight, i. e. if it has a higher quotient of sharp edges to the total number of cycle edges than some other cycle. We also keep track of the side for which the elimination weight has more sharp edges, as this side will be used as the enclosed set of quadrilaterals on which the construction phase adds hexahedra.

Moreover, we use a weight counting the number of sharp primal edges corresponding to edges of the dual cycle. A *second rule* says that one should eliminate a cycle only if this second weight is positive. The intuition behind this rule is that one should not eliminate a cycle which lies in plane, as this may lead to a bad quality of the geometric mesh.

After each cycle elimination we update this classification for all primal edges which are affected by the elimination. To be precise, if two primal edges become identified by contraction of a quadrilateral, the resulting edge is classified as a sharp edge if at least one of the edges was sharp before the identification. Hence, the weights of a dual cycle will change after eliminations.

Double cycle elimination. Suppose that some dual cycle C_m has two parallel neighbors, one on the right and one on the left side. Then it can certainly be eliminated in a feasible way. If, in addition, all edges of the corresponding left and right primal neighboring cycles are classified as sharp edges, the cycle C_m has a high preference to be selected for elimination. However, in such a situation it seems to be better to eliminate both the left and right neighboring parallel cycles simultaneously. Such a *double elimination* corresponds to a series of graph transformations from Fig. 2. In this case, the set of enclosed quadrilaterals is just the dual cycle C_m in the middle. The elimination can be interpreted as removing a torus of hexahedra, see the left example of Fig. 3.

Adding new cycles. We now sketch a strategy to resolve the case that no perfect cycle elimination exists but all cycles are simple. The case of self-intersecting cycles will be discussed in the next section. Recall that all graph transformations of Fig. 2 can be reversed. Hence, through a series of such transformations we cannot only eliminate a dual cycle but also insert a new one. This insight is helpful in those cases where the elimination process gets stuck. If no cycle can be eliminated in a feasible way, the idea is to add one or more new cycles in such a way to the current graph that at least one of the old cycles can be feasibly eliminated afterwards. To ensure that this strategy finally leads to a perfect elimination scheme, a new cycle is inserted such that it crosses each old cycle exactly twice. We omit the details of such an insertion.

Embedding and smoothing. Up to this point we have only discussed how to find a combinatorial mesh. Embedding of a combinatorial hex complex into the prescribed surface such that all hexahedra are well-shaped is a non-trivial task. It is not even clear which conditions are sufficient for a "nice" embedding. However, we have been quite successful with a strikingly simple strategy. Namely, to find an initial layout we consider a hex complex as a graph, fix the surface nodes according to the surface mesh, and for all other nodes the node position is determined as a convex combination (the barycenter) of the position of its neighbors. Mesh smoothing can be used afterwards to improve the node positions.

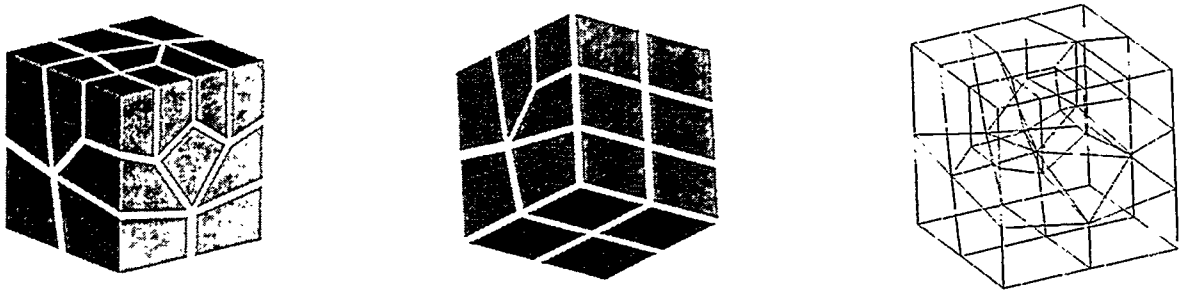


Figure 4: The "doublefold" example. The hex complex consists of 17 hexahedra.

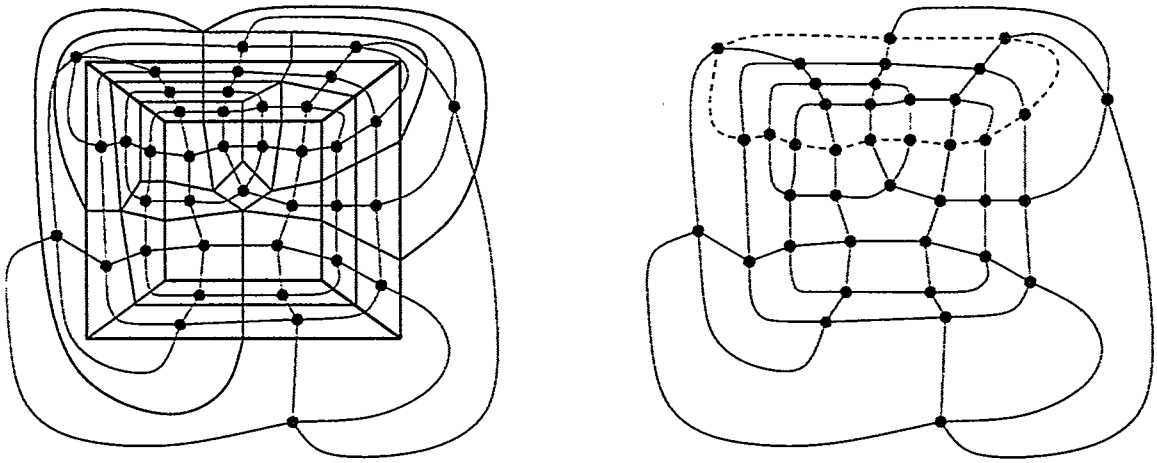


Figure 5: The primal and dual graph of the surface mesh of the "double fold" example, dual nodes are drawn as solid circles (left); the first dual cycle of a perfect elimination scheme is dashed, it encloses 9 dual nodes (right).

Example: the "double fold" problem revisited. We use the example of the so-called "double fold" problem to illustrate how our algorithm works. The same example has been used to study the whisker weaving algorithm [TBM96]. The geometry of this example is simply a cube, but the surface mesh is quite irregular (several nodes of degree five and three), see Fig. 4. The Figs. 5 to 8 show the successive elimination of dual cycles from the original surface mesh to the mesh corresponding to a single hexahedron, and the construction phase. The first cycle to be eliminated encloses 9 dual nodes, the second 3, the third 2, and the fifth and sixth only one node. Hence, in the reversed order, the series of hexahedra to add to the first one is 1,1,2,3,9.

4 A meshing scheme for arbitrary domains

Self-intersections of dual cycles are a major source of difficulties within the mesh generation process. As announced earlier, we use a little trick to circumvent these problems. After decomposing the domain of our solid body into topological balls, we use the quadrilateral surface mesher with only half the required mesh density. Then we subdivide each quadrilateral into four new ones (by halving all edges) to meet the required density. This replacement duplicates all dual cycles. In particular, all quadrilaterals where self-intersections occur appear in pairs, see the left part of Fig. 9. Even more, all self-intersections appear not only in pairs but the pairs are also locally close together in a well-defined relative position. So it is possible to change the

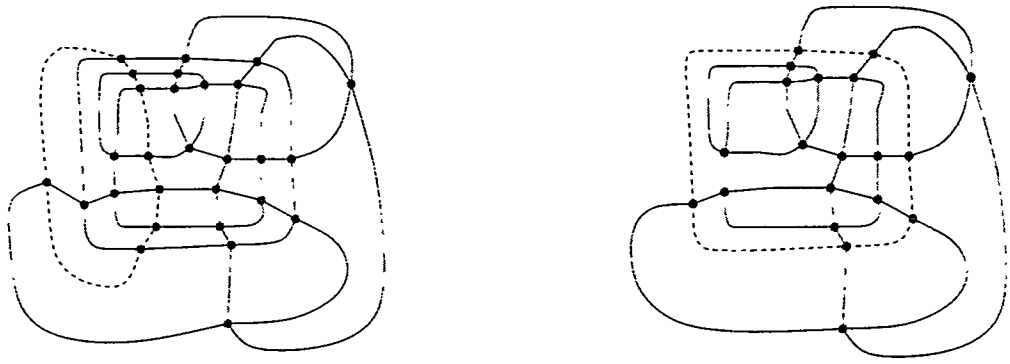


Figure 6: The dual graph after the first and second elimination. The cycle to be eliminated next is again dashed. The dashed cycle on the left side encloses 3 dual nodes, on the right side 2 dual nodes (on the unbounded side of this drawing!).

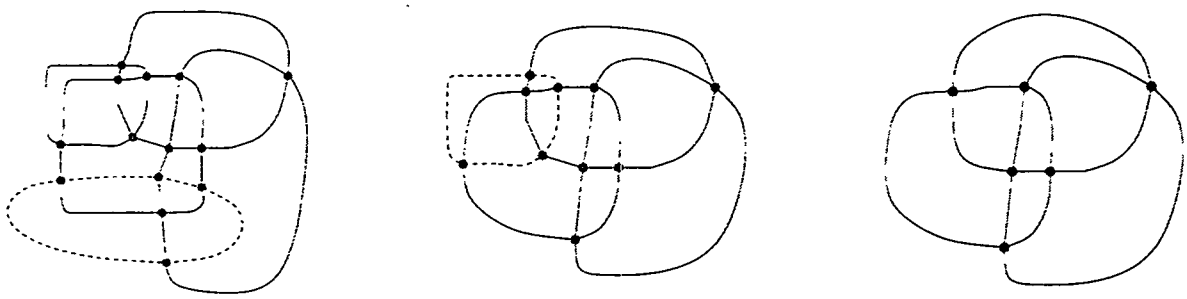


Figure 7: The dual graph after the third, forth and fifth elimination. The right figure shows final configuration of the remaining three cycles.

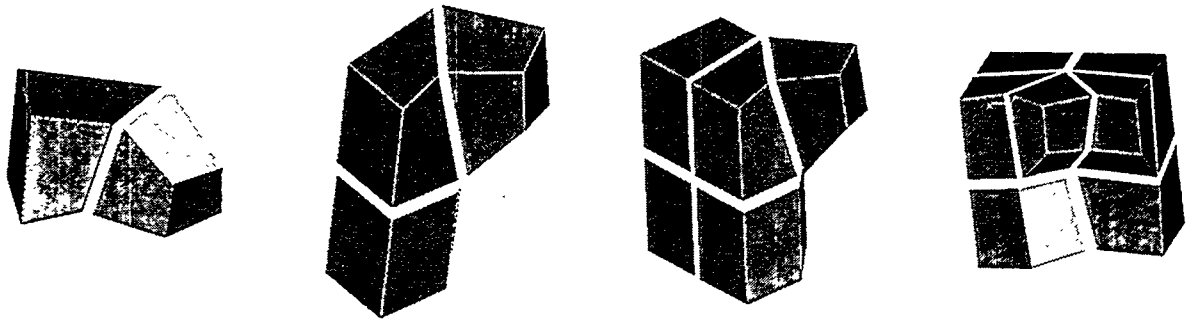


Figure 8: Construction of the hex complex for the "doublefold" example. The figures show from left to right the intermediate steps where hexahedra are added in reversed order of the elimination scheme.

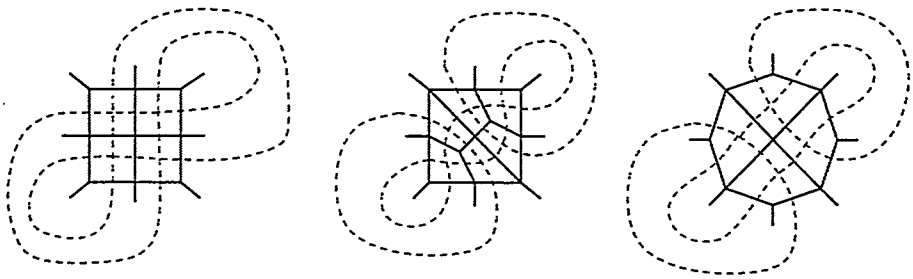


Figure 9: Getting rid of self-intersecting dual cycles: dual cycles are duplicated (left), the surface mesh is locally transformed (middle), and finally the application of two graph operations, first (a) and then (d) of Fig. 2 remove the self-intersection (right).

surface mesh locally at all such places, see the middle part of Fig. 9, without changing the structure of other cycles. This does not resolve the self-intersections immediately, but allows the following. We can mesh a layer of hexahedra on the complete surface graph (an additional sheet) - the combinatorial structure of the surface graph remains unchanged - and then use two graph transformations for each pair of self-intersections, first operation (a) and then (d) of Fig. 2. Thus, self-intersections can always be removed with the same sequence of graph operations. The extra layer of hexahedra is used to avoid degenerated surface quadrilaterals in the geometric embedding. Hence, we can directly use the resulting surface graph and avoid an additional layer and the two graph operations if the surface mesh can be embedded such that the rightmost pattern of Fig. 9 is non-degenerated.

Now we can state our algorithm for arbitrary domains. It involves the following steps:

1. Decompose the whole domain into subdomains which are topologically balls and "almost convex."
2. Quadrangulate the surface mesh with half of the required density.
3. Replace each quadrangle by four new ones.
4. Do for each subdomain
   (a) Eliminate self-intersecting dual cycles.
   (b) Search for a perfect cycle elimination scheme.
   (c) Build up a hex complex.
5. Embed hex complex and perform mesh smoothing.

Example: Model of a shaft. To illustrate first experimental results with our algorithm we use the CAD model of a shaft with a flange shown in Fig. 10. Several features make this model useful for testing purposes: it contains 8 holes, a cylindrical inlet in the lower part (not visible in the hidden surface description of Fig. 10), and concave surface patches. The input model is axis-symmetric, and so is our output – although our algorithm has no tools incorporated to detect and to exploit symmetry. The complex input model has first been decomposed into topological balls, see the left part of Fig. 10. Then our quadrilateral meshing algorithm [MH97] has been run according to some specified uniform mesh density. The resulting surface mesh has the nice property to be free of self-intersecting dual cycles for all subregions. For all subregions, our algorithm was successful to find a perfect cycle elimination scheme. Even more, the embedding of the hex complexes has led also to geometric meshes of a very good quality. Figs. 11, 12, 13 and 14 show the resulting meshes for different parts of the model. In total, the hexahedral mesh for the complete model, depicted in the right part of Fig. 10, consists of 7488 hexahedra.

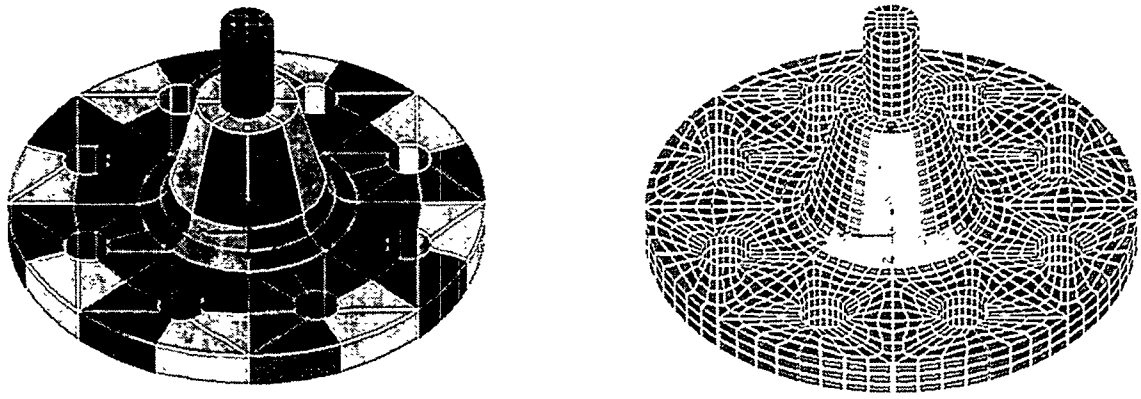


Figure 10: The model of a shaft with a flange which has been decomposed into simpler subdomains where different colors correspond to the subdomains (left), and the hexahedral mesh constructed by our algorithm (right).

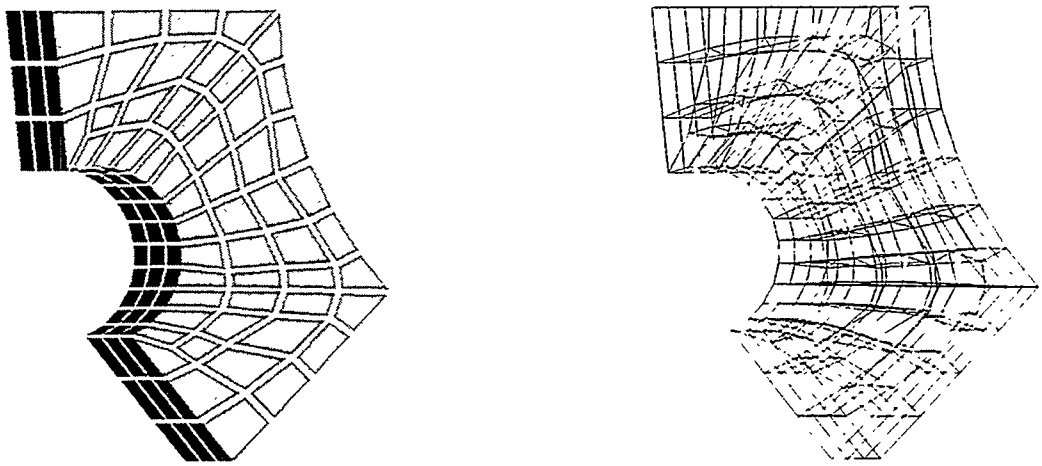


Figure 11: Hexahedral mesh for a part of the model in Fig. 10.

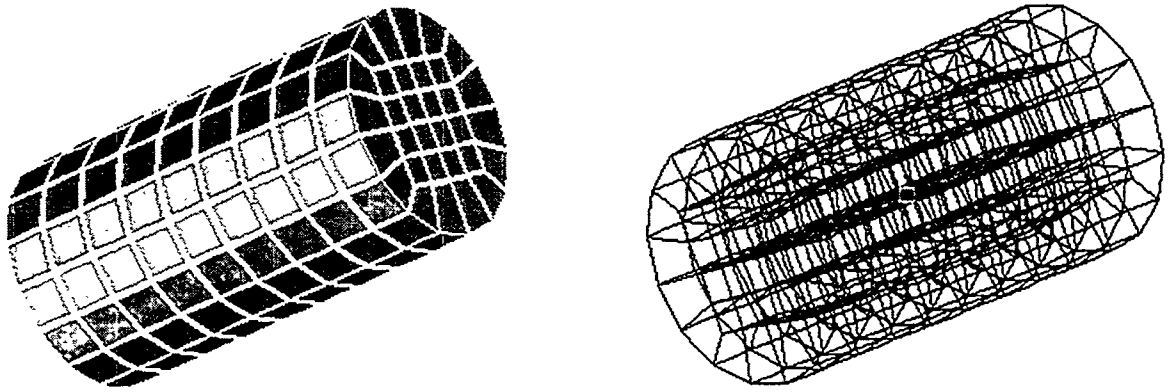


Figure 12: Hexahedral mesh for a cylinder (part of the model in Fig. 10). The cycle elimination scheme first selects parallel cycles corresponding to eight layers, and uses than a double elimination to remove the 16 hexahedra forming a torus on the last layer.

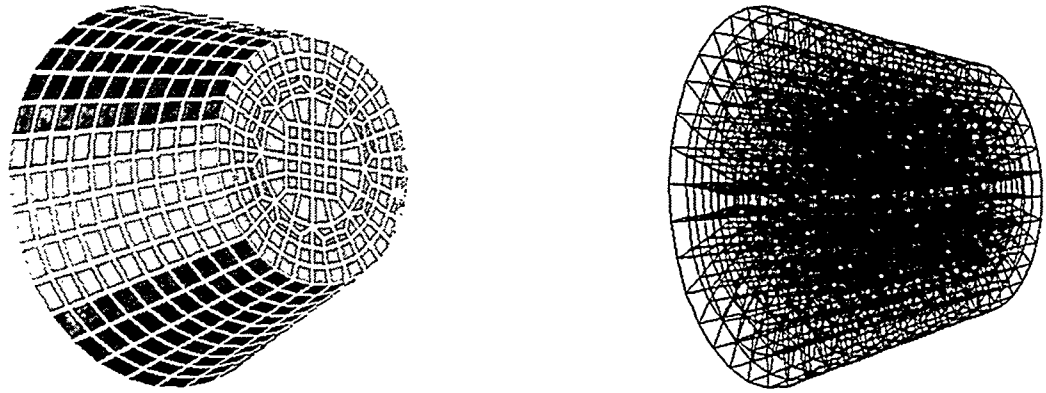


Figure 13: Hexahedral mesh for a truncated circular cone (also part of the model in Fig. 10). The meshing is non-trivial because the base cycles of the cone have different surface meshes.

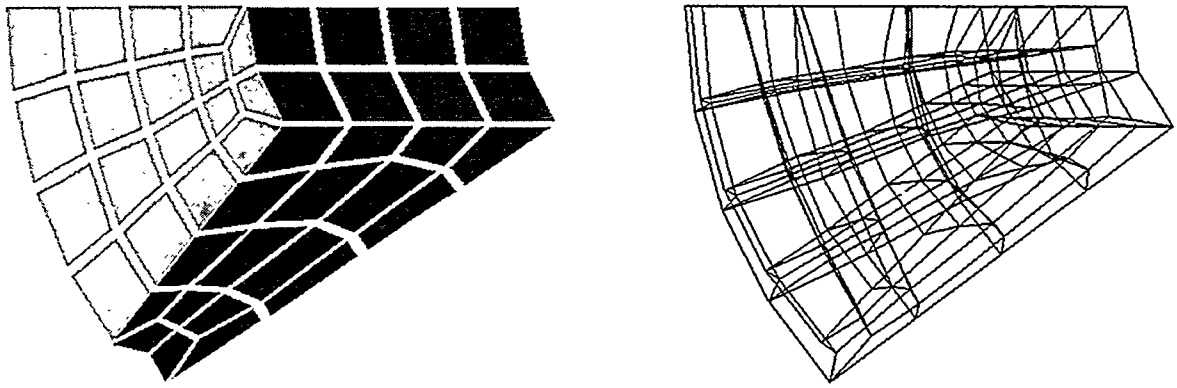


Figure 14: Hexahedral mesh for another part of the model in Fig. 10.

5 Conclusions

We summarize the main features of our approach.

- The mesh generation process is separated into a combinatorial part where an abstract complex of hexahedra is constructed from a surface mesh and a geometric embedding part where the exact positions of the mesh nodes are determined.
- Hexahedral mesh quality benefits from our surface mesh generator which yields very regular meshes (in contrast to paving or advancing front based methods).
- Successive elimination of dual cycles corresponds to the meshing of complete layers of hexahedra (sheets) in each step. Cycle selection can use global geometric information to find good elimination orders. Our strategy avoids all kind of combinatorial degeneracies rigorously. Problems with knives or wedges as known from the whisker weaving or plastering algorithms do not occur.
- Only the first part of our meshing procedure needs global access to the data. Precisely, this includes preprocessing of the input data, the decomposition into simpler subdomains, and solving a single system of linear equations over $GF(2)$ to ensure the parity conditions for the surface meshing. The very core of our algorithm, and thus the computational most expensive part, can be performed in parallel for each subdomain.

Currently, we have only implemented a first prototype for our algorithm and our experiences are still limited. First results are encouraging, but more testing has to be done to refine several parameters of our algorithm, for example the cycle selection rules. We would also like to improve our strategies for cycle insertion for those cases where no perfect elimination scheme exist.

To guarantee that we can resolve all self-intersections we used the somewhat dirty trick to duplicate all dual cycles. We think that self-intersections should be avoided in the preprocessing part as far as possible and future work should concentrate on how to minimize the number of self-intersections of dual cycles in surface meshing algorithms in combination with rules for the subdivision into topological balls.

Acknowledgment. The author wishes to thank Karsten Weihe for many fruitful discussions and G. Krause for providing us with the finite element preprocessor ISAGEN (which we used for our illustrations) and instances from the automobile industry.

References

[BE95] M. Bern and D. Eppstein, *Mesh generation and optimal triangulation*, Computing in Euclidean Geometry, 2nd Edition (D.-Z. Du and F. Hwang, eds.), World Scientific, Singapore, 1995, pp. 47–123.

[BM93] T. D. Blacker and R. J. Meyers, *Seams and wedges in plastering: A 3d hexahedral mesh generation algorithm*, Engineering with Computers **9** (1993), 83–93.

[BP97] M. Bern and P. Plassmann, *Mesh generation*, Handbook of Computational Geometry (J. Sack and J. Urrutia, eds.), Elsevier Science, 1997, to appear.

[BS91] T. D. Blacker and M. B. Stephenson, *Paving: A new approach to automated quadrilateral mesh generation*, Int. J. Numer. Methods in Eng. **32** (1991), 811–847.

[Can92] S. A. Canann, *Plastering: A new approach to automated, 3d hexahedral mesh generation*, Am Inst. Aeronautics and Astronautics (1992).

[CO82] W. A. Cook and W. R. Oakes, *Mapping methods for generating three-dimensional meshes*, Computers in Mechanical Engineering, CIME Research Supplement (1982), 67–72.

[Epp96] D. Eppstein, *Linear complexity hexahedral mesh generation*, Proceedings of the 12th Annual ACM Symposium on Computational Geometry, Philadelphia, ACM, 1996, pp. 58–67.

[HT73] J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. of Computing **2** (1973), 135–158.

[LMA95] T. S. Li, R. M. McKeag, and C.G. Armstrong, *Hexahedral meshing using midpoint subdivision and integer programming*, Computer Methods in Applied Mechanics and Engineering **124** (1995), 171–193.

[MH97] M. Müller-Hannemann, *High quality quadrilateral surface meshing without template restrictions: A new approach based on network flow techniques*, Proceedings of the 6th International Meshing Roundtable, Park City, Utah, Sandia National Laboratories, Albuquerque, USA, 1997, pp. 293–307.

[Min97] W. Min, *Generating hexahedron-dominant mesh based on shrinking-mapping method*, Proceedings of the 6th International Meshing Roundtable, Park City, Utah, Sandia National Laboratories, Albuquerque, USA, 1997, pp. 171–182.

[Mit96] S. A. Mitchell, *A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume*, Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS'96), Lecture Notes in Computer Science 1046, Springer, 1996, pp. 465–476.

[MMW97] R. H. Möhring, M. Müller-Hannemann, and K. Weihe, *Mesh refinement via bidirected flows: Modeling, complexity, and computational results*, Journal of the ACM **44** (1997), 395–426.

[NC88] T. Nishizeki and N. Chiba, *Planar Graphs: Theory and Algorithms*, Annals of Discrete Mathematics, vol. 32, North–Holland, 1988.

[Owe] S. Owen, *Meshing research corner*, http://www.andrew.cmu.edu/user/sowen/mesh.html.

[PA97] M. A. Price and C. G. Armstrong, *Hexahedral mesh generation by medial surface subdivision: Part II, solids with flat and concave edges*, Int. J. Numer. Methods in Eng. **40** (1997), 111–136.

[PAS95] M. A. Price, C. G. Armstrong, and M. A. Sabin, *Hexahedral mesh generation by medial surface subdivision: Part I, solids with convex edges*, Int. J. Numer. Methods in Eng. **38** (1995), 3335–3359.

[Sch] *Information on finite element mesh generation*, available online at http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html.

[Sch95] R. Schneiders, *Open problem*, available online at http://www-users.informatik.rwth-aachen.de/~roberts/open.html, 1995.

[Sch96] R. Schneiders, *A grid-based algorithm for the generation of hexahedral element meshes*, Engineering with Computers **12** (1996), 168–177.

[SRW96] R. Schneiders, R.Schindler, and F. Weiler, *Octree-based generation of hexahedral element meshes*, Proceedings of the 5th International Meshing Roundtable, Sandia National Laboratories, Albuquerque, USA, 1996.

[TBM96] T. J. Tautges, T. Blacker, and S. A. Mitchell, *The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes*, Int. J. Numer. Methods in Eng. **39** (1996), 3327–3349.

[TC97] P. M. Tuchinsky and B. W. Clark, *The "HexTet" hex-dominant automesher: An interim progress report*, Proceedings of the 6th International Meshing Roundtable, Park City, Utah, Sandia National Laboratories, Albuquerque, USA, 1997, pp. 183–193.

[Thu93] W. Thurston, *Hexahedral decomposition of polyhedra*, Posting to sci.math., 25 Oct., 1993, available online at http://www.ics.uci.edu/~eppstein/gina/Thurston-hexahedra.html.

[TM95] T. J. Tautges and S. A. Mitchell, *Whisker weaving: Invalid connectivity resolution and primal construction algorithm*, Proceedings of the 4th International Meshing Roundtable, Sandia National Laboratories, Albuquerque, USA, 1995, pp. 115–127.

[WMBS95] D. R. White, L. Mingwu, S. E. Benzley, and G. D. Sjaardema, *Automated hexahedral mesh generation by virtual decomposition*, Proceedings of the 4th International Meshing Roundtable, Sandia National Laboratories, Albuquerque, USA, 1995, pp. 165–176.

[Zie98] G. M. Ziegler, *Lectures on polytopes*, revised ed., Graduate Texts in Mathematics, vol. 152, Springer-Verlag, New York, 1998.

Surface Meshing

MESH GENERATION METHODS OVER PLANE AND CURVED SURFACES

Alejandro Díaz-Morcillo, Agustín Bernal-Ros and Luis Nuño
Departamento de Comunicaciones, Universidad Politécnica de Valencia
Camino de Vera s/n, 46071 Valencia (Spain)
Tel.: +34 96 387 78 22, Fax: +34 96 387 73 09
E-mail: adiaz@dcom.upv.es

Abstract. In this paper, a set of mesh generation methods over plane and curved surfaces is presented. The surfaces can be plane, just defined by their boundary (as cylinders or cones), quadrics, surfaces of revolution or a combination of these by means of a multiblock definition. The basic method for plane surfaces and those defined by their boundary is based on a advancing front / algebraic combined technique for two dimensions. Quadrics and surfaces of revolution are transformed from the real space into a parametric plane, in which the previous basic method can be applied and, later, the inverse transformation is done to obtain the final mesh.

Keywords. Surface mesh generation, Advancing front, Transfinite interpolation, Multiblock methods, Structured and unstructured meshes, Ruled, quadric and revolution surfaces

Introduction

The mesh generation is a main part of the finite element analysis, and the solution obtained with the finite element method depends strongly on the quality of the mesh. A good meshing method must generate a correct mesh, i.e., boundary conforming, without holes, without free edges and without intersecting elements, and containing as few badly shaped elements as possible.

There are a large variety of meshing methods in two and three dimensions. In a simple classification we can refer advancing front [1-3], Delaunay-Voronoï [4-7], and quadtree/octree [8,9] methods, which are able to generate unstructured meshes of arbitrary geometries. Structured meshes can be obtained by means of transfinite mapping methods [10,11] or solving a partial differential equation system [12]. A descriptive review of these methods can be found in [13]. Concerning the surface mesh generation, a general method for surface patches can be found in [14], and more specific methods in [15].

The aim of this paper is the description of a basic method for meshing plane surfaces or those defined by their boundary, and two methods for quadrics and surfaces of revolution, respectively, which use the method for plane surfaces to generate a mesh in a parametric plane and, later, transform it in the final mesh. Although the set of methods for plane or curved surfaces presented is of wide application, we will use examples of domains corresponding to electromagnetic problems in closed boundaries, concretely in transmission lines (2D) and electromagnetic cavities (3D).

Concerning the type of mesh, these methods can generate triangular or quadrilateral elements, and, therefore, it is possible to generate triangular meshes and quadrilateral/triangular hybrid meshes. Triangular meshes are created as a result of the breakdown of quadrilateral elements in two triangles. Two criteria can be applied for this splitting: the shortest diagonal or the biggest angle criterion. Better results are obtained with the last one and, consequently, it has been implemented here.

Plane Surfaces

The mesh generation method for plane surfaces is based on an advancing front / algebraic combined technique and is the base for the meshing methods for more complex surfaces as quadrics or surfaces of revolution. In its basic form, it is able to mesh quadrilateral domains, i.e., areas defined by four lines. Nevertheless, a modification of the method has been developed in order to allow the meshing of triangular areas. In that case, the triangle is transformed in a quadrilateral by "cutting" a corner of the triangle, that is, generating an edge and a triangular element in one corner, and obtaining the mesh of the rest of the domain by means of the method for quadrilateral areas.

Fig. 1 shows the evolution of the basic method for a simple example. Starting from one side of the quadrilateral, the mesh progresses until the domain is completely discretized. In each step of the method, if the new quadrilateral can be meshed with an algebraic method, i.e., the number of divisions or edges in its opposite sides is the same, a transfinite mapping method [10] is used. If it is impossible, a set of inner nodes and edges joining these nodes are generated in order to split the quadrilateral in two smaller quadrilaterals, one of them with, at least, one side with just one division. Fig.2 shows an example of this splitting. The front advances in the direction where the two opposite sides have a bigger difference in number of divisions. In order to assure the convergence in the number of divisions of the two lateral opposite sides of the quadrilateral along the algorithm evolution, the excess of divisions in a lateral side must be distributed between the two new quadrilaterals (*c1* and *c2*). The method applies recursively with each one of these quadrilaterals. Each branch of the recursive algorithm finishes when the quadrilateral has two opposite sides with one division (in that case, only edges and elements are generated, as Fig. 3 shows), or when the quadrilateral can not be broken down in two quadrilaterals. This last situation occurs when two sides sharing a vertex have just one division. These quadrilaterals require a specific management according to the number of divisions in their two other sides. The mesh quality becomes worse when the number of divisions of these sides increases. Fortunately, it is very unusual to reach situations with a great difference in the number of divisions. Fig. 4 shows the meshing patterns for this type of quadrilaterals.

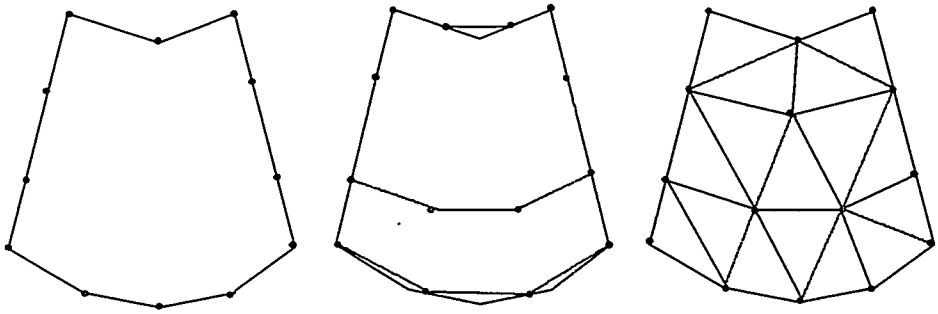


Figure 1. Mesh generation method for quadrilateral areas

The number of inner nodes generated at each step is a weighted average of the number of nodes in each side in the direction of the front. This weighting depends on the distance (in number of lateral divisions) from the front to the upper and bottom side.

In order to transfer the irregularity of the boundary to the mesh and obtain boundary conforming meshes of curved quadrilateral domains, the inner nodes are created by means of a linear interpolator based on a transfinite mapping [10]. As we have commented before, this method requires the same number of divisions in the opposite sides of the quadrilateral. Fig. 1 describes how to achieve this condition in a single case. It is necessary to perform a temporary discretization of the two opposite sides by modifying their number of divisions in order to have the same number of nodes on them. These temporary nodes are generated by means of an uniform interpolation along both sides. Fig. 1.b shows the inner nodes and the mesh progress at that step.

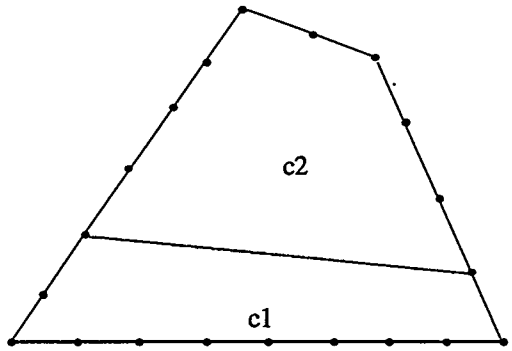


Figure 2. Splitting of a generic quadrilateral

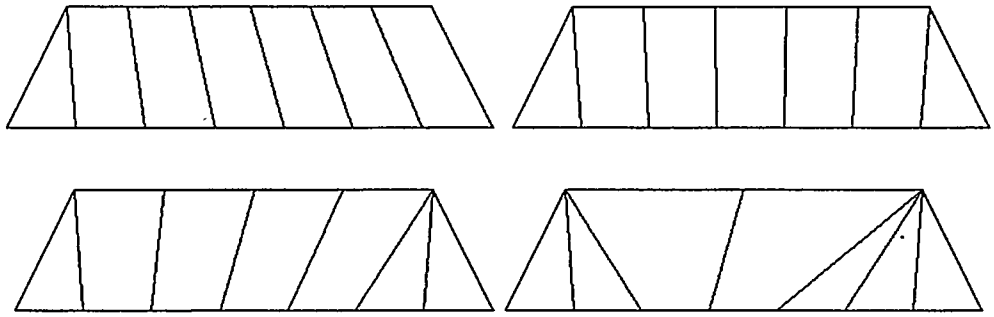


Figure 3. Meshing patterns for quadrilaterals with two opposite sides with one division

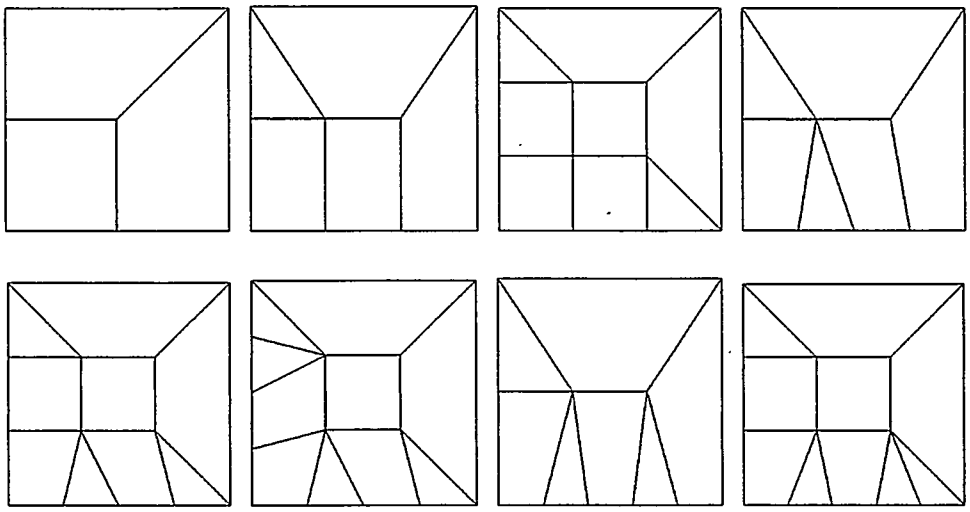


Figure 4. Meshing patterns for quadrilateral with two consecutive sides with one division

Next, the framework of the algorithm is described:

1. Triangular or quadrilateral area?
   Triangular area → go to step 2.
   Quadrilateral area → go to step 3.
2. Generate a triangular element in a corner of the area. The remaining area is considered as a quadrilateral area.
3. Has the quadrilateral the same number of edges in opposite sides?
   Yes → apply the algebraic mesh generator. End.
   Not → continue.
4. Has the quadrilateral two opposite sides with one division?
   Yes → generate edges and elements using the meshing patterns of figure 3. End of this branch.
   Not → continue.
5. Has the quadrilateral two consecutive sides with one division?
   Yes → generate the mesh using the meshing patterns of figure 4. End of this branch.
   Not → continue.
6. Generate the front line.
7. Calculate the number of nodes over the front line
8. Generate the nodes and edges over the front line.
9. Go to step 3 for quadrilateral 1
10. Go to step 3 for quadrilateral 2

With the mesh generators for quadrilateral and triangular areas and a multiblock approach [16] it is possible to mesh arbitrary domains. Fig. 5 shows an example of the definition of complex domains starting from quadrilateral areas, and the resulting mesh. The figure represents a pair of cables enclosed in a hollow metallic cylinder.

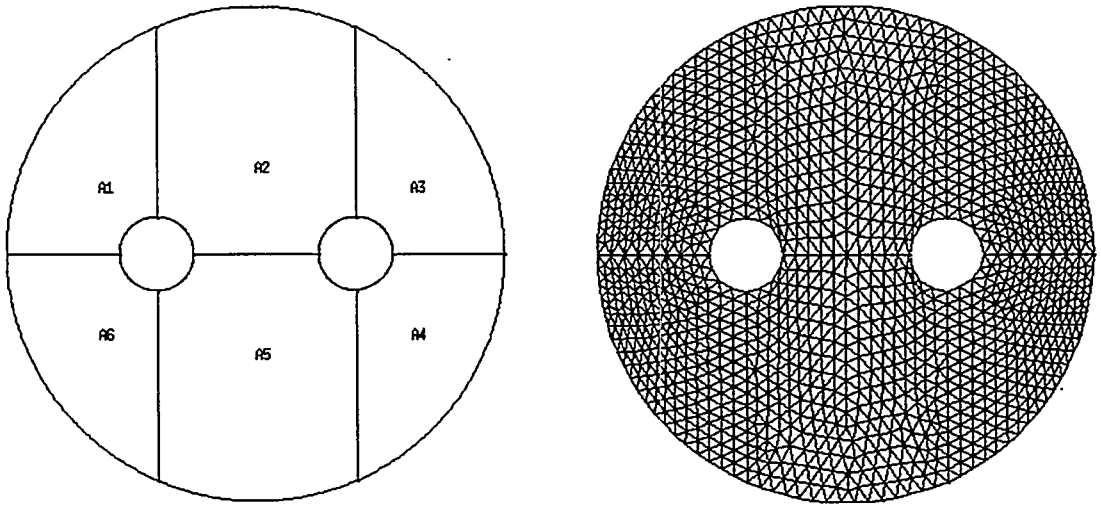


Figure 5. Multiblock mesh generation of complex domains

Surfaces Defined by their Boundary

We consider a surface defined by their boundary if the surface is not plane, and the unique information given when we create the surface is the boundary, not a equation. Although different surfaces could fit to that boundary lines, in order to mesh these surfaces, we look for one that adjust to the boundary lines, following the shape given by the boundary.

The mesh of this kind of surfaces can be obtained using the same method which has been described previously for plane surfaces, that uses the interpolator to create the inner nodes. When the lines are curved the interpolator follows the curvature of the lines. If two lines of the boundary are straight lines then the discretization is the ruled surface

that fits to the other lines. Among these surfaces we can mention well known geometric shapes like cones, cylinders or other ruled surfaces. Figure 6 shows the mesh for a surface of this last type.

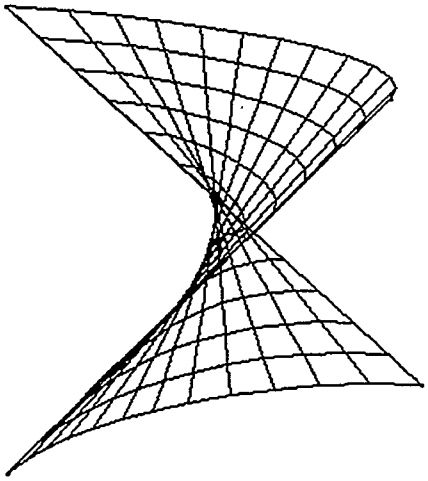


Figure 6. Meshing of a ruled surface

Note that when we talk about the lines of the boundary, we are thinking in boundaries of only three or four lines. In the cases of more sides some of them must be linked and considered like only one line Fig. 7 shows an example of compound line.

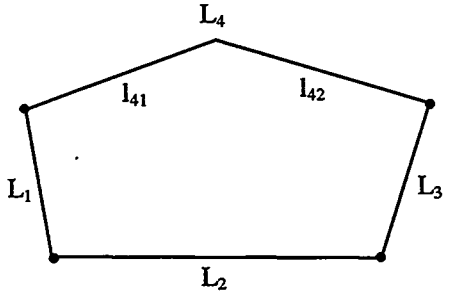


Figure 7. Quadrilateral area with single and compound lines

Quadric Surfaces

The number of coefficients of the general equation for quadrics can be reduced for those surfaces with some symmetries. Also the equation can be expressed in a simplified way if the surface is placed in the origin of co-ordinates and he axis of the surface is located over one of the co-ordinate axis. We are going to use these properties to simplify the creation of the surface with the program.

Among the reduced quadrics we choose for our study those that could be useful in electromagnetic problems. They are the ellipsoid, the parabolic form, the hyperbolic form and the sphere like a special case of ellipsoid, but very important. Immediately afterwards the equations for these surfaces, centred and oriented, are shown.

Sphere : $x^2 + y^2 + z^2 = R^2$

Ellipsoid : $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$

Parabolic form : $\frac{x^2}{a^2}+\frac{y^2}{b^2}=c\cdot z$

Hyperbolic form : $\frac{x^2}{a^2}-\frac{y^2}{b^2}-\frac{z^2}{c^2}=1$

One of the objectives of the method is to have a total control over the discretization, in the sense that we want to be able to do the mesh with a higher density in some zones, but lower in others. The method is focused to define in the first step the lines, which can be discretized independently ones from others. These lines are taken as the base to build the surfaces using them to create the boundary of the mentioned surfaces. If no more specifications are indicated, we have a surface defined by boundary, but when the lines are curves that fit to a quadric, and we really want that quadric, the information must be completed with the necessary data in each case. For the sphere it is necessary only the centre, and for the others quadrics, besides the centre, we need the axis and the points in the axis that cut the quadric (a, b, c). With all these data the surface is completely defined in the space.

Because of the complexity of meshing directly the surface with variation in three dimensions, and in order to profit the method developed for plane surfaces, we look for a solution passing through the transformation of the problem in three dimensions to another in two dimensions.

The first step in this transformation is to move the surface to the origin of co-ordinates and rotate it to orient the axis of the surface over the axis of co-ordinates, because the surface could be in any point of the space.

The next step is to transform the surface from the real space into a parametric plane whose co-ordinates are angular in such a way that the surface defined with three cartesian co-ordinates, now is defined with two angular co-ordinates. The transformations used are detailed immediately afterwards.

Sphere
$$\begin{cases} R = x^2 + y^2 + z^2 \\ u = a\cos\left(z/R\right) \\ v = a\cos\left(x/R\cdot sin(u)\right) \\ x = R\cdot sin(u)\cdot\cos(v) \\ y = R\cdot sin(u)\cdot sin(v) \\ z = R\cdot\cos(u) \end{cases}$$

Parabolic form
$$\begin{cases} u = \sqrt{z} \\ v = a\cos\left(x/a\cdot u\right) \\ x = a\cdot u\cdot\cos(v) \\ y = b\cdot u\cdot sin(v) \\ z = u^2 \end{cases}$$

Ellipsoid
$$\begin{cases} u = a\cos\left(z/c\right) \\ v = a\cos\left(x/a\cdot sin(u)\right) \\ x = a\cdot sin(u)\cdot\cos(v) \\ y = b\cdot sin(u)\cdot sin(v) \\ z = c\cdot\cos(u) \end{cases}$$

Hyperbolic form
$$\begin{cases} u = a\cosh\left(z/c\right) \\ v = a\cosh\left(x/a\cdot sinh(u)\right) \\ x = a\cdot sinh(u)\cdot\cos(v) \\ y = b\cdot sinh(u)\cdot sin(v) \\ z = c\cdot\cosh(u) \end{cases}$$

The transformation process begins with the lines of the boundary that have been discretized previously, and more exactly with the nodes created over the lines, given in cartesian co-ordinates. As result of the information about the surface, the co-ordinates of the nodes can be transformed to angular co-ordinates. After that, the surface is placed in a parametric plane with two co-ordinates (u, v).

Over the surface reduced to the parametric plane the method for plane surfaces described before can be applied. The new nodes, edges and faces of the mesh over the parametric plane are obtained by means of the method for plane surfaces. In Fig. 8.a we can see the mesh in the plane. Although the faces seem irregular, we must notice that co-ordinates are angles and, for example, the longest edge in the top triangle is the same point in the space. Because of the division of the surface in angles we are going to get a more suitable result thanks to the properties of surfaces, where the angle follows better the shape of the surface than the rectangular co-ordinates.

The following step will be obviously to invert the transformation and get the cartesian co-ordinates from the angular co-ordinates using the equations shown before. As result we have the surface in three dimensions (Fig. 8.b).

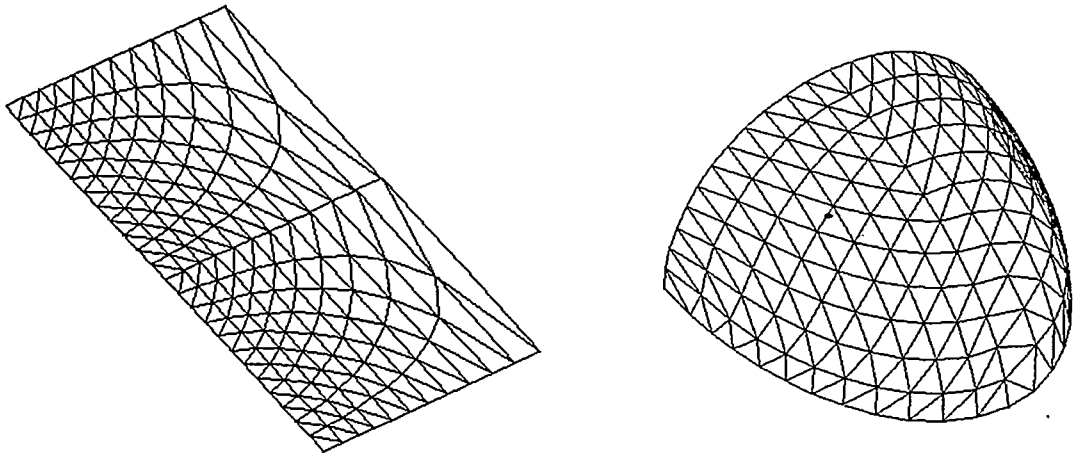


Figure 8. a) Angular mesh in parametric plane (ellipsoid), b) Real mesh (ellipsoid)

The last step would be, if it is necessary, to undo the first movement and rotation applied to the surface and replace the surface in the point of the space where it had been defined before the meshing.

Better results (more uniform meshes) are obtained meshing in angular co-ordinates than meshing, for example, the projection of the surface over a plane in a direct way. Although this last method is the simplest and would be useful for surfaces nearly parallel to a plane, usually the results are poor (see Fig. 9)

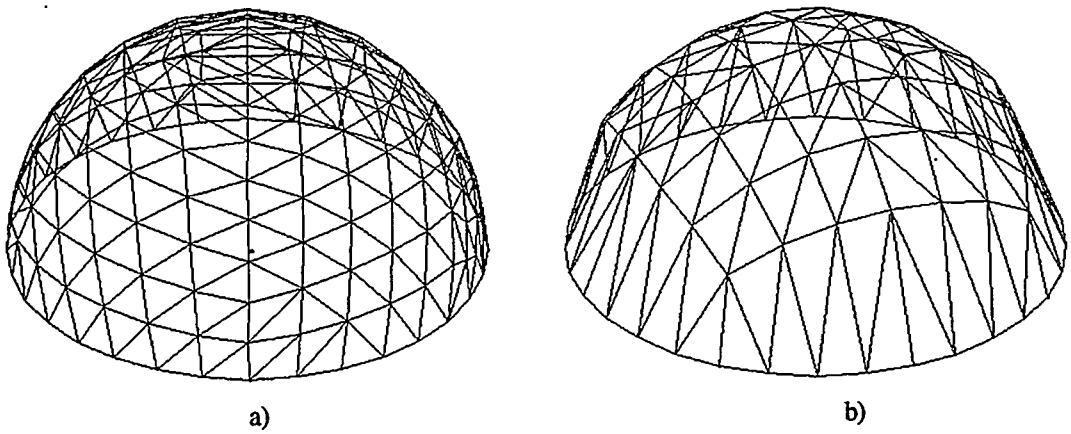


Figure 9. Sphere mesh: a) With parametric plane, b) With projection plane

Surfaces of Revolution

Another kind of surface that deserves our attention are the surfaces of revolution. These surfaces are formed when a line is rotated a certain angle taken an axis as reference. An example of revolution surface is a circular cylinder, that is, the revolution of a straight line. We have treated in a special way this kind of surfaces to facilitate their definition and improve their meshing.

In the mesh generation the proceeding method is similar to that used in quadric surfaces, that is, the transformation of co-ordinates. But in this case we are not going to look for the transformation of the surface from three co-ordinates to two co-ordinates. Instead of that, we are going to do the transformation to a parametric surface although this one is not contained in a plane. The objective is to get a parametric surface where we can apply the method for surfaces defined by their boundary with suitable results.

First of all we have the base line , that we consider placed in angle zero. There is another line with the same shape that the first one and parallel to it, but placed in a defined angle alpha. The transformation consists on creating the parametric surface using the following variables: the angle of rotation, the radius, that is, the distance between the line and the axis of rotation, and the projection of the line over the axis of rotation.

The first step is to place the initial line from an arbitrary situation in the space in a plane that must have one of the co-ordinates equal to zero (to use that co-ordinate as angle). We get this condition with a movement plus a rotation if it is necessary, in order to placed the line over the plane x=0, y=0, or z=0, and the rotation axis over a co-ordinate axis.

The lines of the real surface in different angles are always parallel. If the initial and final lines are joined by lines with the same radius and the same projection over the axis for all the angles those lines are obviously straight. We have seen before that this kind of surface formed by two straight lines and two curved lines could be meshed like a surface defined by their boundary. So we mesh this parametric surface dividing the angle of rotation in a uniform way. Thus, the final discretization will be uniform.

In order to undo the transformation, the first operation is to invert the movement and the rotation only in the variables radius and projection, and, finally, to apply to each node a spin in cartesian co-ordinates of the corresponding angle. An example of this type of meshing is shown in Fig. 10.

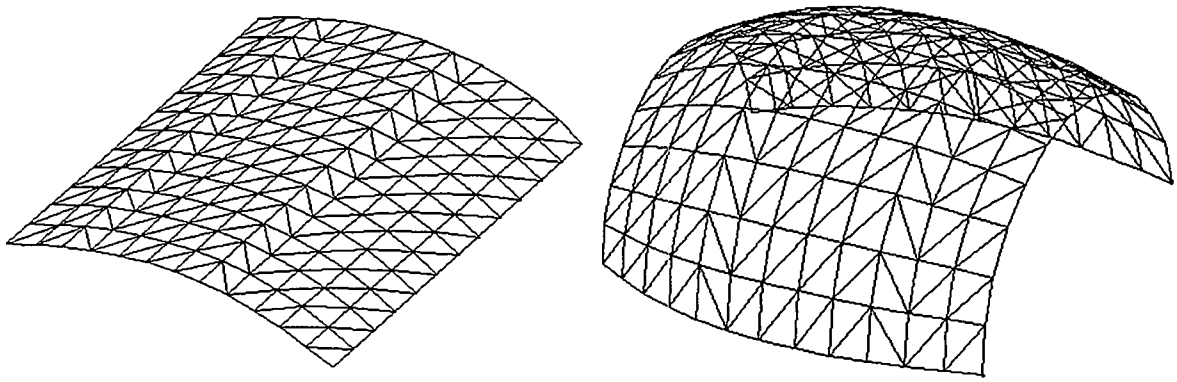


Figure 10. Meshing of surface of revolution: a) In the parametric surface, b) In the real space

Multiblock Definition

More complex surfaces, composed of the basic four types commented above, can be meshed with a multiblock definition of the surface. As an example of this mesh generation, a microwave cavity composed of one circular

cylindrical surface, two circular conical surfaces and two plane covers has been discretized. The resulting mesh is shown in figure 11.

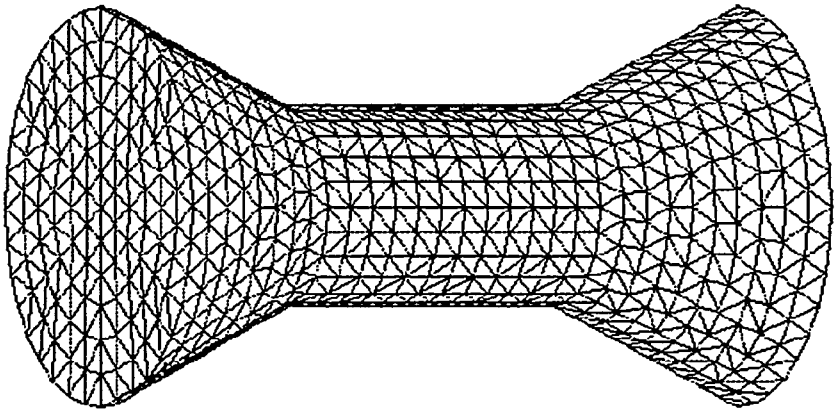


Figure 11. Mesh of a cylindrical-conical microwave cavity

Computational Cost

In order to evaluate the computational cost of the previous meshing methods, a set of mesh generations for plane surfaces (or defined by their boundary), quadrics and surfaces of revolution was realised in an HP-700 workstation. For plane surfaces, a square area was used, and a spherical cap for quadrics and surfaces of revolution. The mesh type for all cases was triangular (i.e., only triangular faces). The results are presented in Fig. 12. It can be observed a quasi-linear performance in the case of plane surfaces. For quadrics and surfaces of revolution the computational cost is bigger.

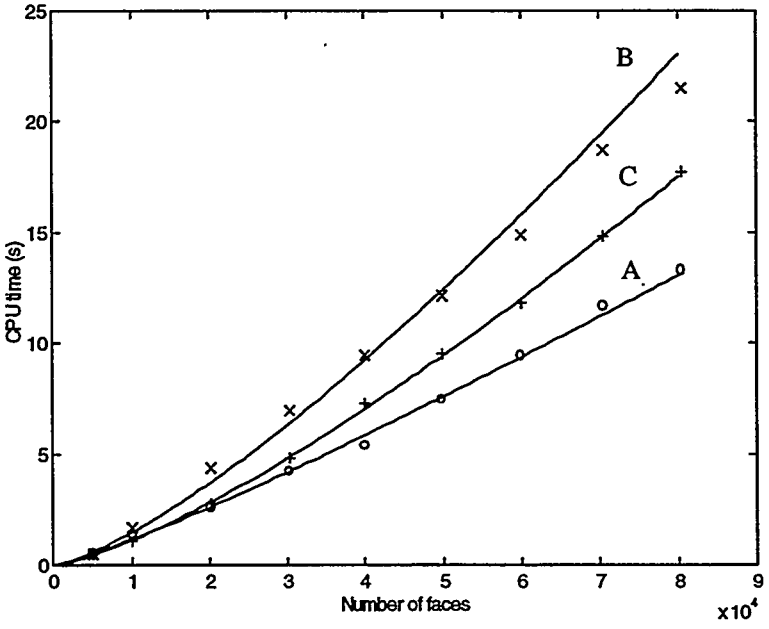


Figure 12. Meshing time for different methods: A) Plane surface, B) Quadric, C) Revolution

For the regression curve $t = An^B$, where t is the CPU time and n is the number of faces in the mesh, we obtained:
Method for plane surfaces: A=2.75e-5, B=1.16, r=0.966
Method for quadric surfaces: A=7.96e-5, B=1.32, r=1.18
Method for surfaces of revolution: A=6.2e-5, B=1.32, r=0.998
where r is the correlation coefficient.

Conclusions

A basic mesh generation method based on an advancing front / algebraic technique for generalized quadrilateral and triangular domains over plane surfaces has been presented. More complex domains can be discretized by means of a breakdown in areas topologically similar to triangles or quadrilaterals. Due to the use of transfinite interpolation, it is possible to use this method to discretize surfaces just defined by their boundary. For surfaces defined by their boundary and their equation it is necessary a more complex method. Here we have presented two methods for quadrics and surfaces of revolution based on the previous basic method. These methods transform the real surface in a plane surface (for quadrics) or in a surface defined just by their boundary (for surfaces of revolution), generate the mesh over that surfaces by means of the basic method, and invert the first transformation in order to obtain the final mesh over the real surface. Besides, a multiblock definition can be used for meshing combination of the four surfaces studied here.

To sum up, the general algorithm can be described as follows:

1. Identify the kind of surface.
2.1. If plane surface or defined by boundary:
  2.1.1. Apply the method for plane surfaces.
  2.1.2. End.
2.2. If quadric surface:
  2.2.1. Translate the surface to the origin and their axes over the co-ordinate axes.
  2.2.2. Apply a transformation over the surface boundary to place it in a plane.
  2.2.3. Mesh the surface in that plane using the method for plane surfaces.
  2.2.4. Undo the transformation, translation and rotation.
  2.2.5. End.
2.3. If revolution surface:
  2.3.1. Translate the surface to the origin and its axis over one of the co-ordinate axes.
  2.3.2. Transform the surface boundary in a boundary with two straight lines.
  2.2.3. Apply the method for plane surfaces.
  2.2.4. Undo the transformation, translation and rotation.
  2.2.5. End.

The tests realised for several types of plane, ruled, quadric and revolution surfaces show a high quality in the final mesh, obtaining little improvements with *a posteriori* refinement techniques as node reposition.

The method for plane surfaces does not need an element overlapping control, while classical advancing front methods spend time in this control or in the choice of the suitable node. Due to it, its computational cost is reduced, with a quasi-linear performance. The methods for quadrics and surfaces of revolution have higher computational costs due to the transformations from real to parametric surface and the inverse transformation.

References

[1] Möller P., Hansbo P., "On Advancing Front Mesh Generation in Three Dimensions", International Journal for Numerical Methods in Engineering, vol. 38, pp. 3551-3569, 1995

[2] Rassineux A., "3D Mesh Adaption. Optimization of Tetrahedral Meshes by Advancing Front Technique", Computer Methods in Applied Mechanics and Engineering, vol. 141, pp. 335-354, 1997

[3] Hassan O., Morgan K., "Unstructured Tetrahedral Mesh Generation for Three-Dimensional Viscous Flows", International Journal for Numerical Methods in Engineering, vol. 39, pp. 549-567, 1996

[4] Aurenhammer F., "Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure", ACM Computing Surveys, vol. 23, n. 3, pp. 345-405, 1991

[5] Cavendish J.C., Field D.A., Frey W.H., "An Approach to Automatic Three-Dimensional Finite Element Mesh Generation", International Journal for Numerical Methods in Engineering, vol. 21, pp. 329-347, 1985
[6] Zheng Y., Lewis R.W., Gethin D.T., "Three-Dimensional Unstructured Mesh Generation. Part 1. Fundamental Aspects of Triangulation and Point Creation", Computer Methods in Applied Mechanics and Engineering, vol. 134, pp. 249-268, 1996
[7] Loze M.K., Saunders R., "Two Simple Algorithms for Constructing a Two-Dimensional Constrained Delaunay Triangulation", Applied Numerical Mathematics, vol. 11, pp. 403-318, 1993
[8] Baehmann P.L., Wittchen S.L., Shephard M.S., Grice K.R., Yerry M.A., "Robust, Geometrically Based, Automatic Two-Dimensional Mesh Generation", International Journal for Numerical Methods in Engineering, vol. 24, pp. 1043-1078, 1987
[9] Yerry M.A., Shephard M.S., "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique", International Journal for Numerical Methods in Engineering, vol. 20, pp. 1965-1990, 1984
[10] Haber R., Shephard M.S., Abel J.F., Greenberg D.P., "A General Two-Dimensional, Graphical Finite Element Preprocessor Utilizing Discrete Transfinite Mappings", International Journal for Numerical Methods in Engineering, vol. 17, pp. 1015-1044, 1981
[11] Haber R., Abel J.F., "Discrete Transfinite Mappings and Meshing of Three-Dimensional Surfaces Using Interactive Computer Graphics", International Journal for Numerical Methods in Engineering, vol. 18, pp. 41-66, 1982
[12] Thompson J.F., "Structured and Unstructured Grid Generation", Critical Reviews in Biomedical Engineering, vol. 20, n. 1, pp. 73-120, 1992
[13] George P.L., *Automatic Mesh Generation. Application to Finite Element Methods*, Ed. John Wiley & Sons - Masson, 1991
[14] Zheng Y., Lewis R.W., Gethin D.T., "Three-Dimensional Unstructured Mesh Generation : Part 2. Surface Meshes", Computer Methods in Applied Mechanics and Engineering, vol. 134, pp. 269-284, 1996
[15] Lo S.H., "Finite Element Mesh Generation over Curved Surfaces", Computers & Structures, vol. 29, n. 5. pp. 731-742, 1988
[16] Shaw J.A., Weatherill N.P., "Automatic Topology Generation for Multiblock Grids", Applied Mathematics and Computation, vol. 52, pp. 355-388, 1992

Advancing Front Quadrilateral Meshing Using Triangle Transformations

Steven J. Owen[1,2], Matthew L. Staten[2], Scott A. Canann[1,2] and Sunil Saigal[1]

[1]Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A.
[2]Ansys Inc., 275 Technology Drive, Canonsburg, Pennsylvania, 15317, U.S.A.
steve.owen@ansys.com

Abstract

Quad-morphing is a new technique used for generating quadrilaterals from an existing triangle mesh. Beginning with an initial triangulation, triangles are systematically transformed and combined. An advancing front method is used to determine the order of transformations. An all-quadrilateral mesh containing elements aligned with the area boundaries with few irregular internal nodes can be generated.

KEY WORDS: mesh generation, quadrilateral, advancing front, surface meshing, Q-Morph, Paving

1. Introduction

Previous methods for unstructured quadrilateral meshing have included both direct and indirect methods. Indirect methods (Lo,1989; Johnston,1991; Lee,1994; Borouchaki,1998) include procedures that require an initial triangle mesh. Adjacent triangles are combined systematically, in most cases resulting in an all-quadrilateral mesh. While these methods can be fast, they can sometimes leave a large number of irregular nodes. An irregular node on the interior of a quadrilateral mesh is one that has more or less than four adjacent elements. Direct methods, on the other hand, do not involve an initial triangle mesh. Quadrilaterals are instead placed directly onto the surface. Quadrilaterals may be placed after first decomposing the surface into simpler regions (Baehmann,1987; Talbert,1991; Tam,1991; Joe,1995) or by using an advancing front approach (Zhu,1991; Lo,1985; Blacker,1991). In most cases, direct methods provide higher quality elements with fewer irregular nodes.

Of the direct, quadrilateral methods, the paving algorithm (Blacker,1991) provides several desirable characteristics. Blacker describes these as "(a) *Boundary Sensitive*. Mesh contours should closely follow the contours of the boundary. This characteristic is of particular importance since well-shaped elements are usually desirable near the boundary, (b) *Orientation Insensitive*. Rotating or translating a given geometry should not change the resulting mesh topology. A mesh generated in a transformed geometry should be equivalent to the original mesh transformed, and (c) *Few irregular nodes*. This is a critical mesh topology feature because the number of elements sharing a node controls the final shape of the elements, even after smoothing. Thus a mesh with few irregular nodes, especially near the boundary where element shape is critical, is often preferred.". The paving algorithm is currently in wide use. Since its initial development, it has been enhanced to incorporate three-dimensional surfaces (Cass,1996), as well as other improvements (White,1997).

In spite of the beneficial characteristics of paving, some quality and performance issues must be addressed. The advancing front method used by the paving technique requires many expensive intersection calculations as each row is placed in order to avoid overlapping elements. Figure 1(a) shows a simple case where intersection checks must be made. Figure 1(b) shows another case often encountered during paving where colliding fronts must merge. If element sizes differ greatly, poor element quality can often result.

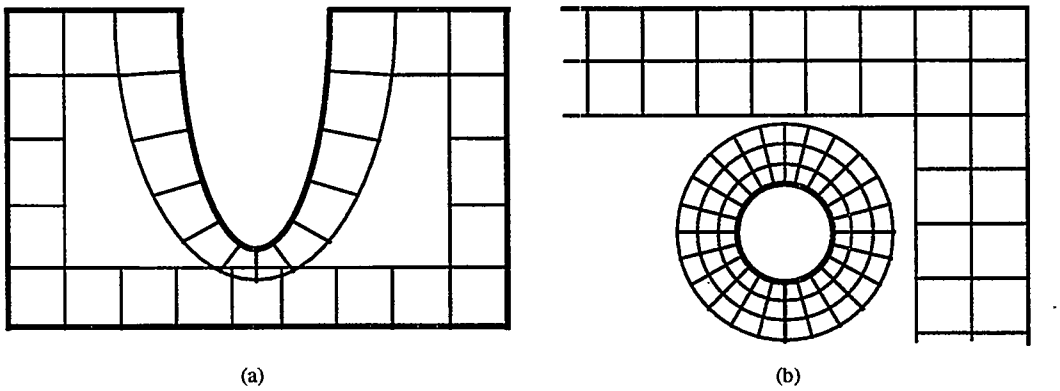


Figure 1. (a) First row of elements placed using paving algorithm illustrating interference of opposing elements. (b) Large element size differences between opposing fronts often encountered in paving leading to poor meshes.

This paper proposes an alternative to the traditional paving algorithm. The proposed Quad-morphing (Q-Morph) algorithm maintains the desirable features of paving while addressing some of its weaknesses. Q-Morph can be categorized as an unstructured, indirect method that utilizes an advancing front algorithm to form an all-quad mesh. As an indirect method it is able to take advantage of local topology information from the initial triangulation. Unlike other indirect methods it is able to generate boundary sensitive rows of elements, with few irregular nodes.

2. Outline of Quad-Morphing Algorithm

Quad-morphing is briefly outlined in the following steps:

1. **Initial Triangle Mesh.** The surface is first triangulated. This may be done using any surface triangulation method. Any sizing (Owen,1997) or adaptivity information should be built into the initial triangulation. The local sizing for the final quadrilateral mesh will roughly follow that of the triangle mesh.

2. **Front Definition.** The initial front is defined from the initial triangle mesh. Any edge in the triangulation that is adjacent to only one triangle becomes part of the initial front.

3. **Front Edge Classification.** Each edge in the front is initially sorted according to its *state*. The state of a front edge defines how the edge will eventually be used in forming a quadrilateral. Angles between adjacent front edges determine the state of an individual front. Front edges will be updated and reshuffled as the algorithm proceeds. Figure 2 shows the four possible states of a front, where the front edge is indicated by the bold line.

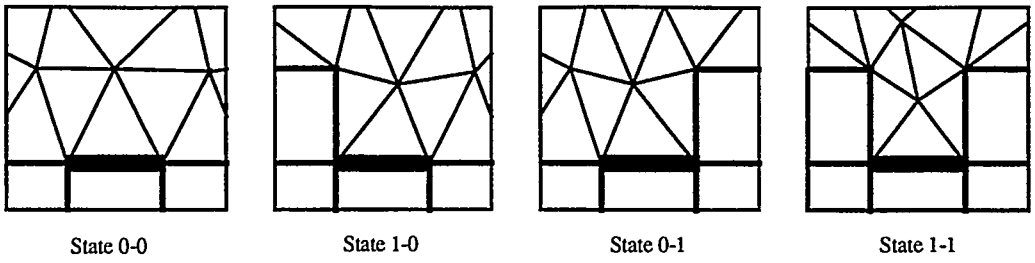


Figure 2. States of a front edge

4. **Front Edge Processing.** Each front edge is individually processed to create a new quadrilateral from the triangles in the initial mesh. Figure 3(a) shows front N_A-N_B in the triangulation ready to be processed. Front edges are handled differently according to their current state classification. As quadrilaterals are formed, the front is redefined and adjacent front edge states are updated. The current front always defines the interface between quadrilateral elements in the final mesh and triangle elements in the initial triangle mesh. This process can be further subdivided into the following sub-steps:

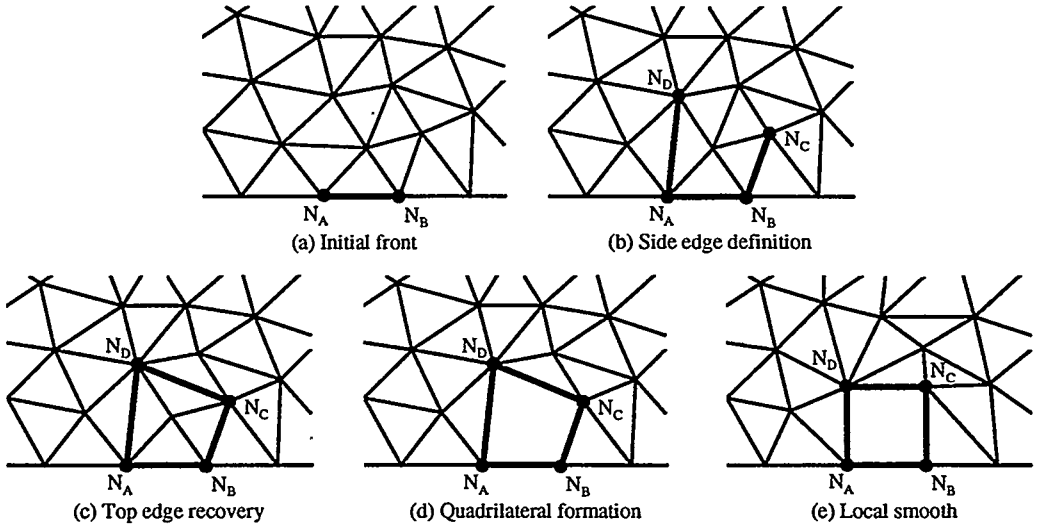


Figure 3. Steps demonstrating process of generating a quadrilateral from Front N_A-N_B.

- **Check for Special Cases.** Before proceeding to construct a quadrilateral from the current front, several special case scenarios are checked. These include situations where large transitions or small angles exist local to the front. In these cases a *seam*, or *transition seam* operation is performed.

- **Side Edge Definition.** Using the front edge as the initial base edge of the quadrilateral, side edges are defined. Side edges may be defined by using an existing edge in the initial triangle mesh, by swapping the diagonal of adjacent triangles, or by splitting triangles to create a new edge. In Figure 3(b), side edge N_B-N_C shows the use of an existing edge, while the side edge N_A-N_D was formed from a local swap operation.

- **Top Edge Recovery.** The final edge on the quadrilateral is created by an *edge recovery* process. During this process, the local triangulation is modified by using local edge swaps to

enforce an edge between the two nodes at the ends of the two side edges. Edge N_C-N_D in Figure 3(c) was formed from a single swap operation. Any number of swaps may be required to form the top edge.

- **Quadrilateral Formation.** Merging any triangles bounded by the front edge and the newly created side edges and top edge as shown in Figure 3(d) forms the final quadrilateral .
- **Local Smoothing.** The mesh is smoothed locally to improve both quadrilateral and triangle element quality as shown in Figure 3(e).
- **Local Front Reclassification.** The front is advanced by removing edges from the front that have two quadrilateral adjacencies and adding edges to the front that have one triangle and one quadrilateral adjacency. New front edges are classified by *state*. Existing fronts that may have been adjusted in the smoothing process are reclassified.

Front edge processing continues until all edges on the front have been depleted, in which case an all-quadrilateral mesh will remain, assuming an even number of initial front edges. When an odd number of boundary intervals is provided, a single triangle must be generated, usually towards the interior of the mesh.

5. **Topological Clean-up.** Element quality is improved by performing local quadrilateral transformations in an attempt to improve the individual edge valences at the nodes of the mesh.
6. **Smoothing.** A final smoothing pass is performed further improving the element qualities.

3. Implementation

3.1 Front Definition and Classification

The initial set of front edges is defined from the initial triangulation. All edges in the triangulation adjacent to a single triangle are used as the front. The state of a front edge is determined by computing the angle at the nodes on either end of the edge with each of its adjacent front edges. Practically, the state of a front edge is defined by two bits, the first representing the state at the *left* node and the second, the state at the *right* node. If the angle at either node is less than a specified tolerance ($3\pi/4$), the node bit is set (1); otherwise it is unset (0).

Angles at the nodes on the front can be approximated by summing the angles at adjacent triangles. In direct advancing front methods (Cass,1996), angles must be computed by first evaluating the surface normal and projecting edge vectors to a tangent plane. By approximating the angle at the front from the adjacent triangles, expensive geometric evaluations can be eliminated.

Edges are placed on one of four *state* lists as shown in Figure 2. Classifying front edges according to states serves two purposes. First, it defines which edges must be defined before a complete quadrilateral can be formed. Side edges must be defined only at the side of the front where the state bit has not been set. Second, it prioritizes which fronts will be processed first. Front edges in state 1-1 are given first priority followed by edges in states 0-1 and 1-0, followed by edges in state 0-0.

3.2 Front Edge Processing

Front edges are processed one at a time to form quadrilaterals from the initial triangulation. A front edge is *popped* from one of the four state lists, drawing from the higher states first. Priority is also given to the lowest *level* edge on the list. Edges in level zero are those on the initial front; level one are those on the front after the first row of quadrilaterals has been placed; level two after the second row; and so on. This ensures that an entire row of quadrilaterals will be placed before starting a new row.

Where large transitions are required, experience has shown that placing smaller quads first generally produces a better graded mesh. In order to do so, it is sometimes necessary to select short, higher level fronts before selecting longer lower level fronts. The criteria used for selecting the next front to be processed is, therefore, based not only on the current state and level of the front but also on its size.

3.2.1 Side Edge Definition

The current state of a front edge determines how the edge is processed. Front edges in states 0-0, 1-0 and 0-1 must first define either one or two *side* edges. A side edge may be formed in one of three ways: (1) an existing edge in the initial triangle mesh may be used, (2) the diagonal between two adjacent triangles may be swapped, or (3) an edge may be created by splitting a pair of triangles.

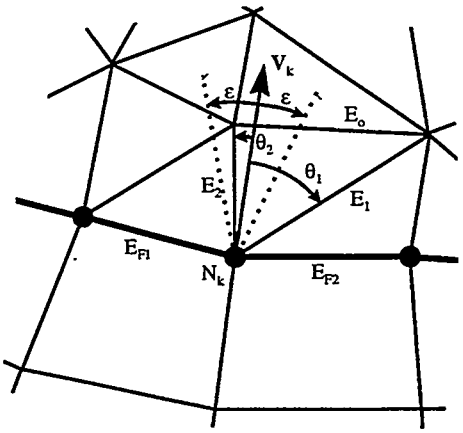


Figure 4. Side edge selection

Figure 4 shows a situation in which an existing edge is used. A new side edge is to be defined at node N_k, which is a node on the front between edges E_{F1} and E_{F2}. The ideal vector V_k for the new side edge is defined by bisecting the vectors formed by E_{F1} and E_{F2}. Angles θ_i are computed between V_k and all edges, E_i, of triangles sharing node N_k. The edge with the smallest angle θ is selected as the candidate side edge. The edge is selected, provided θ is less than a constant ε ($\pi/6$). Edge E_2 in Figure 4 is selected as the side edge in this situation.

When there is no angle θ_i less than ε, one of two options may be used. The opposite edge, (E_o in Figure 5) may either be swapped or split. The swap option is used if the angle β between V_k and V_m is less than ε. The split option is performed if $\beta > \varepsilon$ or the resulting length of E_k from a swap is excessively long compared to E_{F1} and E_{F2}. In this latter case, a new node N_n is defined, splitting edge E_o at the intersection of vector V_k and edge E_o. Edges E_k and E_m are also added to the triangle mesh, splitting the two triangles adjacent to edge E_o. Edge E_k is then used as the side edge of the prototype quad. The following shows a summary of the criteria for selection or creation of edge E_k, to be used as a side edge in the new quadrilateral:

$$E_k = \begin{cases} E_i & \text{for } \theta_1 < \varepsilon \\ \text{swap} \Rightarrow \overline{N_k N_m} & \text{for } \beta < \varepsilon \text{ and } \|N_k N_m\| < \sqrt{3}\dfrac{\|E_{F1}\| + \|E_{F2}\|}{2} \\ \text{split} \Rightarrow \overline{N_k N_n} & \text{otherwise} \end{cases} \quad [1]$$

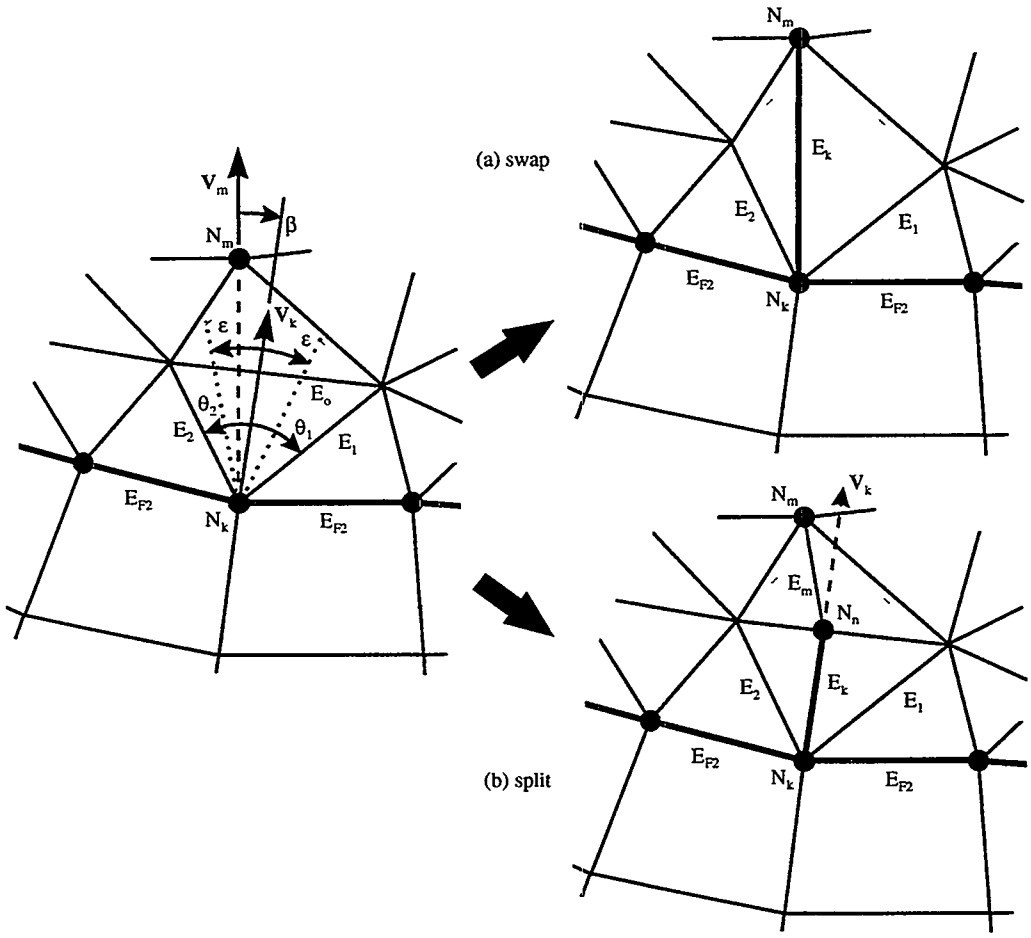


Figure 5. Side edge creation

3.2.2 Top Edge Recovery

Once the base and the two sides of the quadrilateral have been formed, the next step is to define the top edge. This is done by *recovering* the edge between the end nodes of the two sides. The edge recovery technique, which was presented independently in the literature by Jones (1990), Sloan (1993), and George (1991), is used commonly in boundary constrained Delaunay triangle meshing. Edge recovery involves systematically swapping edges between adjacent triangles until an edge is achieved between the desired nodes. An example of an edge recovery process is shown in Figure 6. The triangulation before recovery is shown at the top left with the successive swaps numbered. In this example a total of four local swaps was required to recover the edge N_C-N_D from the triangulation. Algorithm 1 details the transformations necessary to accomplish the recovery.

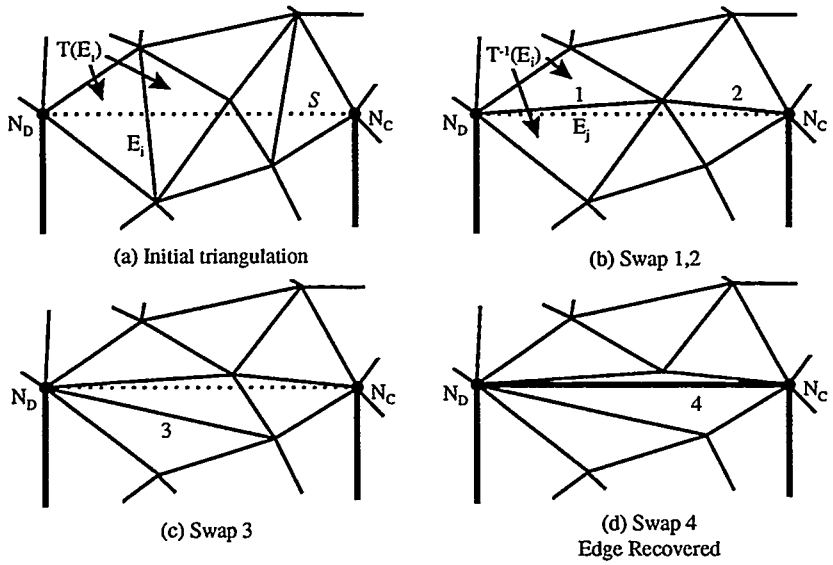


Figure 6. Edge Recovery Process

1. LET S be the line segment from N_C to N_D
2. LET $\Lambda(S)$ be a list of edges E_i that are intersected by S (see algorithm 2)
3. FOR EACH $E_i \in \Lambda(S)$
4. LET $T(E_i)$ be the set of 2 triangles adjacent E_i
5. LET $T^{-1}(E_i)$ be the set of 2 triangles where the diagonal edge E_i has been swapped.
6. IF area of both triangles in $T^{-1}(E_i) > 0$ THEN
7. Form $T^{-1}(E_i)$
8. Delete E_i from $\Lambda(S)$
9. LET E_j be the edge common to both triangles in $T^{-1}(E_i)$
10. IF E_j intersects S add E_j last on $\Lambda(S)$,
11. ELSE,
Place E_i last on $\Lambda(S)$
12. NEXT E_i on $\Lambda(S)$

Algorithm 1. Edge recovery process

The edge recovery process requires that an initial set of edges, $\Lambda(S)$, through which the recovered edge will pass, be first compiled. Algorithm 2 and Figure 7 detail how this may be accomplished.

3.2.3 3D Edge Recovery

Current literature assumes that the edge recovery process will be performed on a planar domain. Since this condition cannot be guaranteed in this application, an extension of the edge recovery process to include three-dimensional surfaces was necessary. Specifically, the dot product calculations of steps 5 and 16 in Algorithm 2 must be performed on vectors in a plane that is tangent to the surface. The tangent plane can be approximated from the neighboring triangles. For example, the tangent plane normal $\mathbf{P}_i$ at edge E_i can be estimated from the average normal vector of triangles $T_i(E_i)$ and $T_{i+1}(E_i)$. The dot product calculation in step 16 can then be replaced as:

$$\left(\left(\mathbf{P}_i \times \mathbf{V}_S\right) \times \mathbf{P}_i\right) \cdot \left(\left(\mathbf{P}_i \times \mathbf{V}_i\right) \times \mathbf{P}_i\right) < 0 \qquad [2]$$

Step 5 can be modified in a similar manner. An approximated tangent plane normal, $\mathbf{P}_C$, at N_C, can be defined as the average normal vector of the triangles in $T(N_C)$. The dot product calculation can then be replaced as:

$$\left(\left(\mathbf{P}_C \times \mathbf{V}_S\right) \times \mathbf{P}_C\right) \cdot \left(\left(\mathbf{P}_C \times \mathbf{V}_k\right) \times \mathbf{P}_C\right) > 0 \text{ and } \left(\left(\mathbf{P}_C \times \mathbf{V}_S\right) \times \mathbf{P}_C\right) \cdot \left(\left(\mathbf{P}_C \times \mathbf{V}_{k+1}\right) \times \mathbf{P}_C\right) > 0 \qquad [3]$$

1. LET $T(N_C)$ be the ordered set of ccw triangles and quads adjacent N_C, $T_k(N_C) \in T(N_C)$
2. LET $E(N_C)$ be the ordered set of ccw edges adjacent N_C; $E_k(N_C) \in E(N_C)$, where $E_k(N_C)$ and $E_{k+1}(N_C)$ are on $T_k(N_C)$
3. LET $\mathbf{V}_k$ be the vector from N_C in direction of $E_k(N_C)$
4. LET $\mathbf{V}_S$ be the vector from N_C to N_D
5. FOR EACH $T_k(N_C) \in T(N_C)$
   IF $\mathbf{V}_S \cdot \mathbf{V}_k > 0$ and $\mathbf{V}_S \cdot \mathbf{V}_{k+1} < 0$, LET $T_i(E_i) = T_k(N_C)$
6. LET E_i be the edge opposite N_C on $T_k(N_C)$
7. IF E_i is not on front THEN, Add E_i to $\Lambda(S)$, ELSE *fail*
8. WHILE not *done*
9. LET $T_{i+1}(E_i)$ be the triangle adjacent E_i where $T_{i+1}(E_i) \neq T_i(E_i)$
10. IF N_D is on $T_{i+1}(E_i)$, THEN *done*
11. $T_i(E_i) = T_{i+1}(E_i)$
12. LET N_i be the node opposite E_i on $T_i(E_i)$
13. LET $\mathbf{V}_i$ be the vector from N_C to N_i
14. LET E_n be the next ccw edge on $T_i(E_i)$ from E_i
15. LET E_{n+1} be the next cw edge on $T_i(E_i)$ from E_i
16. IF $\mathbf{V}_S \cdot \mathbf{V}_i < 0$, THEN $E_i = E_n$, ELSE $E_i = E_{n+1}$
17. IF E_i is not on front THEN, Add E_i to $\Lambda(S)$, ELSE *fail*
18. CONTINUE

Algorithm 2. Formation of $\Lambda(S)$

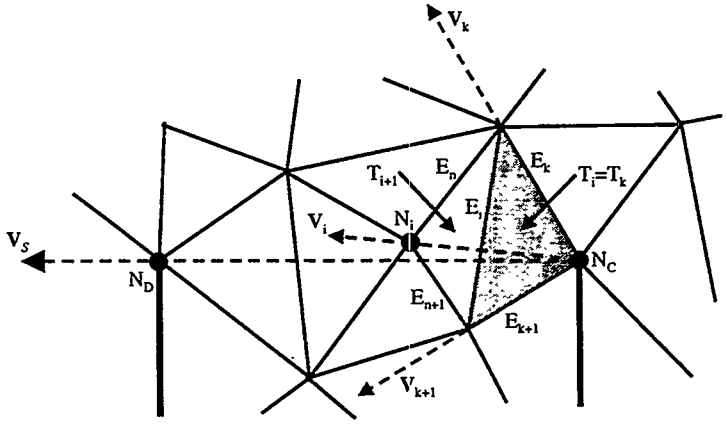


Figure 7. Formation of $\Lambda(S)$

3.3 Quadrilateral Formation

The quadrilateral is formed from the edge on the front, two side edges, and recovered top edge. Before forming the quadrilateral, the triangles contained within the four edges must first be deleted. This can be done with a procedure that starts with the triangle adjacent to the front edge and recursively advancing to adjacent triangles, deleting them as it proceeds. Unused nodes and edges are also removed. The recursion continues until the top or side edges of the prototype quadrilateral are encountered.

3.4 Local Smoothing

Smoothing is an important part of the Q-Morph algorithm. Node locations local to the new quadrilateral are readjusted to improve element shape. This includes nodes both behind and in front of the current front. Local smoothing accomplished before processing the next front, as smoothing angles between adjacent fronts will affect the front *states* and hence the final topology of the quadrilateral mesh. In practice, any node on the new quadrilateral and any node connected by an edge are smoothed. Nodes on the front must be handled differently than those behind or ahead of the front.

For nodes not located on the current front, a simple Laplacian smooth (Field,1988) is adequate, or alternatively, a modified length weighted Laplacian smooth as suggested by Blacker (1991). Since it is at the front where the new quadrilateral elements are formed, it is more critical at these nodes that the smoothing produce well proportioned quadrilaterals. A modified form of the smoothing process suggested by Blacker is used for the *row* nodes defined in that reference. These are nodes connected to exactly two adjacent quadrilaterals at the front. The smoothing process presented by Blacker involves an isoparametric smooth (Hermann,1976) followed by corrections for squareness and angle smoothness. For cases where large transitions may be involved, it is useful to take advantage of sizing information provided by the triangles ahead of the front. As a result, an improved transition can be achieved.

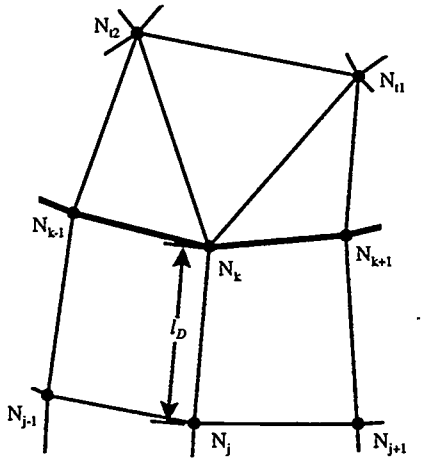


Figure 8. Definition of edge length l_D at node on front N_k

Let l_D be the length of the edge N_k-N_j where N_k is the node on the front to be smoothed as shown in Figure 8. Where a very large transition in element size is required, l_D can be defined as the average length of all edges connected to N_k. For smaller transitions, fewer irregular nodes will be created if equation 4 is used. Let n be the number of nodes ahead of the front connected to N_k, then l_D can be defined as an average of edge lengths on adjacent quadrilaterals and edges ahead of the front as follows:

$$l_D = \frac{\left\|N_{k-1} - N_{j-1}\right\| + \left\|N_{j-1} - N_j\right\| + \left\|N_{k+1} - N_{j+1}\right\| + \left\|N_{j+1} - N_j\right\| + \sum_{i=1}^{n} \left\|N_k - N_{ti}\right\|}{4+n} \quad [4]$$

The method used for computing l_D is decided purely on heuristics. For the current implementation, if the ratio of largest to smallest edge length, t_r, on the boundary is less than 2.5, then the smoothing method proposed by Blacker (1991) is used unmodified. This method is preferred since it tends to produce the fewest number of irregular nodes. As t_r increases, it is necessary to introduce irregular nodes so that a smooth transition may be afforded. Equation 4 is used for l_D when t_r is greater than 2.5, and the average length of adjacent edges to N_k is used when t_r is greater than 20.

When smoothing nodes at the front, as a result of improving the quadrilaterals, it is possible that the triangles immediately ahead of the front become inverted. While the Q-Morph algorithm does not require triangles to be near equilateral, it does rely on the fact that all triangles are uninverted throughout the meshing process. To ensure that this does not occur, triangles and quadrilaterals neighboring the smoothed node must be checked for consistent normals. In the case of an inverted element, an adjustment must be made to the new node location. The node location can be adjusted incrementally on a vector from the old location to the new location until all neighboring elements are no longer inverted.

3.5 Local Update and Reclassification of Fronts

Once a new quadrilateral has been formed, it becomes necessary to update the current list of fronts. Fronts are redefined so that edges now adjacent to both a triangle and a quadrilateral are placed on one of the four state lists and edges no longer adjacent to a triangle are removed from the state lists. In addition, other nearby edges on the front may need to be updated. By smoothing nodes on the front, angles between adjacent fronts may have been changed; thus necessitating moving fronts to an alternate state list.

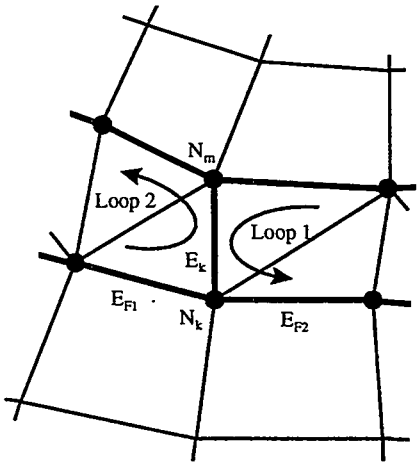


Figure 9. Selection of side edge forming of new front loop

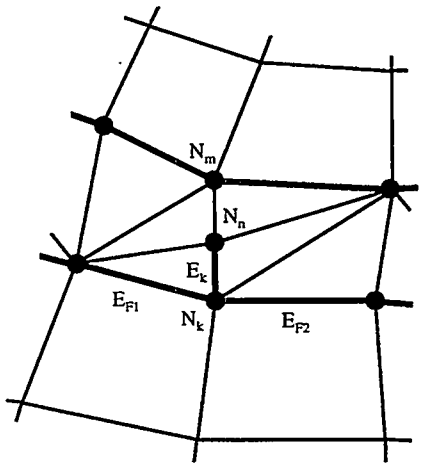


Figure 10. Splitting of side edge to maintain even loop

3.6 Closing the Front

When defining a new side edge, the opposite node, N_m, as shown in Figure 9, may lie on an opposing front. Edge E_k may have been selected from the existing triangles as in Figure 4, or from a swap operation as in

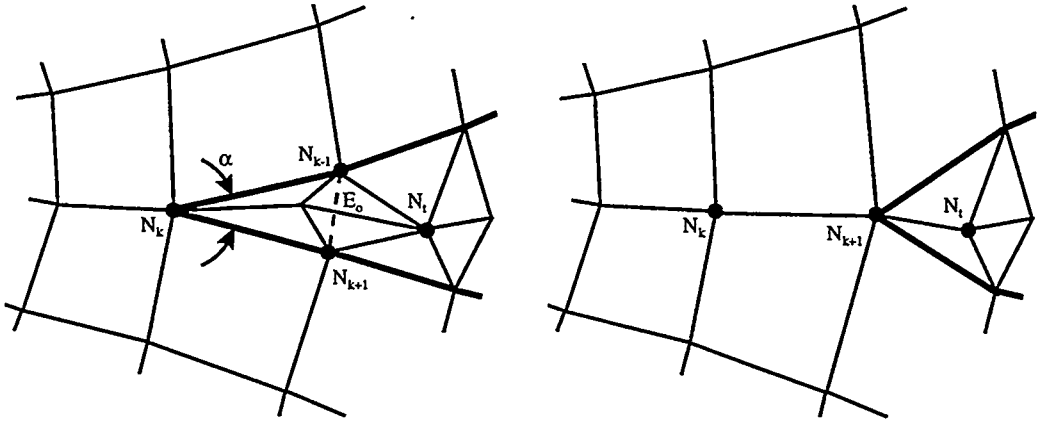
Figure 5. In either case, E_k can only be used if the number of edges on each resulting front loop is even. A front loop may be defined as all edges on a front comprising a continuous unbroken ring. Any number of loops may be active at a given time in the process of meshing. In order to ensure an all-quadrilateral mesh, it is required that any individual loop be comprised of an even number of edges. Selecting an edge to be used as a new side edge may result in the formation of a loop with an odd number of front edges. To avoid this occurrence, the potential number of front edges on the new loop about to be formed is first determined. If the number of edges is even, then the selection is made and a new loop is defined. If the number of edges on the new loop is odd, no connection is made. Instead the edge, E_k, is split creating a new node, N_n, as shown in Figure 10. This permits a subsequent side selection operation to define an even number of front edges on adjacent loops.

If the side edge is to be created from a swap or split operation, as in Figure 5, the edge E_o should first be checked to see if it is part of the opposing front. Since swapping or splitting E_o would destroy the continuity of the front, the operation should not be performed. For this reason, it is advantageous, when N_m is on an opposing front, to allow for a larger value of ε. This increases the chances of selecting an existing edge and closing the front.

3.7 Seams

When the angle, α, between two adjacent edges on the front is small, then a seaming operation is performed. Although paving (Blacker,1991) incorporates a seaming operation, it must also be defined within the context of the Q-Morph algorithm, in order to account for triangles ahead of the front. In addition to angle α, the criteria for seaming is also based on the number of quadrilateral elements, n_Q, adjacent to the node to be seamed. Blacker proposes the following criteria for seaming:

$$\left\{\begin{array}{l}\alpha < \varepsilon_1 \text{ for } n_Q \geq 5 \\ \alpha < \varepsilon_2 \text{ otherwise}\end{array}\right\} \quad \text{where } \varepsilon_1 < \varepsilon_2 \qquad [5]$$



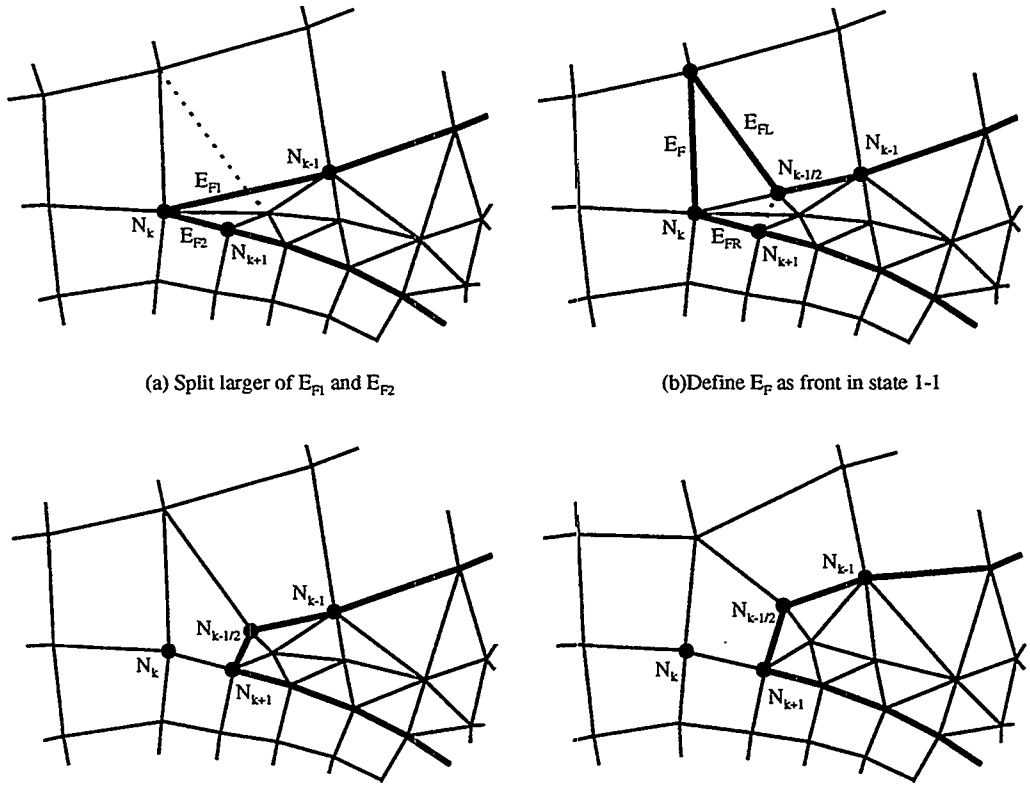
(a) Front to be seamed (b) Seam closed

Figure 11. Seaming operation

To accomplish the seam, nodes N_{k-1} and N_{k+1}, shown in Figure 11, must be merged. Let N_k be the node on the front whose angle, α, satisfies equation 5. The temporary edge, E_o, connecting N_{k-1} and N_{k+1}, if not already part of the initial triangle mesh, is first recovered using Algorithm 1 above. Knowing E_o, its

adjacent triangle comprising nodes N_{k-1},N_{k+1},N_t, can be determined. With this information, all triangles and nodes bounded by the quadrilateral composed of nodes N_{k-1},N_k, N_{k+1}, N_t can be deleted. Finally nodes N_{k-1} and N_{k+1} can be merged at a location midway between their initial positions. Local smoothing is then performed, followed by an update of the states of any adjacent fronts that may have changed. In rare cases, the new location of node N_{k+1} may result in one or more inverted elements. In this case, an optimization based smoothing algorithm (Canann,1998) is employed which adjusts the node location with the objective of improving a local shape metric for neighboring elements.

Another operation described by Blacker (1991) is the transition seam. This is required when there is a large difference in size between adjacent fronts. In Figure 12(a), E_{F1} and E_{F2} are the edges on the front adjacent to N_k. If the ratio of lengths between E_{F1} and E_{F2} is greater than 2.5, then a transition seam operation is performed. The longer of the two edges, E_{F1} and E_{F2}, is first split at its midpoint adding node $N_{k-1/2}$ or $N_{k+1/2}$. In Figure 12(b), E_{F1} is split, dividing its adjacent triangle and quadrilateral as shown. Edges E_F, E_{FL}, and E_{FR} can then be defined as front edges. With this new configuration, edge E_F can be processed as a front in state 1-1, requiring only the recovery of the top edge between $N_{k-1/2}$ and N_{k+1} as in Figure 12(c). Finally, the transition seam is completed after local smoothing and updating of the front as shown in Figure 12(d).



(a) Split larger of E_{F1} and E_{F2}

(b)Define E_F as front in state 1-1

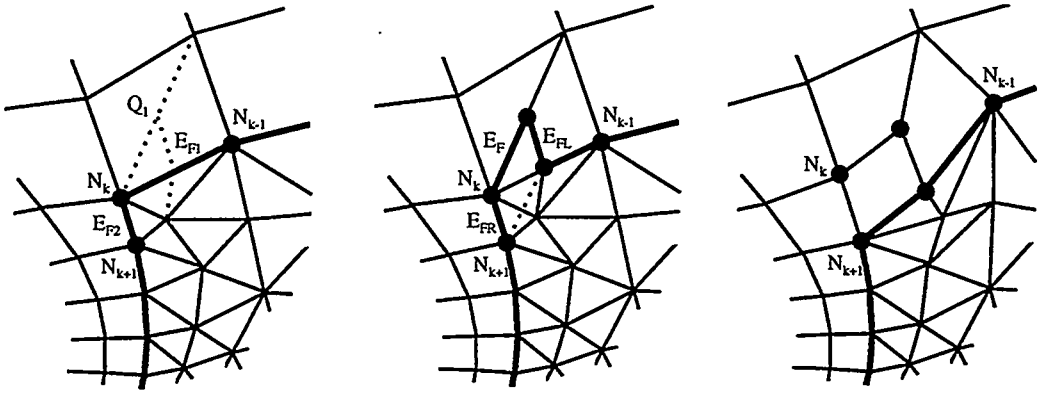
(c) Form new transition quad

(d) Smooth and reclassify fronts

Figure 12. Transition seam operation

3.8 Transition Split

An operation useful for improving transitions is shown in Figure 13. Although similar to the transition seam operation shown in Figure 12, it is applicable when $\alpha > \varepsilon_1$ or $\alpha > \varepsilon_2$ (see equation 5). The transition split operation is performed when the ratio of lengths between E_{F1} and E_{F2} is greater than 2.5. Let Q_1 be the quadrilateral adjacent to the longer of E_{F1} or E_{F2}. Q_1 is split into two quadrilaterals and a single triangle, as shown in Figure 13(a). Front E_{F1} in Figure 13(a) is split at its midpoint, which causes its adjacent triangle to be split. An additional node is then inserted at the centroid of Q_1. As a result, new front edges, E_F, E_{FL} and E_{FR}, can be defined as shown in Figure 13(b). Similar to the transition seam, E_F can now be defined as a front in state 1-1 and processed to create a new quadrilateral. Figure 13(c) shows the configuration after smoothing and reclassification of fronts.



(a) Split Q_1 (b) Define E_F as front in state 1-1 (c) Form transition quad and smooth

Figure 13. Transition split operation

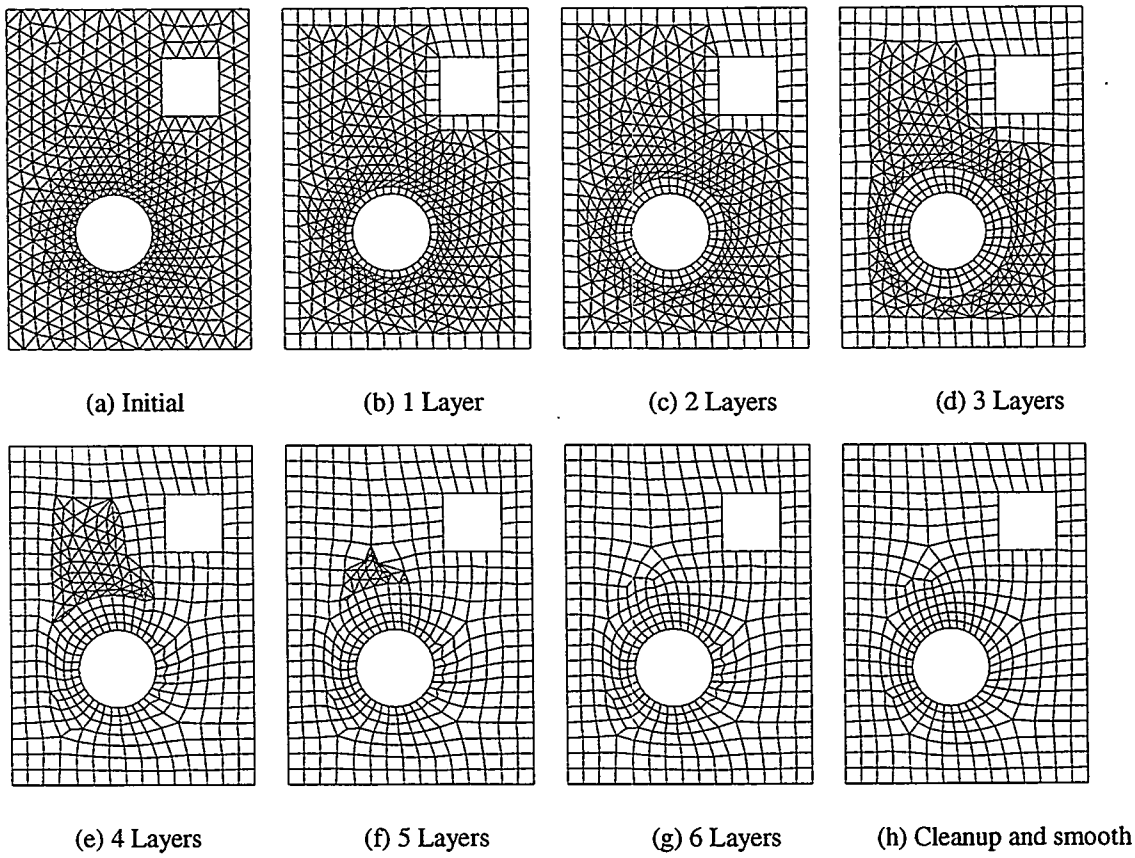
3.9 Topological Cleanup and Smoothing

Once all of the front edges have been processed and an all-quadrilateral mesh is generated, it is often beneficial to perform local topological cleanup operations to decrease the number of irregular nodes. Although Q-Morph attempts to minimize the number of irregular nodes, they may, as a necessity, be introduced as a result of non-orthogonal boundaries or from element size transitions. Irregular nodes may also be introduced when the local nodal density and connectivity provided by the initial triangle mesh is insufficient to generate equilateral quadrilaterals. Many of these irregular nodes can be eliminated through local topological cleanup. Topological cleanup modifies the connectivity of the quadrilaterals through a series of single-step operations including edge swaps, face opens, face closes, and two-edge node removals/insertions (Staten,1997; Canann,1994; Kinney,1997). In addition, these single operation modifications can be combined into multi-step modifications to further decrease the number of irregular nodes. By reducing irregular nodes through topological cleanup, the mesh contours can more closely follow the contours of the boundary.

The final smoothing step involves a limited number of iterations of a constrained Laplacian smoothing algorithm. Each node is moved to the centroid of its neighbors only if an improvement in element shape metric (Lee,1994) would result. In situations where Laplacian smoothing produces poor results, an optimization based smoothing (Canann,1998) operation may be performed.

4. Example Problems

Four example problems shown in Figure 14 to Figure 18 demonstrate various features of the Q-Morph algorithm. The first example, shown in Figure 14, demonstrates the progression of the Q-Morph algorithm on a simple planar domain with two holes. Figure 14(a) shows the initial triangle mesh before Q-Morph begins. In this case an advancing front triangle mesher (Canann,1997) was used to create the triangles. The method used for triangulation is unimportant, inasmuch as the appropriate nodal density is provided. Figure 14(b)-(g) show the progression of the algorithm as each successive layer of elements is completed. Figure 14(c) shows an additional layer of small elements meshed on the internal circle loop before meshing the larger elements of the outer loop. To improve element transitions, provision is made in Q-Morph to mesh loops with smaller elements before those with larger elements. The mesh is completed in Figure 14(h) after a final pass of cleanup and smoothing.



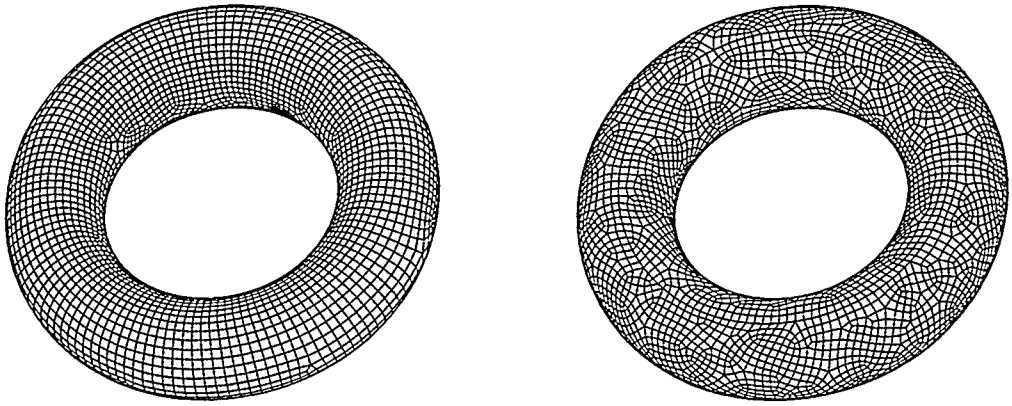
(a) Initial (b) 1 Layer (c) 2 Layers (d) 3 Layers

(e) 4 Layers (f) 5 Layers (g) 6 Layers (h) Cleanup and smooth

Figure 14. Progression of Q-Morph

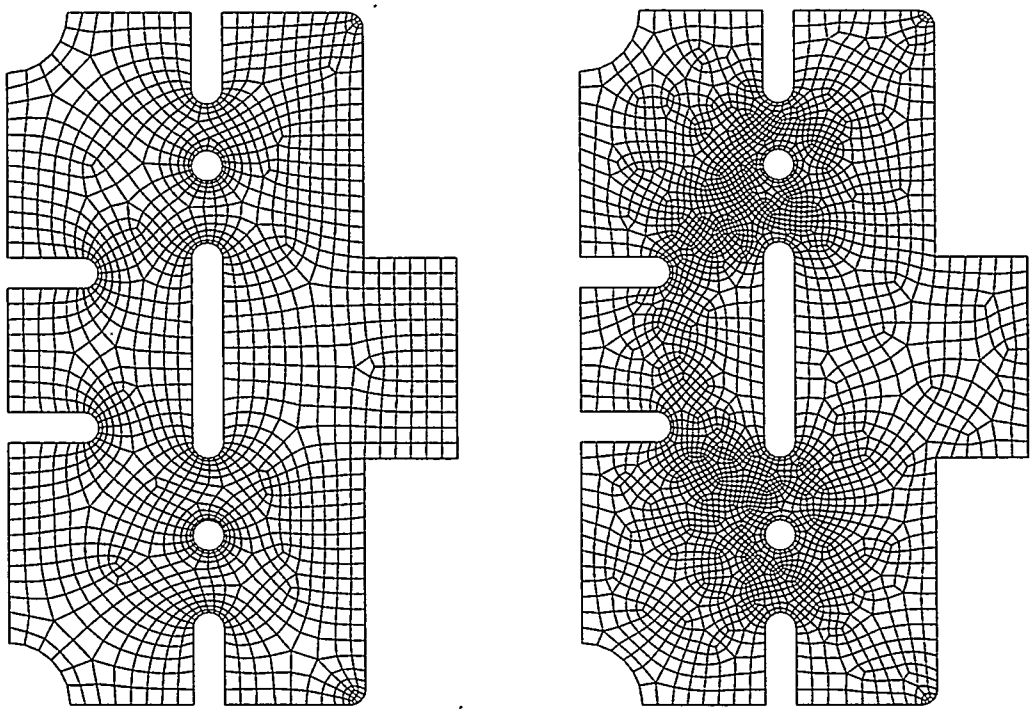
Figure 15 and Figure 16 compares Q-Morph against Lee's (1994) quad meshing algorithm, which uses an indirect method, coupled with an advancing front scheme to combine triangles into quadrilaterals. The toroidal surface of Figure 15 is composed of four surface patches represented as rational B-Splines. Q-Morph utilizes projection and geometric evaluation routines as part of the local and final smoothing procedures to maintain nodal locations on the three-dimensional surface. Both Figure 15(a) and (b) were generated using the same initial triangle mesh as well as the same cleanup and smoothing procedures. Despite using an advancing front scheme, Lee's algorithm shown in Figure 15(b), has difficulty maintaining well-aligned rows of elements introducing many irregular internal nodes. Figure 16 further illustrates the

ability of the Q-Morph algorithm to generate well-aligned rows of elements parallel to a complex domain boundary, while still maintaining the required element size transitions.



(a) Q-Morph (b) Lee's Algorithm

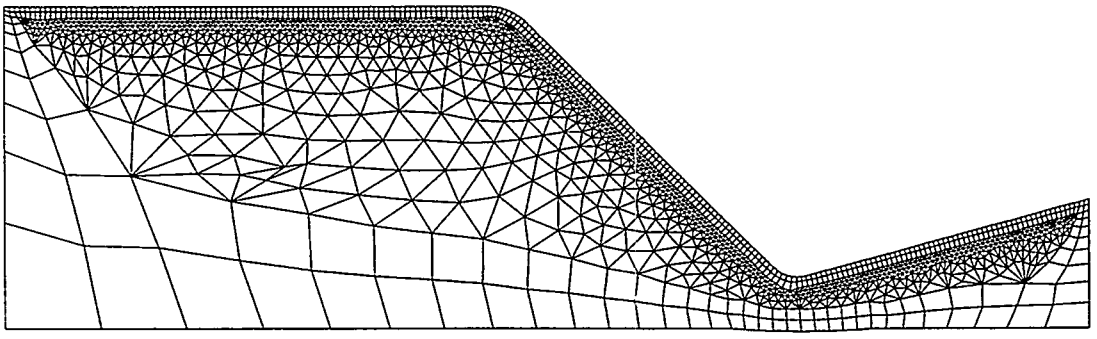
Figure 15. Results of Q-Morph compared with Lee's (1994) advancing front indirect method on toroidal surface



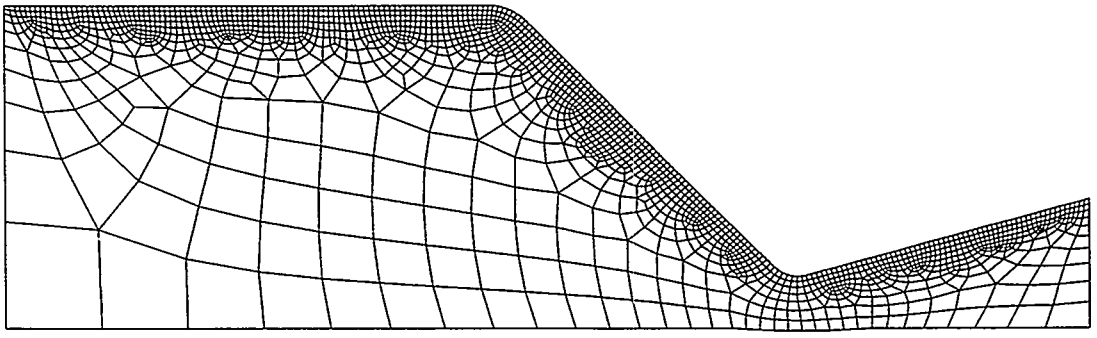
(a) Q-Morph (b) Lee's Algorithm

Figure 16. Comparison of Q-Morph with Lee's Algorithm illustrating element boundary alignment

Figure 17 demonstrates the use of Q-Morph with a planar surface requiring a high degree of transition. Figure 17(a) shows the partially completed quad mesh with two layers of quads placed. Figure 17(b) shows the same area after final cleanup and smoothing. In order to maintain a specified nodal density near the top of the area, a sizing function (Owen,1997) was used during the triangle meshing process. The algorithm's ability to maintain the desired mesh density while still enforcing well-aligned rows of elements transitioning quickly to larger size elements is demonstrated in this example.



(a) Partially completed quad mesh



(b) Mesh after cleanup and smoothing

Figure 17. Large transition mesh for CFD application

The final example in Figure 18 is an industrial application of the Q-Morph algorithm. For this example, the model consisting of 104 separate areas was first constructed using a commercial CAD software application. Surfaces are once again represented by rational B-splines. In practice, the Q-Morph algorithm is used as part of a set of meshing tools which also include mapping methods. In this example, the narrow fillet regions are better represented with a mapped meshing technique, which can more appropriately create elements of high aspect ratio. Q-Morph is better suited to generating near-equilateral, isotropic quadrilaterals. Selection of the appropriate quad meshing method can be done automatically based on the number of lines comprising the area and its aspect ratio. After assigning line divisions, each area is first meshed with triangles and then transformed into quadrilaterals. Note that where an odd number of divisions is assigned to an area, Q-Morph forms a single triangle in the mesh, generally towards the interior of the area.

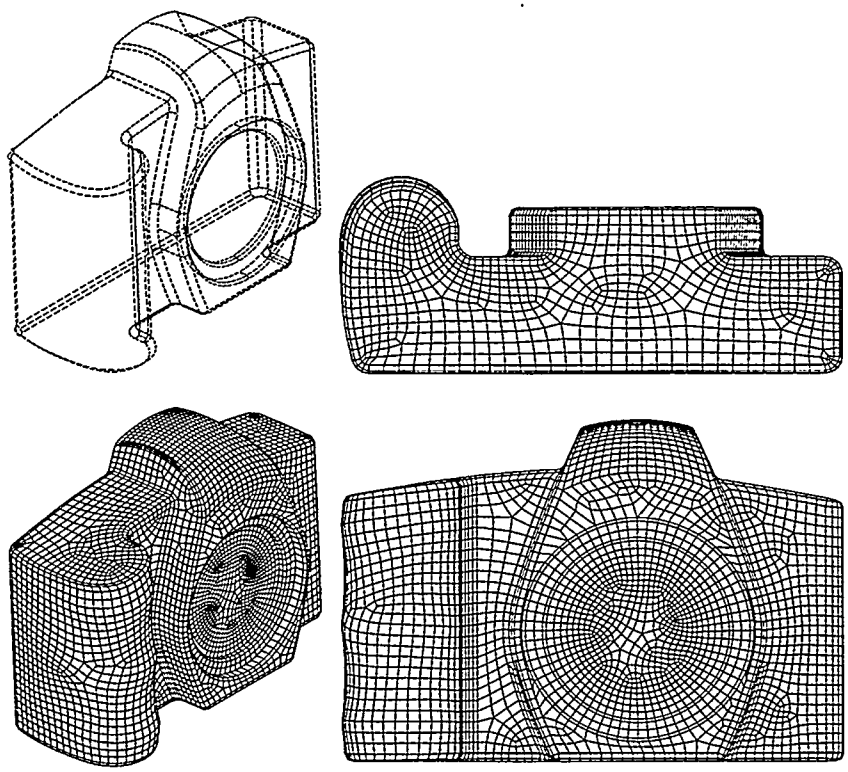


Figure 18. Industrial application of Q-Morph combined with mapped meshing

5. Performance

Both speed and element quality of the resulting elements from Q-Morph was evaluated as part of this study. Table 1 shows performance results from two of the example problems above. For the models in Figure 15 and Figure 17, various element densities were specified and their results noted.

5.1 Speed

Table 1 shows CPU times for both the quad-conversion and the clean-up and smoothing portions of the Q-Morph algorithm. Tests were performed on a 195 MHz SGI UNIX workstation. For the toroidal surface in Figure 15, times are necessarily affected by the number of geometric evaluations required. Times range from 141 to 242 quads converted per CPU second. This is in contrast to the flat surface of Figure 17, where times ranged from 313 to 369 quads converted per CPU second. Clean-up and smoothing times were however slower for Figure 17 than for Figure 15 as the transition in element size defined by the quad conversion required additional iterations to converge. A wide variety of factors can affect the overall speed of the algorithm. Table 1 illustrates two cases where geometry and element transition is critical.

5.2 Element Quality

Element quality was measured by shape metric, β similar to that described by Lo (1989) and Canann (1998). For this implementation, β is defined as the minimum triangle shape metric, α, defined by any of the four possible triangles formed by the vertices of the quadrilateral. A β value of 1.0 represents a perfect square, while a value of 0.0 represents a quadrilateral with a single corner angle of π. Concave or inverted quadrilaterals may be represented by negative values of β.

Both minimum and average metrics immediately following quad conversion and after clean-up and smoothing are shown in Table 1. In some cases, inverted or poorly shaped quadrilaterals can be created during the quad conversion as indicated by the negative or zero metrics. Average metrics are however very high. In all cases tested, clean-up and smoothing improved the poorly shaped quads to well within usable limits. Table 1 also shows cases where a single triangle is created in the mesh. This occurs automatically in order to resolve situations where an odd number of boundary intervals are specified.

5.3 Robustness

A diversity of surfaces has been meshed using the Q-Morph algorithm and is currently part of a commercial FEA software release (Ansys,1998). As such, it has been successfully ported to a wide variety of platforms, including Windows, NT and UNIX environments. In general, the Q-Morph algorithm is most beneficial on surfaces where the geometric feature sizes are larger than the specified element size. In addition, high quality quadrilaterals can be expected, provided the background triangle mesh captures the details of the surface and the background triangles are of reasonable quality (ie. $\alpha > 0.1$). In most cases where these conditions are not met, Q-Morph will be successful, however element quality may suffer.

	Triangle to Quad Conversion					Clean-up and Smoothing				
Model	Num. Quads	Num. Tris	Min. Metric	Avg. Metric	CPU Time (s)	Num. Quads	Num. Tris	Min. Metric	Avg. Metric	CPU Time (s)
Figure 15	351	0	0.371	0.893	1.45	350	0	0.515	0.905	0.44
	1208	0	0.391	0.905	5.31	1206	0	0.529	0.925	1.89
	4870	0	0.170	0.936	25.4	4845	0	0.376	0.948	10.9
	19209	0	-0.155	0.940	136	19070	0	0.359	0.949	34.2
Figure 17	727	1	0.00	0.740	2.32	696	1	0.255	0.802	2.23
	1892	0	0.00	0.790	5.19	1785	0	0.344	0.859	4.36
	4472	1	0.00	0.811	12.1	4288	1	0.370	0.889	15.7
	10581	0	-0.155	0.817	33.4	10231	0	0.382	0.889	31.4

Table 1. Performance results from Q-Morph

6. Future Work

The Q-morph algorithm has been implemented and is currently part of a recent commercial release of ANSYS (1998). Although significant improvements have been made to the quad meshing capabilities of the ANSYS meshing tools through the addition of Q-morph, the main research objective was to define a prototype for the more difficult problem of hexahedral meshing. Work is currently under way to extend the principles introduced by Q-morph, to a general-purpose hex-dominant (H-morph) mesher.

7. Conclusion

The Q-Morph algorithm is an indirect quadrilateral meshing algorithm that utilizes an advancing front approach to transform triangles into quadrilaterals. It generates an all-quadrilateral mesh, provided the number of intervals on the boundary is even. The resulting mesh has few irregular internal nodes and produces elements whose contours, in general, follow the boundary of the domain. Overall element quality is excellent. The Q-Morph algorithm borrows many of its techniques from the paving method (Blacker,1991; Cass,1996) but adapts them for use as an indirect method, operating on an existing set of triangles. In so doing, it is able to improve upon the paving technique by resolving some of its inherent difficulties. The intersection problem, common to most direct methods of advancing front meshing, is eliminated by relying on the topology of the initial triangle mesh to close opposing fronts. Improvements also include facility for handling individual element placement through the use of *states* for classifying front edges. Facility for handling transition in element sizes has also been addressed through the use of sizing information provided by the initial triangle mesh and the definition of specific transformations that enable improved mesh transitions. Additionally, the initial triangle mesh provides information that reduces the cost of direct evaluations on three dimensional surface geometry.

References

ANSYS 5.5 (1998) Software program, ©Ansys, Inc., Canonsburg, PA USA

Baehmann, P. L., S. L. Wittchen, M. S. Shephard, K. R. Grice and M. A. Yerry (1987), "Robust geometrically-based, automatic two-dimensional mesh generation", *Int. J. Numer. Meth. Engng.*, **24**, 1043-1078.

Blacker, T. D. and M. B. Stephenson (1991), "Paving: A new approach to automated quadrilateral mesh generation", *Int. J. Numer. Meth. Engng.*, **32**, 811-847.

Borouchaki, H. and P. Frey (1998), "Adaptive triangular-quadrilateral mesh generation", *Int. J. Numer. Meth. Engng.*, **41**, 915-934.

Canann, S. A., J. R. Tristano and M. L. Staten (1998), "An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral and tetrahedral meshes", *submitted to 7th Int. Meshing Roundtable.*

Canann, S. A., S. Muthukrishnan and B. Phillips (1994), "Topological improvement procedures for quadrilateral and triangular finite element meshes", *Proc. 3rd Int. Meshing Roundtable*, 559-588.

Canann, S. A., Y. C. Liu, A. V. Mobley (1997), "Automatic 3D surface meshing to address today's industrial needs", *Finite Elements Anal. Des.*, **25**, 185-198 (1997)

Cass, R. J., S. E. Benzley, R. J. Meyers and T. D. Blacker (1996) "Generalized 3-D Paving: An automated quadrilateral surface mesh generation algorithm", *Int. J. Numer. Meth. Engng.*, **39**, 1475-1489.

Field, D. A.(1988), "Laplacian smoothing and Delaunay triangulations", *Commun. Appl. Numer. Meth.*, **4**, 709-712.

George, P. L., F. Hecht and E. Saltel (1991), "Automatic mesh generator with specified boundary", *Comput. Meth. Appl. Mech. Engng.*, **92**, 269-288.

Hermann, L. R. (1976), "Laplacian-isoparametric grid generation scheme", *J. Eng. Mech. Div. ASCE*, **102**, EM5, 749-756

Joe, B. (1995), "Quadrilateral mesh generation in polygonal regions", *Comput. Aid. Des.*, **27**, 209-222.

Johnston, B. P., J. M. Sullivan Jr. and A. Kwasnik (1991). "Automatic conversion of triangular finite element meshes to quadrilateral elements", *Int. J. Numer. Meth. Engng.*, **31**, 67-84.

Jones, N. L. (1990), "Solid Modelling of Earth Masses for Applications in Geotechnical Engineering", *Doctoral Dissertation*, University of Texas at Austin.

Kinney, P.(1997), "CleanUp: improving quadrilateral finite element meshes", *Proc. 6th Int. Meshing Roundtable*, 449-467.

Lee, C. K. and S. H. Lo (1994), "A new scheme for the generation of a graded quadrilateral mesh", *Comput. Struct.*, **52**, 847-857.

Lo, S. H (1989), "Generating quadrilateral elements on plane and over curved surfaces", *Comput. Struct.*, **31**, 421-426.

Lo, S. H. (1985), "A new mesh generation scheme for arbitrary planar domains", *Int. J. Numer. Meth. Engng.*, **21**, 1403-1426.

Owen, S. J. and S. Saigal (1997), "Neighborhood-based element sizing control for finite element surface meshing", *Proc. 6th Int. Meshing Roundtable*, 143-154.

Sloan, S. W. (1993), "A fast algorithm for generating constrained Delaunay triangulations", *Comput. Struct.* **47**, 441-450.

Staten, M. L. and S. A. Canann (1997), "Post refinement element shape improvement for quadrilateral meshes", *Trends in Unstructured Mesh Generation*, ASME, AMD **220**, 9-16.

Talbert, J. A. and A. R. Parkinson (1991), "Development of an automatic, two dimensional finite element mesh generator using quadrilateral elements and Bezier curve boundary definition", *Int. J. Numer. Meth. Engng.*, **29**, 1551-1567.

Tam, T. K. H. and C. G. Armstrong (1991), "2D finite element mesh generation by medial axis subdivision", *Adv. Eng. Software*, **13**, 313-324.

White, D. R. and P. Kinney (1997), "Redesign of the Paving algorithm: Robustness enhancements through element by element meshing", *Proc. 6th Int. Meshing Roundtable*, 323-335.

Zhu, J. Z., O. C. Zienkiewicz, E. Hinton and J. Wu (1991), "A new approach to the development of automatic quadrilateral mesh generation", *Int. J. Numer. Meth. Engng.*, **32**, 849-866.

Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition

Joseph R. Tristano, Steven J. Owen and Scott A. Canann
ANSYS, Inc.
275 Technology Drive
Canonsburg, PA 15317
and
Carnegie Mellon University
Department of Environmental and Civil Engineering
Pittsburgh PA
{ joe.tristano, steve.owen, scott.canann }@ansys.com

Abstract

A method is presented for meshing 3D CAD surfaces in parametric space using an advancing front approach and a metric map to govern the size and shape of the triangles in the parametric space. The creation of the metric map will be discussed. The advancing front mesher generates triangles based on the metric map, stretching them in order to capture the change in parameterization of the surface. The benefits of this algorithm include better quality elements without having to do costly real space calculations.

Keywords: Triangulation, free surface meshing, Riemannian metric, CAE, finite elements

1. Introduction

1.1 Importance of work

The finite element method is a powerful tool for today's engineering community. One of the barriers to automating finite element analysis is robust automatic mesh generation on CAD surfaces. There are many manual, semi-automatic, and automatic methods available today, and all have their own advantages and drawbacks[1]. Current commercial codes tend to use either advancing front or Delaunay[2] triangulation to generate surface meshes. These methods, while very robust in two dimensions, are not as effective on 3-dimensional parametric surfaces, common in CAD models produced in industry. Direct three-dimensional extensions of Delaunay[3] and advancing front[4] techniques have been proposed, but tend to be slower and less robust. For this reason, two dimensional methods are preferred, utilizing a parametric surface mapping where appropriate. However, standard two-dimensional methods cannot be used directly since the mapping from parametric space to real space can produce distorted elements. A method is needed to place anisotropic triangles in two dimensions that will map back to isotropic triangles in three dimensions. A metric map based on the first fundamental form of the surface can be used. While a significant amount of research has gone into meshing surfaces using a metric map with Delaunay

triangulation[5], little work has been done in the area of advancing front[6,7]. This paper proposes an expanded use of the metric map for use in advancing front mesh generation.

1.2 Previous work

Previous work in this area has been done by George et al at INRIA[5,8] and others[9,10] who have investigated the use of the metric and metric map quite extensively with the use of a Delaunay kernel. Möller and Hansbo investigated using a metric to define the shape of triangles in parametric space but did not propose an algorithm to produce the mesh[6]. Cuilière[7] has also explored the use of a metric in meshing parametric surfaces using advancing front. He explored the notion of an orthogonal vector in Riemannian space (see section 3.4.3.3, equation 14) quite thoroughly.

1.3 Overview of paper

The blend of advancing front and a metric map is new since previous work with the advancing front method has been in parametric space (either directly meshing the parametric space[11]or modifying the space[12]) or directly on the 3D surface[4].

This paper describes the details of generating a finite element mesh in the parametric space of a CAD surface using a Riemannian metric map. Section 2 presents the overall algorithm. Section 3 gives the details of the algorithm. Section 4 presents some results and comparison to other methods. Section 5 draws conclusions and discusses a few areas where work is still needed.

2. Overall Algorithm

As with any advancing front method, the algorithm begins with a set of boundary loop segments, defined as the initial "front". Triangles are constructed from the front segments and grow towards the interior of the domain, "advancing the front as it proceeds". More specifically, the proposed algorithm involves the following steps:

- Discretize the boundary (section 3.2)
- Compute background mesh (section 3.3)
- Orient the front segments (section 3.4.1)
- Initialize the fronts (section 3.4.2)
- Process the fronts (section 3.4.3)

For each front:

- Check metric of nodes on front to determine appropriate method for distance calculation (section 3.4.3.1)
- Check to see if the angle between adjacent fronts is below a threshold angle. If it is, see if the formation of a triangle is possible (section 3.4.3.2)
- Determine the best location for a candidate node based on interpolation of the background mesh. (section 3.4.3.3)
- Find all nearby nodes on the current front (section 3.4.3.4)
- See if any of the nearby nodes form an acceptable triangle (section 3.4.3.5)
- If all of the above methods fail, use a brute force method to go through all the remaining nodes on the front and form the best triangle (section 3.4.3.6)
- Check boundary node normals (section 3.4.3.7)
- Form triangle

- Update the front.

The proposed algorithm can be used for both flat and curved surfaces. While curved surfaces require the use of a metric map, flat surfaces can use the identity matrix for the metric, provided an initial transformation of the three dimensional boundary nodes to the x-y plane is first accomplished.

3. Details of the Algorithm

This section will define the metric and it's uses. It will also describe the creation of the background mesh and the surface mesh that utilize the metric.

3.1 Definition of the metric

For 3D surfaces with a parameterization denoted by σ there is a metric of the tangent plane at every point P defined as:

$$[M]_P = \begin{bmatrix} E & F \\ F & G \end{bmatrix} \quad [1]$$

where

$$\begin{aligned} E &= \vec{\tau}_1 \cdot \vec{\tau}_1 \\ F &= \vec{\tau}_1 \cdot \vec{\tau}_2 \\ G &= \vec{\tau}_2 \cdot \vec{\tau}_2 \end{aligned} \quad [2]$$

and

$$\begin{aligned} \vec{\tau}_1 &= \vec{\sigma}_u'(u,v) \\ \vec{\tau}_2 &= \vec{\sigma}_v'(u,v) \end{aligned} \quad [3]$$

where $\vec{\sigma}_u'(u,v)$ and $\vec{\sigma}_v'(u,v)$ are the gradient vectors at the point P on the surface.

The 3D distance between two points along the surface, A and B, as measured from point A can be computed using their coordinates in parametric space and the metric at A:

$$\|AB\|_{M_A} = \sqrt{E_A(u_B - u_A)^2 + 2F_A(u_B - u_A)(v_B - v_A) + G_A(v_B - v_A)^2} \quad [4]$$

This function only gives the distance with respect to the metric at A. The distance as computed from both A and B is needed during the meshing process. For a *well behaved* surface the distance can be computed as the average of the distance measured from A and the distance measured from B:

$$\|AB\| = \frac{\|AB\|_{M_A} + \|AB\|_{M_B}}{2} \quad [5]$$

Well behaved implies a minimal deviation of the surface derivatives over the surface. The behavior of the surface is determined during the creation of the background mesh (section 3.3). When the surface is not *well behaved,* numerical integration must be done using a finite number of integration points, N (typically 5), along the line segment between A and B.

$$d_{3D} = \sum_{i=1}^{n} d_i$$
$$d_i = \sqrt{\overrightarrow{P_iP_{i+1}}^T \left([M]_{avg}\right)_i \overrightarrow{P_iP_{i+1}}} \qquad [6]$$
$$\left([M]_{avg}\right)_i = \frac{[M]_i + [M]_{i+1}}{2}$$

3.2 Discretization of the Boundary

The boundary loops of the surface must be discretized in such a way that well shaped elements can be created. A method such as smart sizing[13] can be used where the boundary is discretized and then refined based on the proximity and curvature of the lines in the model. This allows the mesher to capture the curvature of the surface by putting more element divisions on highly curved boundaries.

3.3 Construction of metric & size map

The metric stated above is used to compute 3D distances in a 2D, parametric domain. Since 3D sizes are stored on the 2D domain, the metric allows for the creation of distortion free triangles while performing all of the meshing in 2D parametric space. This is accomplished by using the metric to distort the triangles in the parametric space so they are distortion free when mapped back to real space.

For the sake of efficiency, a background mesh is defined to control element sizing and metric values. The background mesh consists of selected points in the parametric domain where both size and metric are known exactly. The mesher can utilize this information to interpolate local size and metric data as a function of the parametric u,v coordinates. For example:

$$size = f(u,v)$$
$$[M]_P = f(u,v) \qquad [7]$$

One of the benefits of computing the metric map and size map using the same background mesh is that all sizing information and metric information is stored in one place. A second benefit is that the metric of a point in parametric space can be determined by a simple interpolation rather than a costly evaluation of the surface followed by the computation of the metric. An additional benefit is the ability to refine the background mesh based on size and metric values at the same time on the same mesh. This, as described by Owen[14], is a fast and convenient way to compute essentially two different maps on one mesh.

The initial background mesh is a Delaunay tesselation of a subset of the domain's boundary nodes in parametric space. Boundary nodes are inserted into the background mesh only if their resulting contribution to the size map would affect it significantly.

The parametric space does not need to be *well behaved* in order for this method to work well. As a matter of fact, one of the strengths of this method is its ability to handle surfaces that do not have a *well behaved* parametric space.

Once the background mesh is generated, the 3D sizes and metrics are stored at the nodes of the 2D background mesh for later reference.

If the element size must change on the interior of the mesh due to surface curvature or due to the metric changes caused by variations in the mapping between parametric space and global space, internal nodes must be inserted into the background mesh so that an accurate representation of the size and metric can be produced.

The internal nodes in the background mesh are placed by using a quadtree decomposition of the parametric space. The quadtree is initialized by evaluating the tangent vectors at each of the points in an NxN grid. A reasonable value for N is 10. New nodes are inserted as needed at the centroid of each quadtree leaf. The quadtree leaves are refined based on the ratio the deviation of the mapping of the surface from parametric space to real space. This ratio is defined as the maximum magnitude to the minimum magnitude of the tangent vectors at the four corners of the leaf. If this ratio exceeds the maximum slope ratio (1.5 seems to work well for most surfaces) the leaf is refined. If the leaves need to be refined, the surface is not considered to be locally *well behaved.* The level of refinement is limited to a maximum number of iterations as well as by the real world length of the diagonal of the smallest quadtree leaf. If the diagonal length in real space is less than twice the minimum size then refinement of the background mesh is stopped. In addition to being inserted to fully capture transitions in the parametric mapping, nodes are also inserted to capture size transitions and surface curvature (as in Owen[14]).

Any interpolation method can be used to create and evaluate the background mesh, but Owen[14] illustrates that natural neighbor interpolation is an attractive method to use since it is C^1 continuous at the nodes of the background mesh. This is important in representing the metric, since is undesirable to have discontinuities in the metric map. The method also reduces the amount of "banding" i.e., bands of unnecessarily small elements.

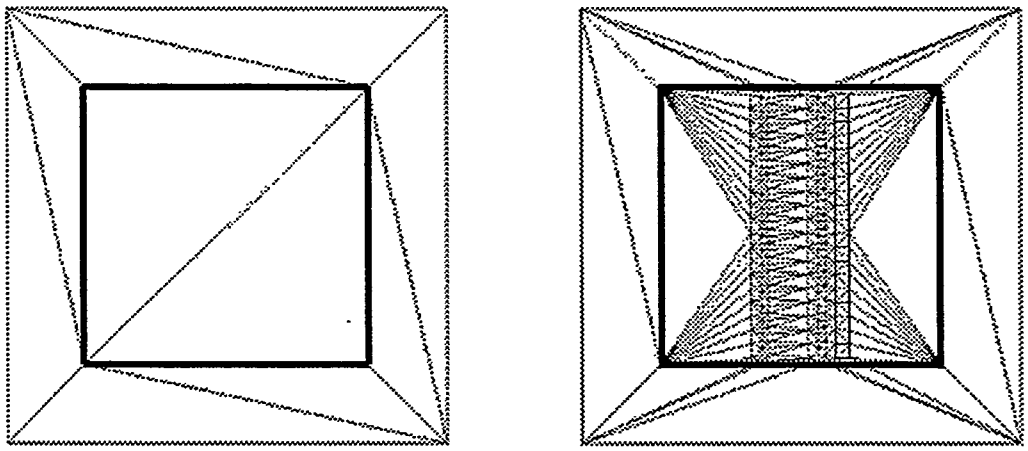


Figure 1 Background mesh before and after refinement

3.4 Advancing front algorithm using the metric map

After the background mesh has been created, the meshing process can begin. While the basic algorithm was outlined in Section 2, the details will be presented here.

3.4.1 Orient front segments

A set of closed boundary loops (closed set of singly connected edges) made up of the initial front serves as input to the advancing front mesher. Boundary loops are oriented such that the exterior loops follow a counter-clockwise direction and the interior loops follow a clockwise direction.

3.4.2 Initialize the fronts

The first step after the boundary has been discretized and the size/metric map has been computed is the initialization of the front:

- Sizes and metrics are interpolated from the background mesh and stored with the boundary nodes. The 3D front length is computed and stored in the front data structure using the distance calculations from Section 3.1.
- For each edge on the front, the equation of the line using the u,v coordinates of the edge's nodes is computed for intersection calculations to be performed later.
- Experience has shown that meshing smaller fronts first can be advantageous. To accomplish this, the fronts are hashed according to their 3D size. The fronts are stored in a collection of bins; each one holding a specified range of sizes. Bin sizes are defined logarithmically, where the front size range for a bin containing small fronts is narrower than for larger fronts. During the meshing process, fronts are first processed from the bin containing the smallest fronts. It was found that sorting based on 3D size is better than 2D size because of problems found with mapping the mesh back to 3D space where the parametric space is highly distorted. Sorting based on 3D distances also helps to better capture size transitions in the mesh.
- The final step in front initialization is to check for intersecting boundary segments. This step is performed in order to check for erroneous or poor input. Each front segment is checked against any non-adjacent front. If any of the boundary fronts intersect, the mesher returns an error code.

3.4.3 Process the fronts

The fronts are now processed from the smallest 3D sized front to the largest, as described in the following sections.

3.4.3.1 Check metric of nodes on the front

Since distance measurement in Riemannian space varies from one location to another throughout the parametric space, special care must be taken where large size transitions occur, or where the parametric space is locally not well behaved.

For each front, the mesher must ensure that the metrics at its end nodes do not exceed the maximum slope ratio defined during the creation of the background mesh. If the ratio is exceeded, distance calculations must be integrated to better approximate the true 3D distance. (Note that in further discussions of distance calculations the ratio of the metrics must be checked in order to determine which distance calculation should be used, i.e. averaging (equation 5) or integrating (equation 6)).

3.4.3.2 Check if the angle between adjacent fronts is below threshold angle

The first step in processing a front is checking the 3D angle it makes between its two adjacent fronts. Using the 3D lengths of the three sides, the law of cosines is used to compute the angle at the desired node as in the following:

$$\theta_A = \cos^{-1}\left(\frac{\|CA\|_{3D}^{\ 2} + \|AB\|_{3D}^{\ 2} - \|BC\|_{3D}^{\ 2}}{2\|CA\|_{3D}\|AB\|_{3D}}\right) \qquad [8]$$

If the angle is below the threshold angle of 75 degrees as shown in Figure 2, the triangle formed from A, B and C becomes a candidate triangle. Before forming triangle ABC, segment BC must first be checked to ensure it does not intersect any other fronts. This can be done as outlined in section 3.4.3.5.

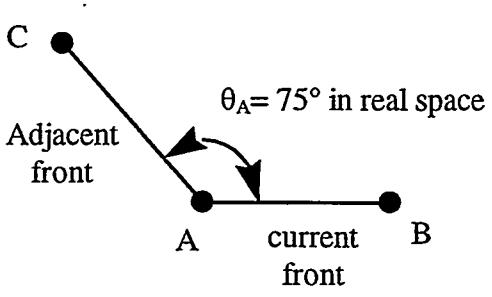


Figure 2 Small angle triangle creation

To ensure that high aspect ratio triangles are not created, an additional check is made before the triangle is created. If the aspect ratio of edge length to front length is more than twice the user controllable growth ratio, g_r, (usually 2.0) the adjacent edge will be split.

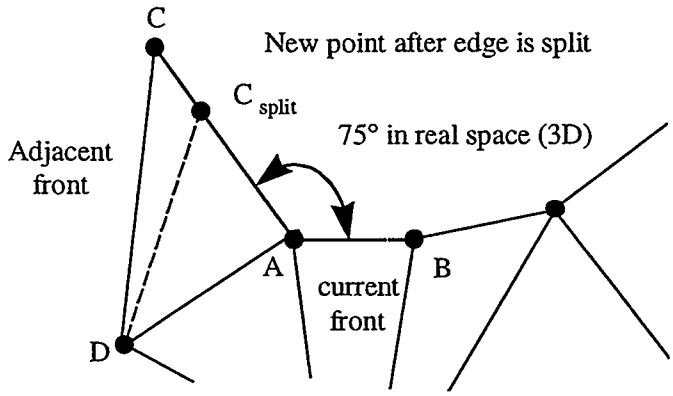


Figure 3 Splitting of triangle because of aspect ratio

For example, in Figure 3, edge AB is the current front and edge AC is an adjacent front with an angle less than 75°. Since edge AC exceeds the maximum transition ratio, the adjacent front AC is split.

$$\frac{\|CA\|_{3D}}{\|AB\|_{3D}} \geq g_r \qquad [9]$$

Edge AC is split at location C_{split}, forming a new node, where the length of the new edge AC_{split} is defined as:

$$\|AC_{split}\|_{3D} = \|AC\|_{3D} \frac{\psi_A}{\psi_A + \psi_C} \qquad [10]$$

where ψ_A, and ψ_C, are the desired sizes at A and C.

The AC is split at C_{split} in order to create a division that best matches the desired sizes at A and C.

The new triangles CDC_{split} and DAC_{split} are then formed from triangle ACD. The triangle ABC_{split} is then created and the front is updated.

3.4.3.3 Determining the best location for a new node

If a triangle cannot be created using the adjacent fronts, as described above, the location of a candidate node, N_c, that would form an ideal triangle is determined. All of the nodes on the current front, N_{Fi}, that are within a specified 3D distance from N_c are found and sorted by their distance to N_c (see section 3.4.3.4 for details). The closest node that forms a valid triangle with the current front is selected. The following describes this process in more detail.

To find the ideal size, ψ_{3D} of the new triangle, T_n, from front AB the following may be used:

$$\psi_{3D} = \min\left(\frac{\psi_A + \psi_B}{2}, \|AB\|_{3D} \cdot g_r\right), g_r > 1 \qquad [11]$$

where ψ_A, ψ_B are the 3D element sizes interpolated from the background mesh at A and B respectively and g_r is a user defined maximum growth ratio.

For flat, non-parametric surfaces, the location of N_c can be constructed by forming an isosceles triangle with front AB, where edge length $\|AN_c\|_{3D} = \|BN_c\|_{3D} = \psi_{3D}$, and the height, h_{3D}, of T_n is defined as:

$$h_{3D} = \sqrt{\psi_{3D}{}^2 - \frac{\|AB\|_{3D}{}^2}{4}} \qquad [12]$$

Since most parametric mappings are not uniform, $\|AN_c\|_{2D} \neq \|BN_c\|_{2D}$, even though $\|AN_c\|_{3D} = \|BN_c\|_{3D}$. However, N_c can be located by computing an equivalent h_{2D} and normal vector, $\bar{N}_{2D}$ to front AB, as shown in Figure 4. For example let:

$$\bar{m}_{AB} = \frac{\bar{B}_{2D} + \bar{A}_{2D}}{2} \qquad [13]$$

$$\bar{N}_{2D} = \begin{Bmatrix} u_{N_{2D}} \\ v_{N_{2D}} \end{Bmatrix} = \begin{bmatrix} -(F_{m_{AB}} + G_{m_{AB}}) \\ E_{m_{AB}} + F_{m_{AB}} \end{bmatrix} \bar{V}_{AB} \qquad [14]$$

where $\bar{V}_{AB}$ is a unit vector pointing from A to B and $\bar{m}_{AB}$ is the midpoint of AB.

Finally, to locate N_c in parametric space, N_{c2D}, the following can be used:

$$\bar{N}_{c2D} = \bar{m}_{AB} + \bar{N}_{2D} \cdot h_{2D} \qquad [15]$$

where

$$h_{2D} = \frac{h_{3D}}{\sqrt{E_{m_{AB}} u_{N_{2D}}{}^2 + 2F_{m_{AB}} u_{N_{2D}} v_{N_{2D}} + G_{m_{AB}} v_{N_{2D}}{}^2}} \qquad [16]$$

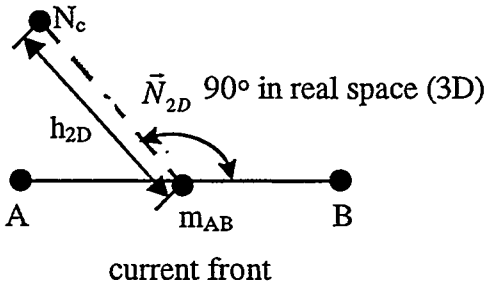


Figure 4 Locating ideal node location

3.4.3.4 Find nearby nodes

It is not always necessary to create a new node at N_c. In many cases, there will be an existing node, N_F, that is close enough to fulfill the local size requirements. A search radius, r_{3D} defined as:

$$r_{3D} = \min(\psi_c, \psi_{3D}) \cdot s_f,\ 0 < sf < 1 \qquad [17]$$

where ψ_c is the interpolated element size at N_c, and s_f is a shrink factor limiting the radius of acceptable nodes, thereby limiting the size transition in the mesh and reducing the number of distance calculations. A typical range for s_f is between .8 and .4.

Since the search radius is a 3D distance, an ellipse based on the metric at N_c must be used in parametric space to calculate a bounding box to reduce the number of distance calculations. The major radius of this ellipse is used as the width of the bounding box.

$$r_{2D} = \frac{r_{3D}}{\sqrt{E_{N_c} u_{N_c}{}^2 + 2F_{N_c} u_{N_c} v_{N_c} + G_{N_c} v_{N_c}{}^2}} \qquad [18]$$

Equation 18 is in the same form as an ellipse in polar coordinates centered about the origin. It can then be transformed into the general form of an ellipse in Cartesian coordinates.

$$Ax^2 + By^2 + Cxy + F = 0 \qquad [19]$$

where:

$$A = E_{N_c}, B = G_{N_c}, C = 2F_{N_c}, F = r_{3D}{}^2, x = r_{2D}{}^2 u_{N_c}, y = r_{2D}{}^2 v_{N_c} \qquad [20]$$

By rotating the ellipse to align with the Cartesian coordinate axis, equation 19 may be reduced and the major radius defined as:

$$a = \frac{1}{\sqrt{\min(A_{rot}, B_{rot})}} \qquad [21]$$

where

$$A_{rot} = \left(A\cos^2\theta + B\sin^2\theta + \frac{C\sin 2\theta}{2} \right) F^{-1}$$
$$B_{rot} = \left(B\cos^2\theta + A\sin^2\theta - \frac{C\sin 2\theta}{2} \right) F^{-1} \qquad [22]$$
where
$$\cot\theta = \frac{B - A}{C}$$

The major radius can now be used as the parametric space bounding box width.

A bounding box with width 2a centered at N_c can be used to quickly filter nodes, N_{Fi}, on the front that do not fall within the ellipse. The distance equations mentioned in section 3.1 can be used to determine if they are actually within the search radius while not violating the maximum growth ratio ,g_r.

3.4.3.5 Validity checks for candidate Triangles

There are three tests that a triangle must pass to be accepted: zero area test, internal node test, and front intersection test.

The first triangle validity check determines if the triangle has zero area in parametric space. Zero area triangles cannot be created.

If T_n passes the zero area test, it must be checked for nodes interior to it or nodes too close to the boundary edges of nearby triangles. The nodes close to the triangle are checked with this same test, referred to as the interior node test.

The area (barycentic) coordinates of a node on the front, N_{Fi}, which lies within the bounding box of the current front are computed with respect to T_n (Figure 5). If any of these coordinates are less than a specified empirical tolerance, N_c must be tested again using the approximate 3D area coordinates. The 3D area of a triangle,γ_{3D}, can be defined from Heron's formula as.

$$\gamma_{3D} = \sqrt{s(s-\|AB\|_{3D})(s-\|BC\|_{3D})(s-\|CA\|_{3D})} \qquad [23]$$

where

$$s = \frac{\|AB\|_{3D} + \|BC\|_{3D} + \|CA\|_{3D}}{2} \qquad [24]$$

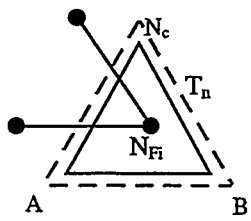


Figure 5 Test for node inside triangle

If any of these area coordinates are algebraically less than an empirical tolerance (currently set to -0.0154321012) then the triangle passes the interior node test.

If the triangle passes the interior node test, the triangle then goes through the front intersection test. This test checks that the triangle does not intersect any of the current fronts. This intersection test is performed in parametric space.

3.4.3.6 Brute force method

If any of the candidate nodes (nearby nodes plus the ideal node N_c), N_{ci}, fail to create a valid triangle, then the brute force method presented in this section is used to create a triangle. The brute force method consists of looping through all of the nodes on the front and finding the best shaped triangle. A distortion metric[15], α, is used to determine the quality of the triangle. For flat non-parametric surfaces, α is defined as:

$$\alpha = 2\sqrt{3}\frac{\overrightarrow{N_cA}\times\overrightarrow{N_cB}\cdot\hat{n}}{\|N_cA\|^2+\|AB\|^2+\|BN_c\|^2} \quad [25]$$

The directions vectors $\overrightarrow{N_cA}$ and $\overrightarrow{N_cB}$ are not known on the parametric surface, so the area squared, γ_{3D}^2, must be computed using equation 23 above. Using this, the distortion metric in Riemannian space, α_{3D}, can be defined as:

$$\alpha_{3D} = 2\sqrt{3}\frac{2\gamma_{3D}^{\ 2}}{\|N_cA\|_{3D}^{\ 2}+\|AB\|_{3D}^{\ 2}+\|BN_c\|_{3D}^{\ 2}} \quad [26]$$

The distortion metric alone is not always the best method to create the new triangle because size criterion may be violated even though one triangle may be of better quality than another. It was found that the best triangle is created by penalizing the distortion metric of the triangle based on a function of the ratio of desired triangle size and longest triangle edge length. This function is illustrated below:

$$penalty = \begin{cases} xe^{(1-x)} - 0.4\sin\pi x & 0 < x < 1 \\ 0.5(xe^{(1-x)} - \frac{1}{9}x + \frac{10}{9} & 1 \le x < \infty \end{cases} \quad [27]$$

where

$$x = \frac{(\psi_A+\psi_B)/2}{\max(\|AN_c\|_{3D}, \|BN_c\|_{3D})} \quad [28]$$

where ψ_A and ψ_B are the desired sizes at A and B.

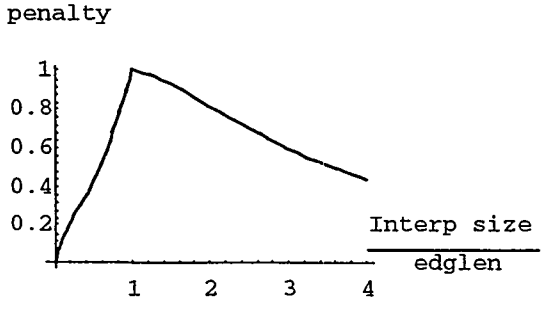


Figure 6 Penalty function for distortion metric

The new alpha then becomes:

$$\alpha_{new} = \alpha_{3D} \cdot penalty \qquad [29]$$

3.4.3.7 Checking boundary normals

There are some cases where degenerate surfaces such as cones have highly distorted parametric space. In such cases it is necessary to check the angle spanned by a triangle connected to boundary nodes to prevent the element from "flattening" out (Figure 7).

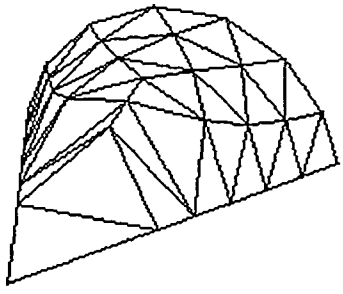


Figure 7 Flattened out cone tip

This check is done only when two or three of the triangle's nodes lie on mutually exclusive boundary curves. The angle spanned by the edge that cuts across the domain from one boundary to another is checked by computing the angle between the surface normals of the boundary nodes. If the computed angle is below the user controllable maximum spanning angle (usually between 5 and 60 degrees), the triangle is then split at its midpoint along the edge that cuts across the domain.

For example, Let nodes A and B of triangle ABC (Figure 8) lie on different boundary curves. Let the surface normals at nodes A and B be $\vec{N}_A$ and $\vec{N}_B$. If the dot product of $\vec{N}_A$ and $\vec{N}_B$ exceeds the maximum spanning angle then triangle ABC will be split at it's 3D midpoint P along edge AB.

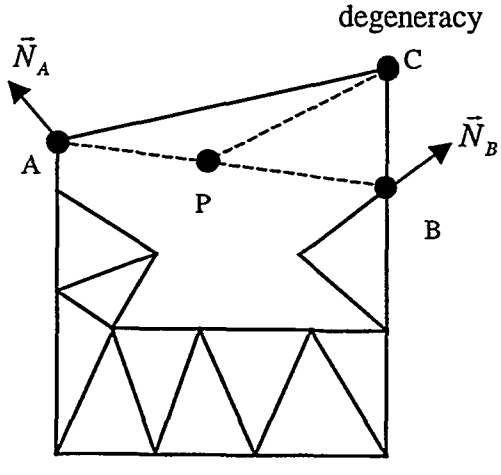


Figure 8 Splitting of flat cone tip in parametric space

4. Results and Comparisons

In this section, the results of the Riemannian space advancing front mesher are compared against the warped parametric space and direct 3D meshers available in the ANSYS program. The warped parametric space mesher[12] is a mesher in the ANSYS program that meshes parametric surfaces in a different manner. This method reparametrizes the surface selectively evaluating surface derivatives (Δu, Δv) over the domain and adjusting local u,v values to hold the magnitude of Δu, Δv roughly constant. Times presented in this section incorporate the total meshing time after boundary discretization. This includes, projecting the 3D boundary nodes to 2D parametric space, background mesh creation, advancing front meshing, cleanup and smoothing, mapping to 3D, and storage of elements to the ANSYS database.

Figures 9(a,b,c) and Table 1 illustrate the advantages of meshing parametric surfaces using a Riemannian surface definition rather than changing the parametric space. The poorly parametrized surface has surface derivatives that are not orthogonal. This phenomena yields poorly shaped, stretched triangles when meshed with the warped parametric space mesher. However, when the surface is meshed using the Riemannian space mesher the resulting triangles are well shaped.

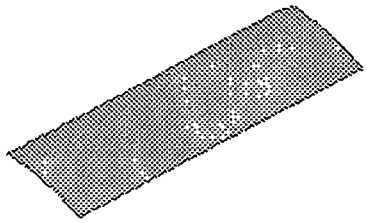


Figure 9a Poorly parameterized surface

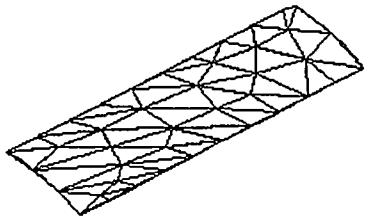


Figure 9b With warped parametric space

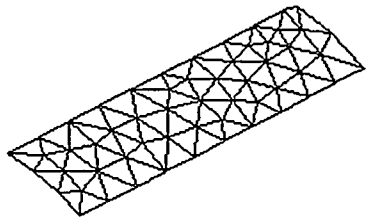


Figure 9c With Riemannian space mesher

Table 1 Distortion metrics of surface mesh

Mesher	Triangles	min α	avg. α
Riemannian space	64	0.778	0.961
warped parametric space	50	0.320	0.743

Figure 10 and Table 2 illustrate the drastic speed improvements that the Riemannian space mesher has over the direct 3D mesher with both meshers yielding meshes of equivalent quality.

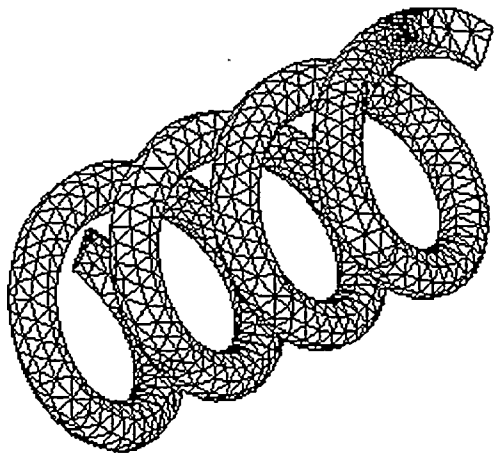


Figure 10 Spring

Table 2 Spring statistics

Mesher	Triangles	min α	Avg. α	time	elements/sec
Riemannian space	3336	0.489	0.923	9.93	335.95
warped parametric space	4356	0.307	0.922	21.89	198.99
direct 3D	3506	0.481	0.927	64.38	54.45

Figures 11(a,b,c) and Table 3 illustrate the overall quality and speed improvements over the warped parametric space and direct 3D meshers for a general CAD surface. The stretched triangles in the warped parametric space mesh are caused by non-orthogonal surface derivatives. The Riemannian space mesher resolves those problems. It also resolves the problems of costly real space calculations done by the direct 3D mesher.

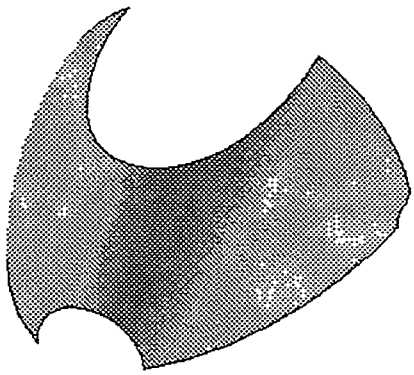


Figure 11a CAD Surface

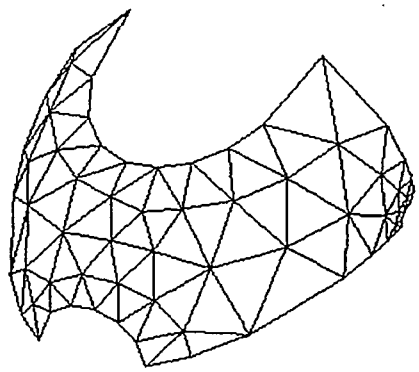


Figure 11b With Riemannian space mesher

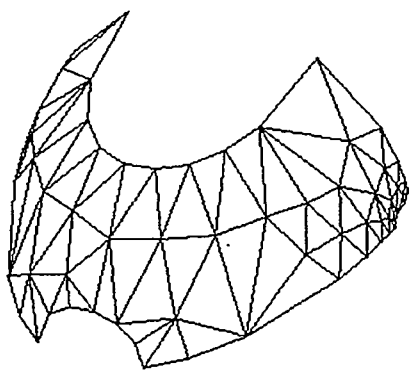


Figure 11c With warped parametric space

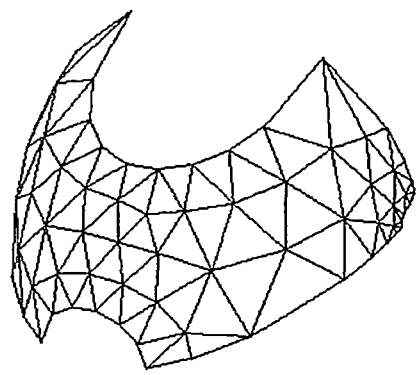


Figure 11d With direct 3D mesher

Table 3 CAD surface statistics

Mesher	Triangles	min α	Avg. α	time	elements/sec
Riemannian space	121	0.650	0.910	0.36	336.11
warped parametric space	115	0.409	0.821	0.29	396.55
direct 3D	125	0.459	0.907	0.74	168.91

5. Conclusion

A method for meshing 3D parametric surfaces using the advancing front method with a Riemannian surface definition was presented. The details of the creation of the metric map used to determine the amount of distortion of the elements in parametric space were given along with the details of an advancing front algorithm that utilizes the metric map.

Meshing surfaces using an advancing front with a Riemannian surface definition proves to be a valuable technique for meshing surfaces common in CAE. This method overcomes anomalies found in the warped parametric space and direct 3D methods currently used in the ANSYS program. Well shaped triangles are produced by this method. Any of three of the methods compared robust and capable of creating high quality meshes for most geometries. However, having all three meshers available in the ANSYS program greatly increases the likelihood of successfully meshing any arbitrary collection of surfaces and hence fully automated constrained tetrahedral meshing. Future areas of work may include the combination of a warped parametric space and Riemannian space mesher in order to create better mappings for high aspect ratio surfaces.

6. References

1 Ho-Le, K., Finite element mesh generation methods: a review and classification, *Computer Aided Design*, Vol 20(1) , 27-38, 1988.

2 Chen, H., and J. Bishop, Delaunay Triangulation for Curved Surfaces, *6th International Meshing Roundtable Proceedings*, pp.115-127, 1997.

3 Chew, Paul L., Guaranteed-Quality Mesh Generation for Curved Surfaces, *9th Annual Computational Geometry*, Vol 73, 1993.

4 Lohner,R., Extensions and Improvements of the Advancing Front Grid Generation Technique, *Communications in Numerical Methods in Engineering*, John Wiley & Sons, Ltd, Vol 12, pp.683-702, 1996.

5 George P. L., and H. Borouchaki, *Delaunay Triangulation and Meshing Application to Finite Elements*, Editions HERMES, Paris, 1998.

6 Moller, P., and P Hansbo, On advancing front mesh generation in three dimensions, *IJNME*, Vol. 38, pp.3551-3569, 1995.

7 Cuilière, J. C., An adaptive method for the automatic triangulation of 3D parametric surfaces, *Computer Aided Design*, Vol 30(2), pp.139-149, 1998.

8 Castro Díaz, M. J., and F. Hect, Anisotropic Surface Mesh Generation, *INRIA Research Report*, N° 2672, 1995.

9 Shimada, K., Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles, *6th International Meshing Roundtable Proceedings*, pp.63-74, 1996.

10 Bossen, F. J., P. S. Heckbert, A Pliant Method fo Anisotropic Mesh Generation, *5th International Meshing Roundtable Proceedings*, pp.375-390, 1997.

11 Lo, S.H., A new mesh generation scheme for arbitrary planar domains, *IJNME*, Vol. 21, pp1403-1426, 1985.

12 Canann, S. A., Y. C. Liu and A. V. Mobley, Automatic 3D Surface Meshing to Address Today's Industrial Needs, *Finite Elements Anal. Des.*, Vol 25, pp.185-198, 1997.

13 Cunha, A., S. A. Canann and S. Saigal, Automatic Boundary Sizing for 2D and 3D Meshes, *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, ASME, pp.65-72, July 1997.

14 Owen, S., Neighborhood-based element sizing control for finite element surface meshing, *6th International Meshing Roundtable Proceedings*, pp..43-154, 1997.

15 Lo, S.H. and C.K. Lee, On Using Meshes of Mixed Element Types in Adaptive Finite Element Analysis, *Finite Elements in Analysis and Design*, Elsevier, Vol 11, pp.307-336, 1992.

Mesh Quality

WINSLOW SMOOTHING ON TWO-DIMENSIONAL UNSTRUCTURED MESHES

PATRICK M. KNUPP
PARALLEL COMPUTING SCIENCES DEPARTMENT
SANDIA NATIONAL LABORATORIES*
M/S 0441, P.O. BOX 5800
ALBUQUERQUE, NM 87185-0441
PKNUPP@SANDIA.GOV

Abstract. The Winslow equations from structured elliptic grid generation are adapted to smoothing of two-dimensional unstructured meshes using a finite difference approach. Winslow smoothing on unstructured quadrilateral meshes results in mesh folding less frequently than with traditional Laplacian smoothing.

Key words. unstructured grid generation, quadrilateral meshing, elliptic smoothing, winslow equations

AMS subject classification. 65M50

1. Introduction

The Winslow elliptic smoother has been used for many years in two-dimensional structured mesh generation because of its connection to harmonic maps between manifolds for which a one-to-one guarantee is proven. The smoother is based on solving the well-known partial differential equations

$$g_{22}\, x_{\xi\xi} - 2\, g_{12}\, x_{\xi\eta} + g_{11}\, x_{\eta\eta} = 0, \tag{1}$$

$$g_{22}\, y_{\xi\xi} - 2\, g_{12}\, y_{\xi\eta} + g_{11}\, y_{\eta\eta} = 0 \tag{2}$$

as a boundary-value problem [11], [10]. The equations are obtained by requiring the logical variables ξ and η to be harmonic functions and by interchanging the dependent and independent variables in the corresponding Laplace equations.

The purpose of this paper is not to compare Winslow and Laplacian smoothing but rather to present a logical extension of Winslow smoothing to unstructured meshes. Many comparisons between Winslow smoothing and Laplacian smoothing on structured meshes have been performed and there is a widely held concensus that the former is the smoother of choice, mainly due to its resistance to grid folding. Despite the effectiveness of this mesh generator in structured meshing, few attempts have been made to extend its applicability to the realm of unstructured meshes. Instead, Laplacian smoothing predominates on unstructured meshes due to its generality and to its ease of implementation. It is the author's belief that the additional work of implementing Winslow smoothing is worth the effort to achieve robustness against mesh folding.

In Laplacian smoothing, node positions are the average of positions of the M neighboring nodes:

$$\mathbf{x}_n = \frac{1}{M} \sum_{m=0}^{M-1} \mathbf{x}_m \tag{3}$$

* SANDIA IS A MULTIPROGRAM LABORATORY OPERATED BY SANDIA CORPORATION, A LOCKHEED MARTIN COMPANY, FOR THE UNITED STATES DEPARTMENT OF ENERGY UNDER CONTRACT DE-ACO4-94AL85000.

Laplacian smoothing on structured meshes can be shown to arise from solving the following pair of partial differential equations [6]:

$$x_{\xi\xi} + x_{\eta\eta} = 0, \tag{4}$$

$$y_{\xi\xi} + y_{\eta\eta} = 0. \tag{5}$$

Although Laplacian smoothing is easy to implement, it's usefulness is limited by the fact that it sometimes results in mesh folding/spillover. No guarantee against folding can be constructed for equations (4) and (5). The fact that Laplacian smoothing on structured meshes can be extended to unstructured meshes strongly suggests that it should be possible to extend Winslow smoothing to unstructured meshes as well.

In his original paper [11], Winslow solved equations (1) and (2) on a structured triangular mesh having six-valent nodes. Others soon extended the procedure to structured quadrilateral meshes having four-valent nodes [10]. To extend Winslow to unstructured meshes clearly requires letting go of the idea of a global mapping. This point was grasped by mesh researchers such as Tipton [9], Allievi [1], and Hagemeier [4]. In spite of their insight, Winslow remains largely unused by the unstructured meshing community although some ad-hoc attempts at extending Winslow have been made [5].

Winslow smoothing on unstructured *quadrilateral* meshes is emphasized due to the author's present association with Sandia's CUBIT project. However, an extension to smoothing of unstructured triangular meshes is briefly outlined.

In section 2 of this paper, equations are derived that give the local discrete approximation of the Winslow equations on an unstructured mesh. Section 3 presents implementation issues, demonstrates results on several CUBIT test problems, and discusses the extension to triangular meshes. Conclusions are given in section 4.

2. Winslow local discretization on an unstructured quadrilateral mesh.

The natural approach to solving (1) and (2) on unstructured meshes is to apply the finite-element method (FEM). Tipton does this for 3D Winslow in a proprietary DOE paper [9]. An alternative approach outlined in this paper is to use finite differences, expanding derivatives about each mesh node in a Taylor Series and solving the resulting non-linear coupled system of equations. This is not meant to suggest that the finite difference approach given here is superior to a FEM approach. The choice merely reflects the fact that the author is more familiar with the former method.

For unstructured meshes, the most commonly used smoother is Laplacian Smoothing. For a given mesh node $\mathbf{x}_n$ a discrete operator D_n from $R^3 \rightarrow R^3$ can be formulated

$$D_n\mathbf{x} = M\,\mathbf{x}_n - \sum_{m=0}^{M-1} \mathbf{x}_m \tag{6}$$

where the sum is over the M neighboring nodes. To smooth the mesh, one iterates on the set of equations $D_n\mathbf{x} = 0$ until some tolerance is satisfied. A similar operator is sought such that in the limit of small mesh size, (1) and (2) will be locally approximated at each node of the mesh.

To derive a discrete operator for (1) and (2) using finite differences, the two first derivatives and the three second derivatives at each node $\mathbf{x}_n$ with valency $M \geq 3$ must be approximated. First consider only the "edge" nodes, i.e., the nodes $\mathbf{x}_m$ adjacent

to $\mathbf{x}_n$. Let a local, discrete, uniform logical space (ξ_m, η_m) be given by

$$\xi_m = \cos\theta_m \tag{7}$$

$$\eta_m = \sin\theta_m \tag{8}$$

where

$$\theta_m = \frac{2\pi m}{M} \tag{9}$$

with $m = 0, 1, ...M-1$. The following well-known formula involving non-negative integral powers of the roots of unity is applied:

$$\sum_{m=0}^{M-1} (\exp^{i\theta_m})^k = \begin{cases} M & \text{k = 0, M, 2M, ...} \\ 0 & \text{else} \end{cases} \tag{10}$$

Explicit results for this formula are given for $k = 1, 2, 3$, and 4 in Appendix I.

Assume that there exists a smooth, C^∞ function $f(\xi, \eta)$ on the local logical space and that one can approximate $f(\xi_m, \eta_m)$ about $f(0,0)$ by the Taylor Series Expansion:

$$\begin{aligned} f(\cos\theta_m, \sin\theta_m) &= \sum_{\ell=1}^{L-1} \frac{1}{(\ell-1)!} (\cos\theta_m \frac{\partial}{\partial\xi} + \sin\theta_m \frac{\partial}{\partial\eta})^{\ell-1} f \mid_{(0,0)} \\ &+ \frac{1}{L!} (\cos\theta_m \frac{\partial}{\partial\xi} + \sin\theta_m \frac{\partial}{\partial\eta})^{L} f \mid_{(\tau\cos\theta_m, \tau\sin\theta_m)} \end{aligned} \tag{11}$$

for some $0 \leq \tau \leq 1$.

To arrive at expressions for the first and second derivatives of f at $(0,0)$, multiply the Taylor Series expansion above by various combinations of $\cos\theta_m$ and $\sin\theta_m$, sum from $m = 0$ to $m = M-1$, and use the identities in Appendix I. Consider only $k \leq 4$ in order to neglect higher-order terms involving third and higher derivatives. For $M \geq 5$ the resulting approximations are:

$$f_\xi = \frac{2}{M} \sum_{m=0}^{M-1} (f_m - f_0) \cos\theta_m \tag{12}$$

$$f_\eta = \frac{2}{M} \sum_{m=0}^{M-1} (f_m - f_0) \sin\theta_m \tag{13}$$

$$f_{\xi\xi} = \frac{2}{M} \sum_{m=0}^{M-1} (f_m - f_0) (4\cos^2\theta_m - 1) \tag{14}$$

$$f_{\xi\eta} = \frac{8}{M} \sum_{m=0}^{M-1} (f_m - f_0) \cos\theta_m \sin\theta_m \tag{15}$$

$$f_{\eta\eta} = \frac{2}{M} \sum_{m=0}^{M-1} (f_m - f_0) (4\sin^2\theta_m - 1) \tag{16}$$

For $M = 3$ and 4 there is not enough information in the "edge" nodes to uniquely determine some of the required derivatives. For example, when $M = 4$, $f_{\xi\eta}$ cannot be approximated by the "edge" nodes and so the "diagonal" nodes (those four nodes $\hat{\mathbf{x}}_m$ lying on the opposite corners of the four quadrilaterals) are included, increasing the

valence to eight. To construct a logical space for the diagonal nodes assume that the logical quadrilaterals form rhombii. Then $\xi_m = L_M \cos\hat{\theta}_m$, $\eta_m = L_M \sin\hat{\theta}_m$, where $L_M^2 = 2\,(1 + \cos 2\pi/M)$ and

$$\hat{\theta}_m = \frac{2\pi(m+\frac{1}{2})}{M} \tag{17}$$

The motivation for this construction is, of course, to get the usual second-order accurate centered-difference approximation to $f_{\xi\eta}$ when $M = 4$. The result for $M = 4$ is then

$$f_\xi = \frac{2}{M}\sum_{m=0}^{M-1}(f_m - f_0)\cos\theta_m \tag{18}$$

$$f_\eta = \frac{2}{M}\sum_{m=0}^{M-1}(f_m - f_0)\sin\theta_m \tag{19}$$

$$f_{\xi\xi} = \frac{4}{M}\sum_{m=0}^{M-1}(f_m - f_0)\cos^2\theta_m \tag{20}$$

$$f_{\xi\eta} = \frac{2}{M}\sum_{m=0}^{M-1}(\hat{f}_m - \hat{f}_0)\cos\hat{\theta}_m \sin\hat{\theta}_m \tag{21}$$

$$f_{\eta\eta} = \frac{4}{M}\sum_{m=0}^{M-1}(f_m - f_0)\sin^2\theta_m \tag{22}$$

where the "hat" notation refers to evaluations on the diagonal nodes.

For $M = 3$, it is impossible to get the derivatives $f_{\xi\xi}$ and $f_{\eta\eta}$ if only the "edge" nodes are used. For this case, use the diagonal nodes of the three adjacent quadrilaterals to augment the nodes to a set of six nodes. Then equations (12)-(16) can be applied but with $M = 6$. The result is exactly the set of formulas used by Winslow [11]. In summary, the first and second derivatives of f may be approximated on an unstructured, quadrilateral mesh. Low-valent nodes with $M = 3$ and $M = 4$ require use of the "diagonal" nodes, while higher valent nodes can be approximated without using the "diagonal" nodes.

The discrete Winslow operator can now be constructed. Let

$$D_n\mathbf{x} = G_{22}\,D_{\xi\xi}\mathbf{x}_n - 2\,G_{12}\,D_{\xi\eta}\mathbf{x}_n + G_{11}\,D_{\eta\eta}\mathbf{x}_n \tag{23}$$

with

$$G_{11} = D_\xi\mathbf{x}_n \cdot D_\xi\mathbf{x}_n \tag{24}$$

$$G_{12} = D_\xi\mathbf{x}_n \cdot D_\eta\mathbf{x}_n \tag{25}$$

$$G_{22} = D_\eta\mathbf{x}_n \cdot D_\eta\mathbf{x}_n \tag{26}$$

and

$$D_\xi\mathbf{x}_n = \frac{2}{M}\sum_{m=0}^{M-1}(\mathbf{x}_m - \mathbf{x}_n)\cos\theta_m \tag{27}$$

$$D_\eta\mathbf{x}_n = \frac{2}{M}\sum_{m=0}^{M-1}(\mathbf{x}_m - \mathbf{x}_n)\sin\theta_m \tag{28}$$

The second derivatives are constructed similarly, but depend on whether or not $M = 4$.

3. Implementation and Applications.

The method outlined in the previous section was implemented in C++ within the CUBIT code [2]. Because CUBIT uses solid model geometry, the equations were solved using vectors in R^3 and the nodes moved to the "owning" surfaces. One thus solves

$$g_{22}\,\mathbf{x}_{\xi\xi} - 2\,g_{12}\,\mathbf{x}_{\xi\eta} + g_{11}\,\mathbf{x}_{\eta\eta} = 0 \tag{29}$$

where $\mathbf{x} \in R^3$.

In order to correctly apply the formula of the previous section one must ensure that the adjacent nodes are consistently ordered (either clockwise or counterclockwise) with respect to the owning surface. A special routine within CUBIT was written to do this. To make this approach compatible with CUBIT's paving/cleanup algorithm [3], two-valent nodes were specially handled by doing Laplacian smoothing.

Figure 1 shows the results of Winslow smoothing on an annular surface in which the upper left-hand side has been meshed with a structured mesh and the lower right-hand side meshed using paving. The large cell aspect-ratios on the smoothed structured mesh may strike some readers as not particularly good. Recall, however, that mesh quality is physics-dependent, as well as geometry-dependent, i.e., the application determines whether or not the mesh is acceptable. If small aspect-ratios are wanted, then the Winslow-smoothed paved mesh on the lower right of Figure 1 is more appropriate. Readers familiar with Winslow smoothing on structured meshes will consider the result in the upper left to be a good mesh compared to that produced by Laplacian smoothing - the latter is well-known to produce badly folded meshes if a structured mesh is used on a sufficiently stretched version of the geometry [7]. A paved mesh on a toroidal surface was Winslow-smoothed in Figure 2. These examples alone do not demonstrate the robustness of Winslow smoothing against folding, but a year's experience with this smoother in CUBIT has demonstrated that the property exhibited on structured meshes has been sucessfully carried over to unstructured quadrilateral meshes. If mesh density is too coarse, truncation error can lead to Winslow producing folded meshes despite the existence of a guarantee for the continuum global mapping [7]. The author has observed this on very coarse paved meshes and also at the center of CUBIT's Circle primitive. Winslow smoothing on structured meshes is derived by solving an elliptic boundary value problem. As a result, the Winslow guarantee does not extend to the mesh at the boundary and folded cells on the boundary are not precluded. In the present implementation, boundary nodes are not smoothed.

Although the author has not yet implemented Winslow for unstructured triangular meshes, the method in section 2 can be modified for this case. The formulas (12)-(16) in that section will apply provided any node in the triangular mesh has a valence of 5 or greater (this includes Winslow's six-valent case of a regular triangular mesh). If a node in the triangular mesh is 3- or 4-valent, valence can be increased by using nodes opposite the edges of the triangles that are opposite the smooth node. The only case for which this cannot be done is if the node is near the boundary; then the opposite nodes may not exist. For certain degenerate cases, such as that of a 3-valent node adjacent to the corner of the domain, one may not be able to sufficently increase the valence by grabbing adjacent nodes. Laplacian smoothing can be applied

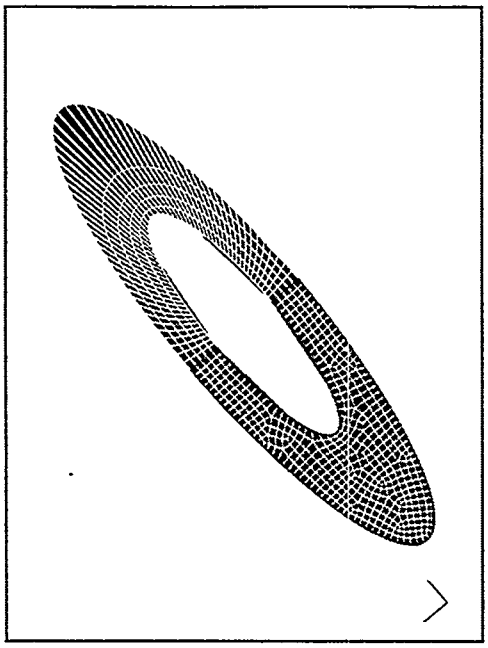


FIG. 1. Winslow Smoothing on Structured and Unstructured Mesh

to those particular nodes.

4. Summary and Conclusions.

Winslow smoothing from structured meshing has been extended to two-dimensional unstructured meshes. Experience with this extension of Winslow within the CUBIT code strongly suggests that resistance to mesh folding has been carried over to the unstructured case. Furthermore, although the computations outlined here are more expensive than those needed for Laplacian smoothing, we have found that the overall CPU time involved with Winslow is slightly less than with Laplacian smoothing. This has been observed in smoothing of structured meshes as well and suggests that the rate of convergence of the iterative smoothing method is larger for Winslow. It is not clear that one can extend the finite difference approach outlined in this paper to three-dimensions. An extension to three-dimensional unstructured meshes using FEM has already been done in [9]. Unfortunately, experience suggests that resistance to mesh folding is not as strong in three-dimensional Winslow smoothing. The author has successfully extended two-dimensional Winslow smoothing to a weighted smoothing scheme for anisotropic smoothing [8].

Acknowledgements

The author wishes to thank Stanly Steinberg, Tim Tautges, David White and the rest of the CUBIT team.

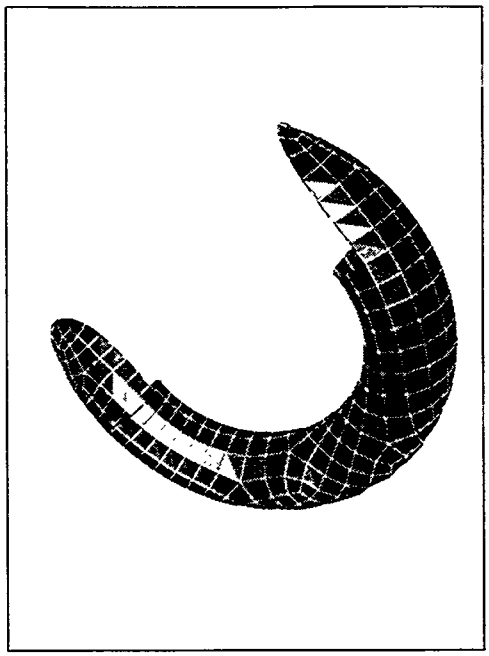


FIG. 2. Winslow Smoothing on Unstructured Mesh on a Torus

REFERENCES

[1] A. Allievi and S. Calisal, *Application of Bubnov-Galerkin Formulation to Orthogonal Grid Generation*, J. Comp. Phys., 98, pp163-173, 1992.

[2] T. Blacker et al., "CUBIT Mesh Generation Environment, Volume 1: User's Manual", SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994.

[3] T. Blacker, M. Stephenson, "Paving: A New Approach to Automated Quadrilateral Mesh Generation", Int. J. Numer. Methods Eng., 32:811-847 (1991).

[4] R. Hagmeier, personal communication, Swansea, 1994.

[5] A. Khamayseh and A. Kuprat, *Anisotropic Smoothing and Solution Adaption for Unstructured Grids*, Intl. J. for Num. Meth. Engr., Vol. 39, pp3163-3174, 1996.

[6] P. Knupp and S. Steinberg, *The Fundamentals of Grid Generation*, CRC Press, Boca Raton FL, 1993.

[7] P. Knupp and R. Luczak, *Truncation Error in Grid Generation: A Case Study*, Num. Methods for PDE's, vol. 11, pp561-571, 1995.

[8] P. Knupp, *Applications of Mesh Smoothing: Copy, Morph, and Sweep on Unstructured Quadrilateral Meshes*, paper submitted for publication.

[9] R. Tipton, *Grid Optimization by Equipotential Relaxation*, manuscript, 26pp, Lawrence Livermore National Laboratory, Novemeber 28, 1990.

[10] J. Thompson, F. Thames, C. Mastin, *Automatic numerical generation of body-fitted curvilinear coordinate system for field containing any number of arbitrary two-dimensional bodies*, J. Comp. Phys., 15, p299-319, 1974.

[11] A, Winslow, *Numerical solution of the quasilinear Poisson equations in a nonuniform triangle mesh*, J. Comp. Phys., 2, pp149-172, 1967.

Appendix I: Expansions of Formula (10)

$k = 1$, $M \geq 3$:

$$\sum_{m=0}^{M-1} \cos\theta_m = 0 \tag{30}$$

$$\sum_{m=0}^{M-1} \sin\theta_m = 0 \tag{31}$$

$k = 2$, $M \geq 3$:

$$\sum_{m=0}^{M-1} \cos^2\theta_m = \frac{M}{2} \tag{32}$$

$$\sum_{m=0}^{M-1} \cos\theta_m \sin\theta_m = 0 \tag{33}$$

$$\sum_{m=0}^{M-1} \sin^2\theta_m = \frac{M}{2} \tag{34}$$

(35)

$k = 3$, $M = 3$:

$$\sum_{m=0}^{M-1} \cos^3\theta_m = \frac{M}{4} \tag{36}$$

$$\sum_{m=0}^{M-1} \cos^2\theta_m \sin\theta_m = 0 \tag{37}$$

$$\sum_{m=0}^{M-1} \cos\theta_m \sin^2\theta_m = -\frac{M}{4} \tag{38}$$

$$\sum_{m=0}^{M-1} \sin^3\theta_m = 0 \tag{39}$$

If $k = 3$ and $M \geq 4$, then the four sums (36)-(39) are all zero.

$k = 4$, $M \geq 3$, $M \neq 4$:

$$\sum_{m=0}^{M=1} \cos^4\theta_m = \frac{3M}{8} \tag{40}$$

$$\sum_{m=0}^{M=1} \cos^3\theta_m \sin\theta_m = 0 \tag{41}$$

$$\sum_{m=0}^{M=1} \cos^2\theta_m \sin^2\theta_m = \frac{M}{8} \tag{42}$$

$$\sum_{m=0}^{M=1} \cos\theta_m \sin^3\theta_m = 0 \tag{43}$$

$$\sum_{m=0}^{M=1} \sin^4 \theta_m = \frac{3M}{8} \tag{44}$$

$k = 4$, $M = 4$:

$$\sum_{m=0}^{M=1} \cos^4 \theta_m = \frac{M}{2} \tag{45}$$

$$\sum_{m=0}^{M=1} \cos^3 \theta_m \sin \theta_m = 0 \tag{46}$$

$$\sum_{m=0}^{M=1} \cos^2 \theta_m \sin^2 \theta_m = 0 \tag{47}$$

$$\sum_{m=0}^{M=1} \cos \theta_m \sin^3 \theta_m = 0 \tag{48}$$

$$\sum_{m=0}^{M=1} \sin^4 \theta_m = \frac{M}{2} \tag{49}$$

Proposal of Benchmarks for 3D Unstructured Tetrahedral Mesh Optimization

Julien Dompierre Paul Labbé François Guibault Ricardo Camarero
Centre de Recherche en Calcul Appliqué (CERCA)
5160, boul. Décarie, suite 400, Montréal, (Québec), H3X 2H9, Canada
[julien|paul|francois|ricardo]@cerca.umontreal.ca

Abstract. *The goal of this paper is to help evaluate the tetrahedral mesh generation, adaptation and optimization algorithms. To achieve this goal, benchmarks are proposed consisting of simple and complete geometries. In order to evaluate the quality of the meshes, the concept of tetrahedron shape measure is clarified and used.*
Finally, as an example, these benchmarks are used to evaluate meshes optimized by ADP3D, *a 3D mesh optimizer. In doing so, a conception of mesh optimization is formulated.*

Keywords: mesh optimization, benchmark, unstructured, tetrahedra, tetrahedron shape measure.

1 Introduction

Publications in the domain of three-dimensional unstructured tetrahedral mesh generation, adaptation and optimization abound. However, they tend to show results only for complex industrial test cases, for which comparisons are most often impossible and results are not reproducible. Sometimes, only images of the optimized meshes are shown, so that it is impossible to tell whether the generation, adaptation or optimization process has converged or not. Different statistics on the quality of the mesh are used, and tetrahedron shape measures may be unusual.

According to the scientific method, one should be able to test a published optimization strategy, reproduce results published elsewhere and compare these results to the ones obtained with other strategies. This is seldom the case in the field of mesh optimization, largely because of the lack of standardization of published test cases. It is thus felt that the definition of a series of benchmarks that would help to establish a basis of comparison constitutes a significant contribution to the field of mesh optimization.

In this paper, a first series of benchmark test cases is proposed that should help compare three-dimensional, optimization meshing techniques on tetrahedra. For each case, the inputs of the benchmarks are clearly defined (§ 2), as well as the measures relating to the optimization process (§ 3) and the ones pertaining to the quality of the results (§ 4). These benchmarks are designed to test the generation and optimization of unstructured tetrahedral meshes. They are not universal or suitable for all meshing problems.

The second part of the paper presents the authors own optimization strategy, implemented in the ADP3D package (§ 5). The benchmarking process is initialized by evaluating the results of ADP3D (§ 6) followed by an analysis of the results (§ 7).

Finally, we conclude that using the benchmarks helps to evaluate the results of tetrahedral mesh optimization algorithms. We feel that the Meshing Roundtable is a good forum to discuss this benchmarks proposal and that it will encourage its use.

2 Description of the Benchmarks

The goal sought in designing these benchmarks is to measure 3D unstructured tetrahedral mesh optimizers. The objectives are first stated and described. The description of the benchmarks themselves will proceed as follows. First, the characteristics of a generic benchmark satisfying the goals will be described, together with the list of the various

measures that are planed to be used. Second, seven benchmarks satisfying the generic characteristics and goals will be proposed.

2.1 Design Objectives of the Benchmarks

To measure the 3D unstructured tetrahedral mesh optimizer, we standardize the inputs, apply the optimizer, and measure the outputs. We assume that the quality of the optimizer will be related to the quality of the outputs. Therefore, the two design objectives of the benchmark are as follow. For the input, the geometry and size specification map are standardized. For the output, the measures used to evaluate the quality of the optimized mesh are standardized.

The size specification map returns informations about what should be the size of the mesh at any location in space. It is used in the following way. Let $P_1 = (x_1, y_1, z_1)^T$ and $P_2 = (x_2, y_2, z_2)^T$ be the endpoints of an edge γ. The Euclidean length of this edge is $\ell_E(\gamma) = \left((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2\right)^{1/2}$. To compute the length of the edge in a metric defined with the target edge length, we refer to [4, 7, 8, 10, 11, 14, 13, 16, 17, 18, 21, 37, 35] among many others, the first reference being [55], and the most complete being [31] and [32]. In brief, if this edge is parameterized as $\gamma(t) = P_1 + t(P_2 - P_1)$ with $t \in [0, 1]$, its metric length $\ell_M(\gamma)$ is given by

$$\ell_M(\gamma) = \int_0^1 \sqrt{\gamma'(t)^T \, M(\gamma(t)) \, \gamma'(t)} \, dt \tag{1}$$

where $\gamma'(t) = P_2 - P_1 = (x_2 - x_1,\, y_2 - y_1,\, z_2 - z_1)^T$ and $M(\gamma(t))$ is the metric defined by the target edge length which is a 3×3 symmetric positive definite matrix. Note that if $M(\gamma(t))$ is the identity matrix, the length of an edge in this metric, $\ell_M(\gamma)$, is simply the Euclidean length $\ell_E(\gamma)$. Multiplying the identity matrix by a constant will stretch or contract space in a uniform way.

The benchmarks proposed here standardize the measures of the quality of a mesh. This is not to say that other important aspects of the optimization process such as generality or efficiency should not be taken into account. Speed and memory are definite issues that are worthy of concern in the benchmark process. But they are very difficult to measure considering the constantly changing heterogeneous environment in which mesh optimizations are done. Generality is also hard to assess quantitatively and might be very desirable in one type of application while being hardly significant in another. Quality of the mesh is therefore the only basis of comparison between optimization techniques.

Even within the limited scope of tetrahedral mesh quality measurement, measures abound and are not all equivalent. This is why a rather broad approach to quality measurement has been favored, which involves many standard measures. Once a mesh has been optimized, computing a complete set of measures on that mesh is a rather straightforward task that does not need to be integrated in the optimization process itself, but can be treated as a post-processing phase for the sole purpose of benchmarking.

This approach to quality assessment leads the way to broad distribution of both the benchmark results and the benchmarking tools used to measure them. In turn, wide distribution of results and tools should allow for the achievement of the second fundamental goal of this benchmarking process: verifiability and repeatability of published results. It is in fact only by making optimized meshes and benchmark results directly available to the widest audience that the establishment of a sound comparison basis will be possible.

A proposed approach to the distribution of benchmarking results is the constitution of a dedicated web site for sharing either links to available results or the results themselves.

2.2 Characteristics of Valid Benchmarking Test Cases

Benchmark test cases must be simple, complete, CAD-free, non-proprietary, inexpensive and available.

This is to allow for greater reproducibility of the cases, independent of the particular computing environments or software used. Most importantly, a good test case has to contain all the necessary information to allow anyone attempting to reproduce the results to do so without further research. Reaserchers are too often confronted to descriptions of test

cases for which essential data remains unavailable. To facilitate porting test cases from one environment to another, geometric data concerning the cases should be made available in a CAD-independent format, or even better, in a CAD-free analytical way. Geometric data should be public domain and as simple as possible. Finally, test cases should not consume too much computing resources. The final size of the results should be compact enough. Making them widely available should not incur an inordinate amount of disk space for storage or bandwidth for transmission.

2.3 Proposed Test Cases

The proposed test cases are very simple. So simple in fact, that a 3D unstructured tetrahedral mesh optimizer might seem an overkill. For example, in the test case of the cube, one can easily get a very regular mesh using a structured hexahedral grid divided into tetrahedra. However, as the 3D unstructured tetrahedral meshers are generic, results obtained on simple test cases *should* be an indicator of meshes that could be obtained for generic geometries.

2.3.1 Unit Tetrahedron Test case. The domain is a unit equilateral tetrahedron. Generally, the necessity for a unstructured tetrahedral mesher to respect complex geometries yields bad elements on the boundaries. A regular tetrahedral domain imposes the least constraints on the mesh. This is the most suitable domain to test the ability of the mesh optimizer to converge. Two unstructured tetrahedral meshes are proposed for this unit equilateral tetrahedron, one with a target edge length of $1/5$ and one with a target edge length of $1/10$.

2.3.2 Unit Cube Test Cases. The domain is a unit cube. With a cubic domain, the boundaries will restrain the possibilities of regular unstructured tetrahedral meshes. This test case will measure the ability of the optimizer to deal with simple boundaries. As the mesh gets finer, the boundaries have less impact on the optimization process and results should improve. Two unstructured tetrahedral meshes are proposed for this unit cube, one with a target edge length of $1/5$ and one with a target edge length of $1/10$.

2.3.3 Unit Sphere Test case. The domain is a unit sphere. With this domain, the target edge lengths are not a natural size of the sphere. This test case will measure the ability of the optimizer to target the prescribed lengths. This test case can also be used to validate the ability of the optimizer to impose the geometry. Two unstructured tetrahedral meshes are proposed for this unit sphere, one with a target edge length of $1/2.5$ and one with a target edge length of $1/5$.

2.3.4 Unit Cube Test Case with Non-Uniform Target Length. One of the most common cause of distorted meshes which needs optimization is a transition in size. This test case is defined with a non-uniform target edge length. The domain is a unit cube $[0,1] \times [0,1] \times [0,1]$ with a target edge length of 0.05 at $z = 0$, a target edge length of 0.20 at $z = 1$ and varies linearly between these two planes. So, the target edge length at a given point $(x, y, z)^T$ is defined as $0.05 + 0.15z$.

As the target edge length is non-uniform, the definition of the length of the edge in a metric (1) is used. In this test case, $M(\gamma(t)) = \mathrm{diag}(\alpha, \alpha, \alpha)$ where $\alpha = (0.05 + 0.15(z_1 + t(z_2 - z_1)))^{-2}$. Performing the integral (1), the length of the edge in the metric is given by

$$\ell_M(\gamma) = \begin{cases} \dfrac{\ell_E(\gamma)}{0.05 + 0.15\overline{z}} & \text{if } z_1 = z_2 = \overline{z}, \\ \dfrac{\ell_E(\gamma)}{0.15(z_2 - z_1)} \log\left(\dfrac{0.05 + 0.15z_2}{0.05 + 0.15z_1}\right) & \text{if } z_1 \neq z_2. \end{cases} \tag{2}$$

Using Eq. (2), if an edge has a unitary metric length, then the edge length fits exactly the target edge length. If the edge has a metric length lower (respectively greater) than the unity, the edge is too short (respectively tall) accordingly to the target edge length.

3 Relevant Measures to Qualify the Mesh

Relevant measures to qualify a 3D unstructured tetrahedral mesh is still an open problem. A few common and basic measures are proposed in a standard format, which should be sufficient to characterize the quality of a mesh.

3.1 Statistics on the Mesh

The first output is the size of the mesh which consists in the number of vertices N_V, edges N_E, faces N_F and tetrahedra N_T. Remember that the Euler-Poincaré relation for domains of these benchmarks is $N_V - N_E + N_F - N_T = 1$.

Major outputs are statistical measures about the meshes, as in [7, 8, 12, 19, 23, 24, 25, 26, 27, 30, 33, 38, 43, 46, 52, 53, 56] among others. These statistics should be non-normalized and normalized where the normalized statistics are on the non-normalized data divided by the average. The statistical outputs are the minimum, the average μ, the maximum, the standard deviation σ, the normalized minimum, (the normalized average is the unity), the normalized maximum and the normalized standard deviation. These measures are 1) the length of the edges, 2) the volume of the tetrahedra, 3) the shape measures of the tetrahedra (see next section for definition of different tetrahedron shape measures).

Statistics on the edge lengths of the mesh evaluate the ability of the mesher/optimizer to track the target length. Statistics on the shape measures of the tetrahedra of the meshes quantify the ability of the mesher/optimizer to create regular meshes.

3.2 Histograms

In order to get an overview of the distribution of the different measures of the mesh, several histograms are proposed. For edge lengths and tetrahedral volumes, we suggest histograms of the percentage of the edge lengths or tetrahedral volumes as a function of normalized data in 0.05 class width. For tetrahedron shape measures, we suggest histograms of the percentage of the tetrahedra as a function of shape measure between 0 and 1 in 0.025 class width.

3.3 Pictures

Finally, some pictures showing nice meshes can be provided but these outputs are similar to color fluid dynamics plots, in the sense that it is always possible to provide nice color pictures of meshes that hide bad tetrahedra...

4 Tetrahedron Shape Measures

We refer to the papers of Liu and Joe [39, 41, 42], and especially [40] for the definitions, discussions and equations of different tetrahedron shape measures. Other reviews of tetrahedron shape measures appear in George [31, 32] and Parthasarathy *et al.* [50]. There are so many tetrahedron shape measures in the literature that we *suggest* the following global definition, derived from [40]:

DEFINITION 1: *A* **tetrahedron shape measure** *is a continuous function that evaluates the quality of a tetrahedron. It must be invariant under translation, rotation, reflection and uniform scaling of the tetrahedron. It must be maximum for the regular tetrahedron and it must be minimum for a degenerate tetrahedron. There is no local maximum other than the global maximum for a regular tetrahedron and there is no local minimum other than the global minimum for a degenerate tetrahedron. For the ease of comparison, it* should *be scaled to the interval* $[0, 1]$*, and be* 1 *for the regular tetrahedron and* 0 *for a degenerate tetrahedron.*

We refer to [31, 32, 40, 50] for the definition and classification of degenerate tetrahedra. In short, a degenerate tetrahedron is a tetrahedron whose volume vanishes. The case where, by uniform scaling to zero, the volume and the edges of the tetrahedron vanish, is *not* considered to be a degenerate tetrahedron. But the case where the volume of the tetrahedron vanishes and some of the edges of the tetrahedron do not vanish, is considered to be a degenerate tetrahedron.

When the volume of the tetrahedron is negative, the tetrahedron is more than degenerate, it is inverted. In this case, some tetrahedron shape measures return a negative number, some others return a positive number because, for example, they depend on the square of the volume. We will assume that all the tetrahedron shape measures return a positive

number and so they are not used to determine the sign of the volume. If a mesh contains negative tetrahedra, the mesh optimizer should try to remove them by optimizing the volume of the tetrahedra, by minimizing the sum of the absolute value of the volume of the tetrahedra like in Coupez [20] for example, the tetrahedra shape measure being useless at this stage.

A review of tetrahedron shape measures is done in the following sections. Two of them do not satisfy the Def. 1 of a tetrahedron shape measure. Notations are those of [40]. $T(t_0, t_1, t_2, t_3)$ stands for a non-degenerate tetrahedron T with vertices t_0, t_1, t_2 and t_3; v denotes the volume of T, $s_0 = \text{area}(\triangle t_1 t_2 t_3)$, $s_1 = \text{area}(\triangle t_0 t_2 t_3)$, $s_2 = \text{area}(\triangle t_0 t_1 t_3)$, $s_3 = \text{area}(\triangle t_0 t_1 t_2)$; and $l_{ij} = \|t_j - t_i\|, 0 \leq i < j \leq 3$, denotes the length of the six edges $\overline{t_i t_j}$ of T.

4.1 The Radius Ratio

The radius ratio ρ of a tetrahedron T is defined to be $\rho = N\rho_{in}/\rho_{out}$ where ρ_{in} and ρ_{out} are the inradius and circumradius of T, respectively, and N is the dimension of the space. In [40], an easy way to compute this tetrahedron shape measure in 3D is given:

$$\rho_{in} = 3v \Big/ \sum_{i=0}^{3} s_i, \tag{3}$$

$$\rho_{out} = \frac{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}{24v} \tag{4}$$

$$\rho = 3\frac{\rho_{in}}{\rho_{out}} = \frac{216v^2}{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}\sum_{i=0}^{3} s_i}, \tag{5}$$

where a, b and c are the products of the lengths of opposite edges of T.

4.2 The Mean Ratio

This definition of the mean ratio η of a tetrahedron T is taken from [40]. Let $R(r_0, r_1, r_2, r_3)$ be an equilateral tetrahedron having the same volume as T. Let M be the matrix involved in an affine transformation from R to T, i.e., $t_i = Mr_{p(i)} + b, 0 \leq i \leq 3$, where $(p(0), p(1), p(2), p(3))$ is a permutation of $(0, 1, 2, 3)$ and b is a translation vector. Then the mean ratio η of a tetrahedron T is the ratio of the geometric mean over the algebraic mean of the eigenvalues λ_1, λ_2 and λ_3 of the matrix $M^T M$. Remember that, for positive numbers, the geometric mean is less or equal to the algebraic mean. In [39], a simple expression, involving only the volume and the edge lengths, is demonstrated to substitute this complex definition of the mean ratio :

$$\eta = \frac{3\sqrt[3]{\det(M^T M)}}{\text{tr}(M^T M)} = \frac{3\sqrt[3]{\lambda_1\lambda_2\lambda_3}}{\lambda_1 + \lambda_2 + \lambda_3} = \frac{12\sqrt[3]{9v^2}}{\sum_{0 \leq i < j \leq 3} l_{ij}^2}. \tag{6}$$

4.3 The Solid Angle

A mesh is a Delaunay mesh if the circumsphere of each element does not contain any other vertex of the mesh. In 2D, the Delaunay property is equivalent to maximize the minimum of the angles of the triangles of the mesh, called the max–min angle criterion. In 3D, Delaunay meshes do not generally satisfy the max–min angle criterion (see [51] for properties of Delaunay triangulations). In fact, Delaunay mesh generators often produce badly shaped tetrahedra like the sliver of Fig. 2. To optimize the mesh, the temptation to use a tetrahedron shape measure based on tetrahedron angles is great. Tetrahedron shape measures based on the minimum of the solid angle θ_{min} and the minimum of the dihedral angle φ_{min} can be used. They are more complex to evaluate and more costly because of inverse trigonometric functions.

Again, following the notation of [40], the solid angle θ_i at the vertex t_i of the tetrahedron $T(t_0, t_1, t_2, t_3)$ is defined to be the surface area formed by projecting each point of the face not containing t_i to the unit sphere centered at t_i. The area of a unit sphere is 4π, the maximum solid angle for a positive tetrahedron is 2π in the case of a flat tetrahedron

where a vertex sees half of the space. The solid angle at the corner of a rectangular tetrahedron is $\pi/2$. It is shown in [28] that $0 \leq \sum_{i=0}^{3} \theta_i \leq 2\pi$. Therefore, a large solid angle near 2π for T implies that T has small solid angles. That is the reason the tetrahedron shape measure based on the solid angles is function of the minimum of the solid angles. Liu and Joe [40] give a simpler formula to computes solid angles:

$$\theta_{min} = \alpha \min_{0 \leq i \leq 3} \theta_i, \tag{7}$$

$$\sin(\theta_i/2) = 12v \left(\prod_{\substack{j,k \neq i \\ 0 \leq j < k \leq 3}} \left((l_{ij} + l_{ik})^2 - l_{jk}^2 \right) \right)^{-1/2} \tag{8}$$

where $\alpha^{-1} = 6 \arcsin\left(\sqrt{3}/3\right) - \pi = 0.5512856$ is the value of the four solid angles of the regular tetrahedron.

Equation (8) can be used to measure solid angles less or equal to π. In fact, the result of $\theta_i = 2\arcsin(\cdots)$ is in the interval $[0, \pi]$. So the maximum solid angle of a tetrahedron can be badly evaluated using Eq. (8). However, the minimum solid angle is not affected. Let $\theta_0 \leq \theta_1 \leq \theta_2 \leq \theta_3$. Since $0 \leq \sum_{i=0}^{3} \theta_i \leq 2\pi$, only θ_3 may be larger than π. The computation with Eq. (8) of θ_3 returns $\widetilde{\theta}_3 = 2\pi - \theta_3$. Substituting θ_3 by $2\pi - \widetilde{\theta}_3$ in $\sum_{i=0}^{3} \theta_i \leq 2\pi$ gives $\theta_0 + \theta_1 + \theta_2 \leq \widetilde{\theta}_3$. So, θ_0 which is lower or equal to θ_1 and θ_2 is also lower or equal to $\widetilde{\theta}_3$. The conclusion is that even if the Eq. (8) does not return the correct value for a solid angle greater than π, it does not affect the tetrahedron shape measure (7) based on the minimum of the solid angles.

Since the right hand side of Eq. (8) has no trigonometric functions, from a computational point of view, a cheaper shape tetrahedron measure is

$$\sigma_{min} = \beta \min_{0 \leq i \leq 3} \sigma_i, \tag{9}$$

where $\sigma_i = \sin(\theta_i/2)$. $\beta^{-1} = \sin(\alpha^{-1}/2) = \sqrt{6}/9 = 0.2721655$ is the value of σ_i for the four solid angles of the regular tetrahedron.

4.4 The Dihedral Angle

Each of the six edges of a tetrahedron is surrounded by two triangular faces. At a given edge, the dihedral angle between the two faces is the angle between the intersection of these faces and a plane perpendicular to the edge. For a positive tetrahedron, the dihedral angle is bounded by zero and π. It is equal to π minus the angle between the normals of the faces. The minimum dihedral angle is a tetrahedron shape measure.

$$\varphi_{min} = \alpha \min_{0 \leq i < j \leq 3} \varphi_{ij} = \alpha \min_{0 \leq i < j \leq 3} \left(\pi - \arccos\left(n_{ij1} \cdot n_{ij2} \right) \right), \tag{10}$$

where n_{ij1} and n_{ij2} are the two triangular faces adjacent to the edge ij and $\alpha^{-1} = \pi - \arccos(-1/3) = 1.230959$ is the value of the six dihedral angles of the regular tetrahedron.

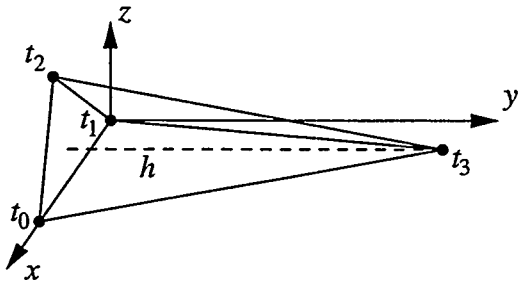


Figure 1: When h tends to infinity, or equivalently when edges $\overline{t_0t_1}$, $\overline{t_1t_2}$ and $\overline{t_2t_0}$ tend to zero, the tetrahedron $T(t_0, t_1, t_2, t_3)$ becomes a *thorn* with regular dihedral angles.

According to Def. 1 of a tetrahedron shape measure, the minimum of the dihedral angles φ_{min} is a not a tetrahedron shape measure. The underlying problem with tetrahedron shape measures based on the dihedral angles is that they fail

to detect some degenerated tetrahedra as the *thorn*. Referring to Fig. 1, let vertices t_0, t_1 and t_2 form a regular triangle in the xz-plane. Let the vertex t_3 be offset by a height h to the barycenter of the triangle $\triangle t_0 t_1 t_2$. When the height h is large, or equivalently, when the three edges $\overline{t_0t_1}$, $\overline{t_1t_2}$ and $\overline{t_2t_0}$ are scaled down at the same rate, the dihedral angles along edges $\overline{t_0t_1}$, $\overline{t_1t_2}$ and $\overline{t_2t_0}$ will be close to 90 degrees and dihedral angles along edges $\overline{t_0t_3}$, $\overline{t_1t_3}$ and $\overline{t_2t_3}$ will be close to 60 degrees.

4.5 The Edge Ratio

The edge ratio r of a tetrahedron T is defined to be the ratio between the smallest edge over the largest edge of the tetrahedron, i.e.,

$$r = \min_{0 \le i < j \le 3} l_{ij} \Big/ \max_{0 \le i < j \le 3} l_{ij}. \tag{11}$$

According to Def. 1 of a tetrahedron shape measure, the edge ratio r is not a tetrahedron shape measure. It fails to detect some degenerated tetrahedra as the *sliver*. Referring to Fig. 2, suppose that the vertices t_0, t_1 an t_2 are in the xy-plane and the vertex t_3 if offset by a distance h of the xy-plane. When h tends to zero, the four vertices tend to be coplanar, the volume of the tetrahedron tends to zero but the edge ratio r does not vanish.

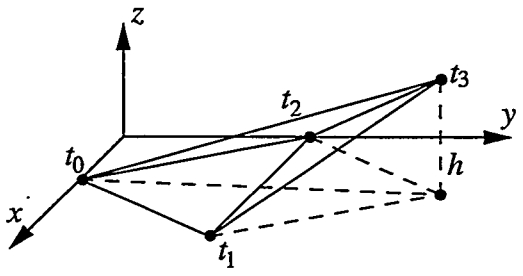


Figure 2: When h tends to zero, the tetrahedron $T(t_0, t_1, t_2, t_3)$ becomes a *sliver* with null volume but with non-null edge ratio r.

4.6 The Aspect Ratio

Tetrahedron shape measures that are the ratio of two characteristic sizes of the tetrahedron can loosely be called aspect ratio. They are usually some quantity vanishing with the volume normalized by something that does not vanish when the volume vanishes. So, the aspect ratio may be proportional to the volume of the tetrahedron, the radius, the area or the volume of the insphere, or the minimum of the four solid angles. It can not be a function of the circumsphere, of the smallest edge and of the minimum dihedral angle because degenerate tetrahedron may have non-null circumsphere, smallest edge or dihedral angle. The aspect ratio may be normalized by the longest edge, the average of the edges, the sum of the edges (the perimeter), the sum of the area of the faces, the radius, the area or the volume of the circumsphere, etc. The possibilities are endless.

Among all of them, we arbitrarily select the tetrahedron aspect ratio γ inspired from GAMMA project (Génération Automatique de Maillages et Méthodes d'Adaptation) at INRIA (Institut National de Recherche en Informatique et en Automatique), France, because it is frequently used [15, 26, 27, 30, 31, 32]:

$$\gamma = \frac{12}{\sqrt{6}} \frac{\rho_{in}}{\max\limits_{0 \le i < j \le 3} l_{ij}}. \tag{12}$$

4.7 Equivalence of Tetrahedron Shape Measures

One of the deepest analysis of tetrahedron shape measures available is surely from Liu and Joe [40]. They define the notion of tetrahedron shape measure equivalence:

DEFINITION 2 (FROM [40]): *Let μ and ν be two different tetrahedron shape measures with values $\in [0,1]$. μ and ν are equivalent if there exist positive constants c_0, c_1, e_0 and e_1 such that $c_0\mu^{e_0} \leq \nu \leq c_1\mu^{e_1}$.*

Liu and Joe [40] proved the equivalence of the tetrahedron shape measures ρ, η and σ_{min} (Eqs. (5), (6) and Eq. (9) respectively). It implies that if one of these tetrahedron shape measures approaches zero, which indicates a poorly-shaped tetrahedron, then so do the others. Conversely, if one of these tetrahedron shape measures approaches unity, then so do the others. But the rate at which they approach zero or unity may differ as, for example, do μ and $\nu = \mu^2$.

Liu and Joe [40] made a conjecture about how should be tetrahedron shape measures to be equivalent. We propose the following *conjecture*:

CONJECTURE: *All tetrahedron shape measures that satisfy Definition 1 are equivalent in the sense of Definition 2.*

This conjecture states that any shape measures that satisfy Def. 1 can be used by the mesh optimizer. Remember that, by definition, they will all measure all tetrahedron degeneracies. They will all be sensitive to badly shaped tetrahedra.

More surprisingly, the more a mesh is optimized with a given tetrahedron shape measure, the closer to the optimal mesh it is for any other tetrahedron shape measure. At the limit, if it were possible to mesh a domain with only equilateral tetrahedra, as it is in 2D, all mesh optimizer should converge to that mesh, whichever shape measure is used in the mesh optimizer. The problem is that the optimal mesh does not exist in 3D. It is impossible to fill the space with regular tetrahedra. So, the converged state is unknown and depend slightly of the tetrahedron shape measure used.

5 ADP3D and Mesh Optimization

5.1 ADP3D

ADP3D (Adaptive Discretization Package in 3D), is a object oriented C++ library for the generation, adaptation and optimization of structured and unstructured, isotropic and anisotropic, monozone and multizone, hexahedral, prismatic and tetrahedral meshes. A full description of ADP3D is out of the scope of this paper but here is a brief description of the unstructured tetrahedral mesh optimization.

In ADP3D, the mesh is optimized in an iterative process that combines mesh refinement, mesh coarsening, node smoothing and edge swapping. If the length of an edge is greater than the target length times a threshold value, then a node is added in the middle of the edge. This will divide the n tetrahedra around the edge in $2n$ tetrahedra. If the length of an edge is lower than the target length times a threshold value, then the two nodes of the edges are collapsed in one node. The edge and the n tetrahedra around the edge are destroyed. Each edge is checked to see if swapping the edge could increase the minimum of the shape measure of the involved tetrahedra (see [20, 24, 29, 31, 32, 33, 38, 43] among others). The tetrahedron shape measure used is the mean ratio η (Eq. (6)). Finally, nodes are moved one by one iteratively with a kind of spring analogy (see [4, 5, 9, 10, 11, 19, 21, 22, 24, 36, 44, 45, 47, 48, 49, 54, 57] among others). After refinement with node addition, and after coarsening with node removal, the mesh is topologically regularized with edge swapping, and geometrically regularized with node movement.

There is really nothing new in these algorithms to locally modify the mesh in order to reach the target length, almost everything is described in [31, 32]. In fact, ADP3D is an ongoing project, and is now relatively crude. Our main concern is not yet efficiency, memory and speed, but only convergence. It is not an easy problem to combine all the local modifications to the mesh such that they work together to optimize the mesh [24, 56]. For example, mesh refinement may undo what mesh coarsening did, or diagonal swapping may swap edges whose nodes were optimized with node movement. The results shown in Sec. 6 are not optimal and if someone gets better meshes, to the benefit of the science, they are welcome to publish how the meshes were obtained. This is the main goal of this benchmarks proposal.

5.2 Mesh Optimization

In many papers on unstructured tetrahedral meshes, authors use different words like enhancement, regularization, improvement, smoothing and optimization, but it is not clear that they speak about the same thing. Let us explain what we consider to be mesh optimization.

According to the Webster's Collegiate Dictionary, *optimization* is the *process or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible* and specifically, *optimization* is the *mathematical procedures (as finding the maximum of a function) involved in this.*

The goal of a mesh optimizer is not to get a better mesh than a given one, but it is to get a mesh that fits as well as possible a given specification map. Optimization techniques can be used for mesh generation and for mesh adaptation. In mesh generation, the specification map is a user defined size while in mesh adaptation, it is deduced from an error estimator. For the mesh generation phase, optimization does not have to be converged because a high quality initial mesh is usually not so important and will be adapted later on. For the mesh adaptation phase, optimization is usually more converged in order to get a mesh to compute a better solution.

A mesh optimizer is like a finite element solver. For a FE solver, time stepping, residual smoothing, upwinding, artificial viscosity, non-linear GMRES, preconditionning, multigrid, under-relaxation, variables segregation, etc, are many elements that are combined together to get a robust and fast solver that converges to the solution. For a mesh optimizer, edge swapping, face swapping, tetrahedron shape measures, node smoothing, spring analogy, under-relaxation, edge splitting, face splitting, element splitting, node removal, edge removal, etc, are many elements that can be combined to get a robust and fast mesh optimizer that converges to the optimal mesh. The goal of the ongoing research on mesh optimization is to build a *mesh solver* that will converge in a robust and fast way towards the optimal mesh.

When mesh optimization will be currently used, it will usually not be converged within the machine accuracy. But for these benchmarks, in the process of designing the mesh optimizer, it is crucial to know if it is convergent to evaluate the optimization efficiency. Using again the analogy with a FE solver, solution may converge up to a residual of 10^{-4} or 10^{-5} because it is considerably accurate enough, but the FE solver must be capable to get down the residual to machine accuracy. The same applies for the mesh optimizer: you may converge the mesh a little for mesh generation purposes, and you may want to converge more when doing mesh adaptation, but in all cases, the mesh optimizer should be able to converge to machine accuracy towards the optimal mesh.

As a good iterative FE solver is not sensitive to the initial guess, a good mesh optimizer should be rather independent from the initial mesh. That is the reason why the benchmarks do not provide initial meshes. In fact, the initial meshes used for the results of Sec. 6 are coarse handmade meshes with five to nine vertices.

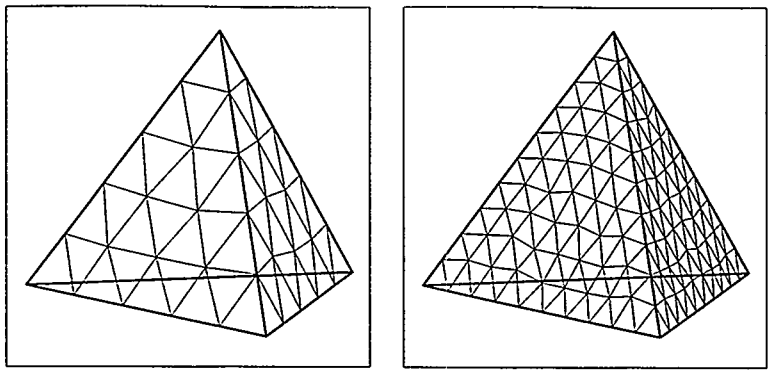
6 Results

Results computed in this section were obtained by using the benchmarks to evaluate meshes optimized by ADP3D.

6.1 Benchmarks on the Unit Tetrahedron

The optimized three-dimensional unstructured tetrahedral mesh for the unit tetrahedron with a target edge length of $1/5$ has 56 vertices, 230 edges, 300 faces and 125 elements. See Fig. 3(a). The statistical data corresponding to this three-dimensional unstructured tetrahedral mesh are given in Table 1. The histograms corresponding to these data are in Fig. 4.

The optimized three-dimensional unstructured tetrahedral mesh for the unit tetrahedron with a target edge length of $1/10$ has 309 vertices, 1606 edges, 2395 faces and 1097 elements. See Fig. 3(b). The statistical data corresponding to this three-dimensional unstructured tetrahedral mesh are given in Table 2. The histograms corresponding to these data are in Fig. 5.



(a) 1/5 (b) 1/10

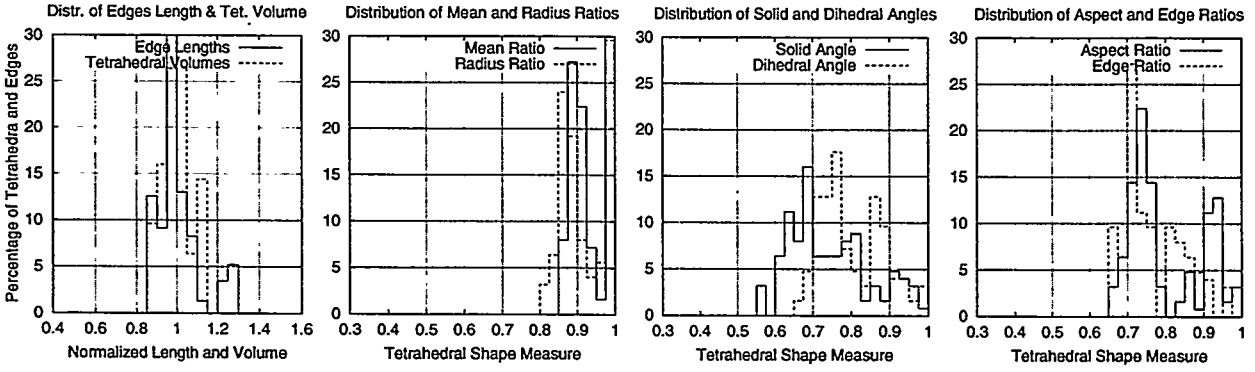
Figure 3: Optimized three-dimensional unstructured tetrahedral mesh for the unit tetrahedron with a target edge length of 1/5 and 1/10.

Table 1: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **tetrahedron** with a target edge length of **1/5**.

	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.1766	0.2060	0.2650	0.0195	0.8573	1.0	1.2864	0.0945
Tetrahedral Volume	8.269(-4)	9.428(-4)	1.084(-3)	6.410(-5)	0.8771	1.0	1.1494	0.0680
Radius ratio ρ	0.8227	0.9135	0.9992	0.0550	0.9006	1.0	1.0939	0.0602
Mean ratio η	0.8551	0.9292	0.9994	0.0442	0.9203	1.0	1.0755	0.0476
Solid angle θ_{min}	0.5653	0.7418	0.9797	0.1039	0.7621	1.0	1.3207	0.1400
Dihedral angle φ_{min}	0.6680	0.8018	0.9808	0.0796	0.8331	1.0	1.2232	0.0993
Edge ratio r	0.6701	0.7668	0.9648	0.0782	0.8739	1.0	1.2583	0.1020
Aspect ratio γ	0.6732	0.7987	0.9814	0.0949	0.8429	1.0	1.2286	0.1188

6.2 Benchmarks on the Unit Cube

The optimized three-dimensional unstructured tetrahedral mesh for the unit cube with a target edge length of 1/5 has 313 vertices, 1709 edges, 2612 faces and 1215 elements. See Fig. 6(a). The statistical data corresponding to this

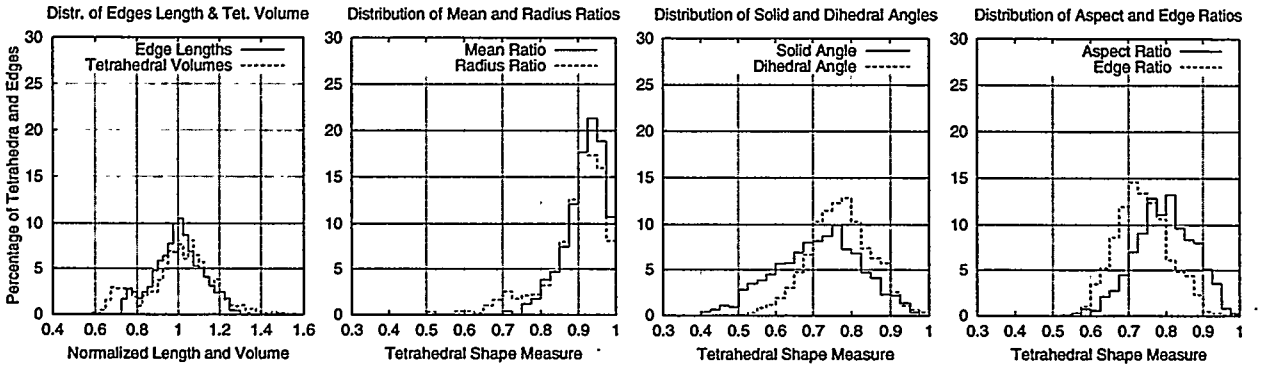


(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

Figure 4: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **tetrahedron** with a target edge length of **1/5**.

Table 2: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **tetrahedron** with a target edge length of $\mathbf{1/10}$.

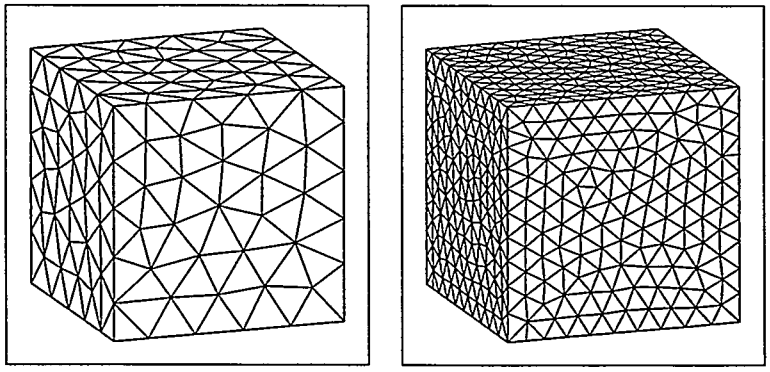
	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.0725	0.1007	0.1504	0.0118	0.7199	1.0	1.4933	0.1175
Tetrahedral Volume	6.230(-5)	1.074(-4)	1.691(-4)	1.779(-5)	0.5799	1.0	1.5738	0.1656
Radius ratio ρ	0.5115	0.8932	0.9997	0.0805	0.5727	1.0	1.1192	0.0901
Mean ratio η	0.7138	0.9166	0.9997	0.0523	0.7787	1.0	1.0907	0.0571
Solid angle θ_{min}	0.4018	0.7102	0.9673	0.1109	0.5658	1.0	1.3621	0.1562
Dihedral angle φ_{min}	0.5237	0.7690	0.9895	0.0806	0.6810	1.0	1.2867	0.1048
Edge ratio r	0.5278	0.7387	0.9780	0.0701	0.7146	1.0	1.3240	0.0949
Aspect ratio γ	0.5535	0.7965	0.9872	0.0782	0.6949	1.0	1.2394	0.0982



(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

Figure 5: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **tetrahedron** with a target edge length of $\mathbf{1/10}$.

three-dimensional unstructured tetrahedral mesh are given in Table 3. The histograms corresponding to these data are in Fig. 7.



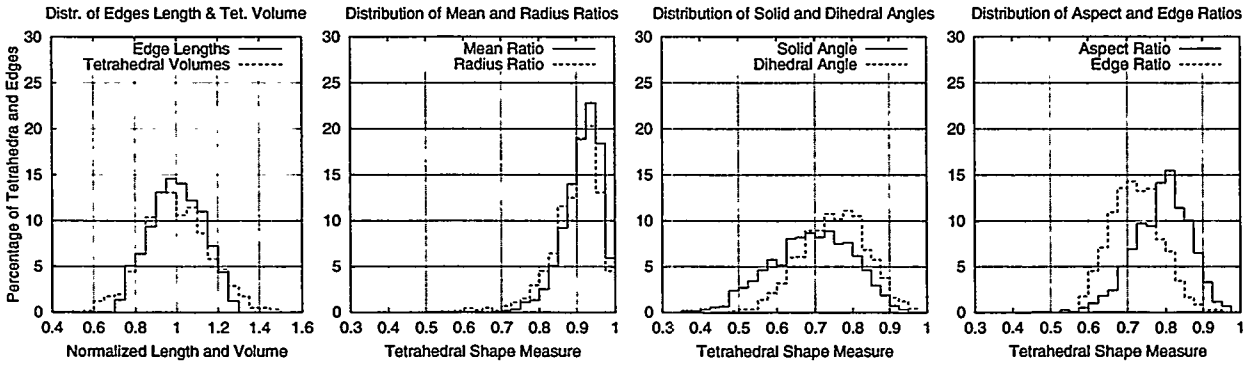
(a) 1/5 (b) 1/10

Figure 6: Optimized three-dimensional unstructured tetrahedral mesh for the unit cube with a target edge length of 1/5 and 1/10.

The optimized three-dimensional unstructured tetrahedral mesh for the unit cube with a target edge length of 1/10 has 1997 vertices, 12158 edges, 19598 faces and 9436 elements. See Fig. 6(b). The statistical data corresponding to this three-dimensional unstructured tetrahedral mesh are given in Table 4. The histograms corresponding to these data are

Table 3: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **cube** with a target edge length of $1/5$.

	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.1440	0.1998	0.2566	0.0250	0.7206	1.0	1.2843	0.1250
Tetrahedral Volume	3.982(-4)	8.230(-4)	1.419(-3)	1.327(-4)	0.4838	1.0	1.7235	0.1612
Radius ratio ρ	0.3941	0.8947	0.9974	0.0679	0.4405	1.0	1.1148	0.0759
Mean ratio η	0.6208	0.9125	0.9978	0.0506	0.6803	1.0	1.0934	0.0554
Solid angle θ_{min}	0.3502	0.6919	0.9374	0.1064	0.5061	1.0	1.3548	0.1538
Dihedral angle φ_{min}	0.3959	0.7504	0.9746	0.0866	0.5277	1.0	1.2988	0.1154
Edge ratio r	0.5764	0.7236	0.9399	0.0651	0.7966	1.0	1.2990	0.0899
Aspect ratio γ	0.4630	0.7914	0.9634	0.0731	0.5851	1.0	1.2173	0.0924



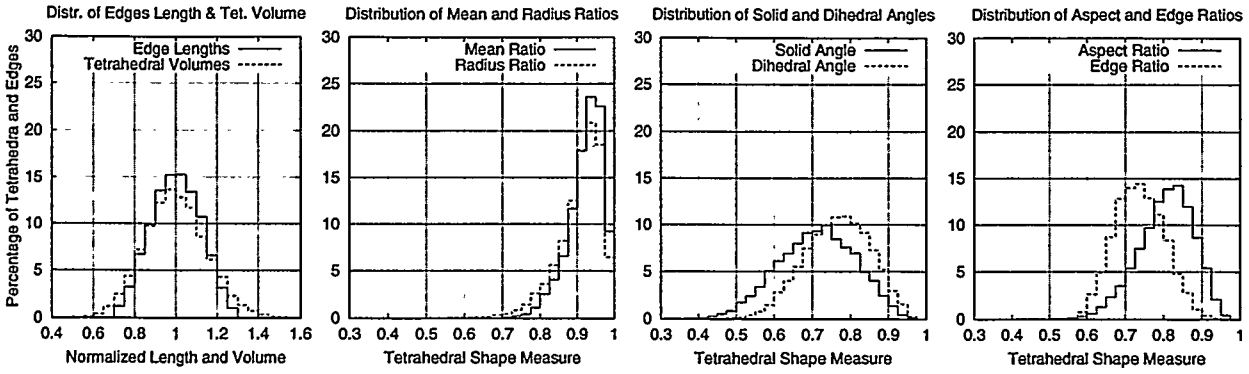
(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

Figure 7: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **cube** with a target edge length of $1/5$.

in Fig. 8.

Table 4: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **cube** with a target edge length of $1/10$.

	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.0721	0.1004	0.1305	0.0117	0.7182	1.0	1.2993	0.1169
Tetrahedral Volume	5.748(-5)	1.060(-4)	1.817(-4)	1.581(-5)	0.5424	1.0	1.7145	0.1492
Radius ratio ρ	0.5151	0.9067	0.9978	0.0602	0.5681	1.0	1.1004	0.0664
Mean ratio η	0.6559	0.9222	0.9979	0.0468	0.7112	1.0	1.0821	0.0508
Solid angle θ_{min}	0.2962	0.7115	0.9697	0.0996	0.4162	1.0	1.3629	0.1340
Dihedral angle φ_{min}	0.4207	0.7657	0.9768	0.0852	0.5494	1.0	1.2756	0.1113
Edge ratio r	0.5696	0.7375	0.9504	0.0641	0.7724	1.0	1.2887	0.0869
Aspect ratio γ	0.4862	0.8058	0.9741	0.0709	0.6034	1.0	1.20883	0.0880

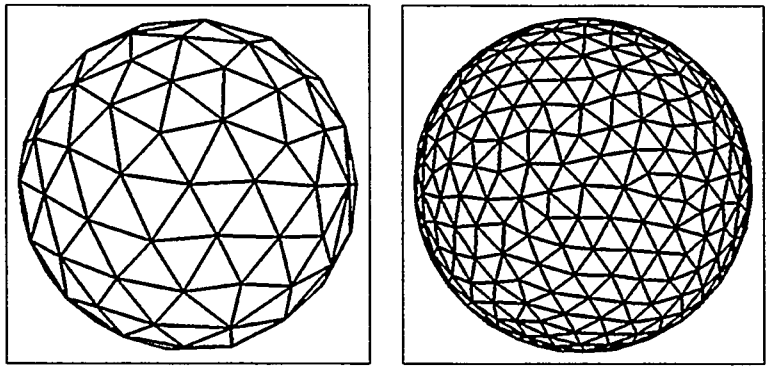


(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

Figure 8: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **cube** with a target edge length of $1/10$.

6.3 Benchmarks on the Unit Sphere

For these two test cases, a bug prevented the optimizer to fully converge. The optimized three-dimensional unstructured tetrahedral mesh for the unit sphere with a target edge length of $1/2.5$ has 156 vertices, 828 edges, 1253 faces and 580 elements. See Fig. 9(a). The statistical data corresponding to this three-dimensional unstructured tetrahedral mesh are given in Table 5. The histograms corresponding to these data are in Fig. 10.



(a) $1/2.5$ (b) $1/5$

Figure 9: Optimized three-dimensional unstructured tetrahedral mesh for the unit sphere with a target edge length of $1/2.5$ and $1/5$.

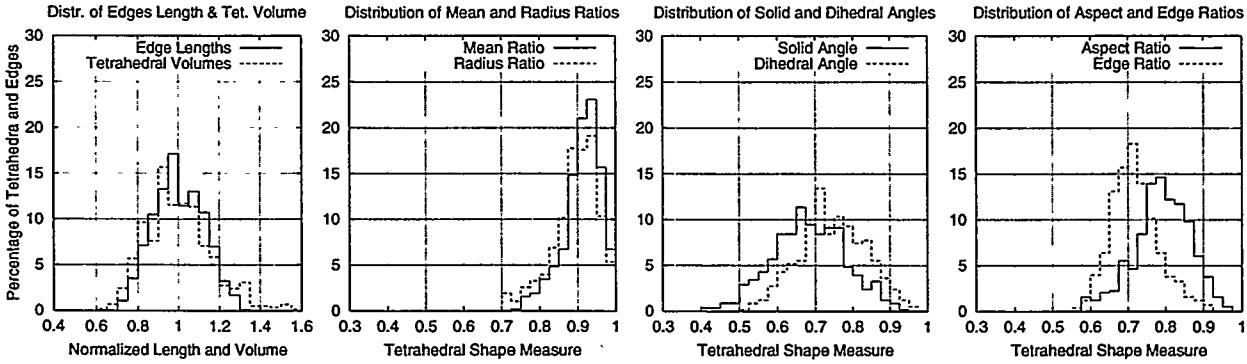
The optimized three-dimensional unstructured tetrahedral mesh for the unit sphere with a target edge length of $1/5$ has 1044 vertices, 6323 edges, 10185 faces and 4905 elements. See Fig. 9(b). The statistical data corresponding to this three-dimensional unstructured tetrahedral mesh are given in Table 6. The histograms corresponding to these data are in Fig. 11.

6.4 Benchmarks on the Unit Cube with Non-Uniform Target Edge Length

The optimized three-dimensional unstructured tetrahedral mesh for the unit cube with a non-uniform target edge length of $0.05 + 0.15z$ has 2368 vertices, 14219 edges, 22751 faces and 10899 elements. See Fig. 12. The statistical data corresponding to this three-dimensional unstructured tetrahedral mesh are given in Table 7. The histograms corresponding to these data are in Fig. 13.

Table 5: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **sphere** with a target edge length of $\mathbf{1/2.5}$.

	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.2911	0.4024	0.5247	0.0490	0.7234	1.0	1.3040	0.1219
Tetrahedral Volume	4.270(-3)	6.799(-3)	1.073(-2)	1.083(-3)	0.6280	1.0	1.5777	0.1592
Radius ratio ρ	0.7039	0.8942	0.9953	0.0601	0.7871	1.0	1.1130	0.0672
Mean ratio η	0.7479	0.9117	0.9960	0.0494	0.8203	1.0	1.0925	0.0541
Solid angle θ_{min}	0.4202	0.6849	0.9291	0.0966	0.6135	1.0	1.3566	0.1411
Dihedral angle φ_{min}	0.5137	0.7463	0.9603	0.0880	0.6883	1.0	1.2866	0.1179
Edge ratio r	0.5716	0.7183	0.9221	0.0628	0.7958	1.0	1.2837	0.0874
Aspect ratio γ	0.5826	0.7900	0.9619	0.0738	0.7375	1.0	1.2176	0.0934



(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

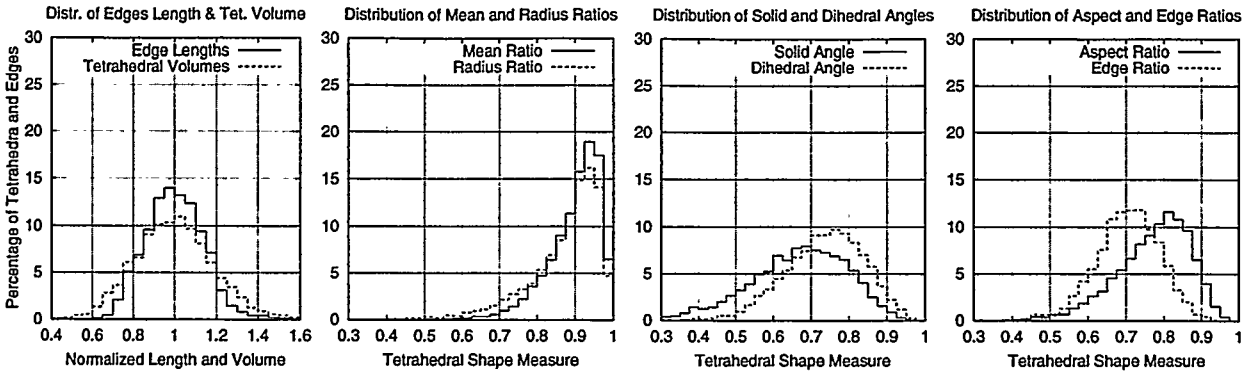
Figure 10: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **sphere** with a target edge length of $\mathbf{1/2.5}$.

7 Analysis of the Results

This paper is the first contribution to the benchmarks. It can be used by others for the purpose of evaluating 3D unstructured tetrahedral mesh optimization. However, the results give us an opportunity to evaluate the benchmarking

Table 6: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **sphere** with a target edge length of $\mathbf{1/5}$.

	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.0998	0.2022	0.3391	0.0284	0.4935	1.0	1.6772	0.1406
Tetrahedral Volume	3.272(-4)	8.411(-4)	1.738(-3)	1.627(-4)	0.3890	1.0	2.0669	0.1934
Radius ratio ρ	0.3244	0.8727	0.9963	0.0941	0.3717	1.0	1.1416	0.1079
Mean ratio η	0.5008	0.8975	0.9969	0.0695	0.5580	1.0	1.1108	0.0774
Solid angle θ_{min}	0.1778	0.6647	0.9632	0.1289	0.2675	1.0	1.4491	0.1940
Dihedral angle φ_{min}	0.2508	0.7351	0.9707	0.1027	0.3411	1.0	1.3204	0.1397
Edge ratio r	0.3887	0.7049	0.9344	0.0833	0.5515	1.0	1.3256	0.1182
Aspect ratio γ	0.3166	0.7721	0.9684	0.0961	0.4101	1.0	1.2543	0.1245



(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

Figure 11: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **sphere** with a target edge length of $1/5$.

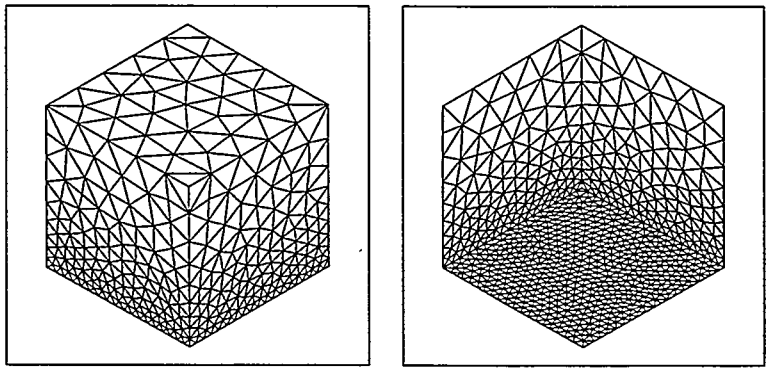
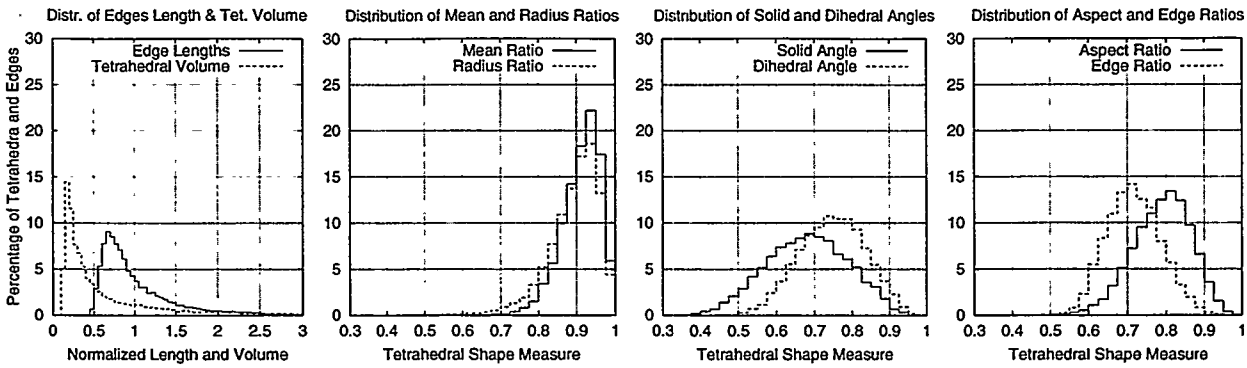


Figure 12: Two different views of the optimized three-dimensional unstructured tetrahedral mesh for the unit cube with a target edge length of $0.05 + 0.15z$.

process itself and to study the behavior of the requested mesh measures.

Table 7: Statistical data corresponding to the optimized three-dimensional unstructured tetrahedral mesh for the unit **cube** with a target edge length of $0.05 + 0.15z$.

	Non-normalized				Normalized			
	min	μ	max	σ	min	μ	max	σ
Edge Length	0.0368	0.0818	0.2514	0.0358	0.4499	1.0	3.0726	0.4380
Tetrahedral Volume	9.046(-6)	9.175(-5)	1.393(-3)	1.465(-4)	0.0986	1.0	15.181	1.5962
Radius ratio ρ	0.4387	0.8916	0.9981	0.0671	0.4920	1.0	1.1194	0.0752
Mean ratio η	0.5715	0.9107	0.9983	0.0502	0.6275	1.0	1.0961	0.0551
Solid angle θ_{min}	0.3215	0.6812	0.9676	0.1076	0.4719	1.0	1.4203	0.1579
Dihedral angle φ_{min}	0.3698	0.7480	0.9814	0.0867	0.4943	1.0	1.3120	0.1159
Edge ratio r	0.5224	0.7170	0.9476	0.0684	0.7286	1.0	1.3216	0.0954
Aspect ratio γ	0.4172	0.7896	0.9792	0.0741	0.5284	1.0	1.2402	0.0938



(a) Lengths and Volumes (b) Mean and Radius Ratios (c) Solid and Dihedral Angles (d) Edge and Aspect Ratios

Figure 13: Distribution of normalized edge lengths and tetrahedral volumes and of the six tetrahedral shape measures for the optimized three-dimensional unstructured tetrahedral mesh for the unit **cube** with a target edge length of $0.05 + 0.15z$.

7.1 General Observations

The average of the edge lengths is close to the target length for all the optimized meshes. The maximum error is 3% for the unit tetrahedron with a target length of $1/5$, the average error being 1%.

Some test cases are too coarse to yield meaningful distributions for plotting in histograms, especially the unit tetrahedron with a target length of $1/5$ and the unit sphere with a target length of $1/2.5$. Finer optimized meshes give smoother and more significant distributions.

The distributions of edge lengths and tetrahedral volumes show nearly Gaussian distribution. For such distributions, the average and the standard deviation are good measures that can be used to quantify distributions. The minimum of the solid angles θ_{min} and the edge ratio τ also have nearly Gaussian distributions behavior. The minimum of the dihedral angles φ_{min} and the aspect ratio λ show slightly skewed distributions. Finally, the distributions for the mean ratio η and for the radius ratio ρ are the most skewed.

7.2 Ordering of Tetrahedron Shape Measures

Generally speaking, for each mesh, the average of the tetrahedron shape measures is ordered as following:

$$\eta > \rho > \gamma > \varphi_{min} > \tau > \theta_{min}. \tag{13}$$

Liu and Joe [40] had already noticed that tetrahedra have usually the $\eta > \rho > \sigma_{min}$ tetrahedron shape measures ordering. For example, this means that the mean ratio η is usually higher than the minimum of the solid angles θ_{min} for the same tetrahedron. You can change this behavior by using the square or the cube of η and the square root or the cubic root of θ_{min}. This also means that a mesh with an θ_{min} average of 0.8 is probably more optimized that a mesh with an η average of 0.9. Unfortunately, this also means that it is impossible to evaluate the quality of a mesh if the only available measure is an unusual tetrahedron shape measure.

7.3 Average of Tetrahedron Shape Measures

Even though the standard deviation is less significant for skewed distributions, the average is always meaningful. Considering the average of the tetrahedron shape measures one by one, the order of the results are always the same with few permutations. For example, the mesh of the unit tetrahedron test case with $1/5$ gets the highest average for five tetrahedron shape measures over six, while the mesh of the unit sphere with $1/5$ gets the lowest average for all six tetrahedron shape measures. So, a mesh optimized for one tetrahedron shape measure is also optimal or close to optimal for the other tetrahedron shape measures. A bad mesh for one tetrahedron shape measure is also a bad mesh for the other tetrahedron shape measures. These observations corroborate the conjuncture of § 4.

If the meshes are ordered according to the average of the tetrahedron shape measure averages, one can get the following order between the meshes: (unit tetrahedron with $1/5$) > (unit cube with $1/10$) > (unit tetrahedron with $1/10$) > (unit cube with $1/5$) > (unit sphere with $1/2.5$) > (unit sphere with $1/5$) (respectively $0.8253 > 0.8082 > 0.8040 > 0.7940 > 0.7909 > 0.7745$).

7.4 Standard Deviation of Edge Length

If the meshes are ordered according to the normalized standard deviation of the edge lengths, one can get the following order between the meshes: (unit tetrahedron with $1/5$) < (unit cube with $1/10$) < (unit tetrahedron with $1/10$) < (unit sphere with $1/2.5$) < (unit cube with $1/5$) < (unit sphere with $1/5$) (respectively $0.0945 < 0.1169 < 0.1175 < 0.1219 < 0.1250 < 0.1406$). This order is the same as the previous one with only one permutation. So, there is a kind of relation between the quality of the tetrahedra and the quality of the distribution of the edge lengths. This is not surprising, since tetrahedra that are close to regular have edges that are close to the same length.

The conclusion is that the average of any tetrahedron shape measure or the normalized standard deviation of the edge lengths are reasonable choice to characterized a mesh optimized with a valid tetrahedron shape measure.

7.5 Minimum of a Tetrahedron Shape Measures

Another measure used to characterize a mesh is the minimum of the tetrahedron shape measures. As the tetrahedron shape measures behave more or less as a Gaussian distribution, the best parameters to characterize the tetrahedron shape measure of all the tetrahedra of the mesh is clearly the average and the standard deviation. The minimum is less important and the maximum is close to useless because it is always one. From a statistical point of view, it is bad to characterize a whole mesh composed of millions of tetrahedra with only one of them. Especially, in complex geometries, the worst tetrahedra are often on the boundaries and may be imposed by a sharp feature or a corner. In this case, if it is just the minimum shape measure that matters, it is impossible to improve the mesh or specify the difference between two meshes.

Even though the strength of a chain is equal to the strength of the weakest link, this is not true in 2D or 3D. Just imagine a very regular mesh with only one very bad triangle or tetrahedron. The solution computed on this mesh may be better than the solution computed on an overall bad mesh without a very bad triangle or tetrahedron. Also, the very bad triangle or tetrahedron may be at a location were the solution is perfectly smooth and being totally harmless. On the other hand, a regular triangle or tetrahedron ten times bigger than the prescribed size and located exactly at the wrong place, can be very harmful for the solver. This emphasizes the fact that tetrahedron shape measures are dimensionless. So, our opinion is that, using principally the minimum shape measure to characterize the mesh is not a good choice.

7.6 Standard Deviation of the Edge Length in the Metric

For the unit cube with a non-uniform target length, the statistics (Table 7) and the histograms (Fig. 13.(a)) about edge lengths and tetrahedral volumes are more or less meaningless. As explained in § 2.3.4, the length of each edge may be evaluated in the metric deduced from the non-uniform target length with Eq. (2). Statistics on the metric lengths are more meaningful [8, 12, 31, 32, 35, 34]. The minimum is 0.7262, the average is 1.0344, the maximum is 1.3789 and the standard deviation is 0.1280. These results are close to those obtained on the same geometry but with constant target lengths ($1/5$ and $1/10$). The histogram for the normalized edge lengths for these two test cases and the histogram for the metric lengths of the edges for the test case with non-uniform target length are shown in Fig. 14.

The statistics and histograms on edge lengths for the three test cases on the unit cube show that the meshes had reached approximatively the same level of optimization. It is more difficult to study statistics on tetrahedral volumes because volume in a generic metric can only be evaluated with a quadrature. The difficulty is the same for evaluating in a generic metric the shape measure of the tetrahedra because it usually used the volume. For this problem, the approach of Berzins [6] may be considered. Remember that there is a relation between the shape measure of the tetrahedra and the normalized standard deviation of the distribution of the edge lengths. So, the statistics on the metric edge lengths is the an easy way to quantify the quality of a mesh when the target edge length is a generic non-uniform anisotropic

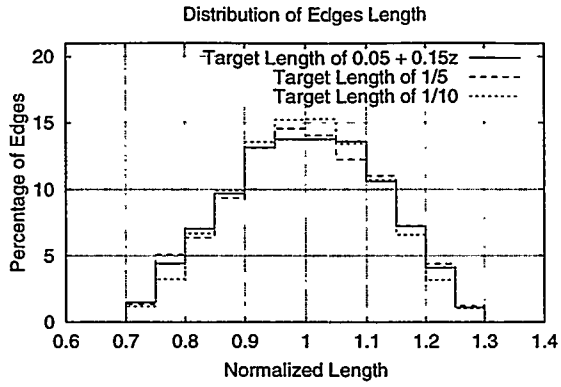


Figure 14: For the unit cube test cases, distribution of normalized edge lengths with a target edge length of $1/5$ and $1/10$ and distribution of the metric edge lengths with a target edge length of $0.05 + 0.15z$.

length. As the average is usually close to one, the key value to characterize the mesh is the normalized standard deviation and it is set to be the residual in the ADP3D mesh optimizer.

8 Conclusion

This paper introduced a series of benchmark test cases (§ 2), for the evaluation of 3D unstructured tetrahedral mesh optimization methods. In doing so, many difficulties were met. Mesh optimization is not already a well define subject and a definitions were given in § 5. Standardization of the measure used to quantify the quality of the meshes was proposed (§ 3). These measures include tetrahedron shape measures (§ 4). A review of published tetrahedron shape measures was made which led to a definition that characterized valid shape measures and the relationship that exists between them. The systematic use of the ADP3D mesh optimizer (§ 5) on these benchmarks (§ 6) provided an opportunity to deduce many interesting facts (§ 7). The most important conclusion is that the problem of mesh optimization is less a problem of tetrahedron shape measures than a problem of combining local modification techniques in a converging process.

While being very simple, the proposed test cases attempt to encompass difficulties met in real world. We emphasize the importance of the contribution to this benchmarking process from other researchers. For ease of comparison, the meshes for the benchmarks and the data, as well as simple programs that compute all the tetrahedron shape measures and that compute the histograms for plotting with `gnuplot`, are available on the web site `http://www.cerca.umontreal.ca/adp3d`.

References

[1] *Fourth International Meshing Roundtable*, Albuquerque, New Mexico, October 1995. Sandia National Laboratories.

[2] *Fifth International Meshing Roundtable*, Pittsburgh, Pennsylvania, October 1996. Sandia National Laboratories.

[3] *Sixth International Meshing Roundtable*, Park City, Utah, October 1997. Sandia National Laboratories.

[4] D. Aït-Ali-Yahia, W. G. Habashi, A. Tam, M.-G. Vallet, and M. Fortin. A directionally adaptive methodology using an edge-based error estimate on quadrilateral grids. *Int. J. Num. Meth. Fluids*, 23:673–690, 1996.

[5] T. Baker. Mesh adaptation strategies for problems in fluid dynamics. *Finite Elem. in Anal. and Design*, 1997.

[6] M. Berzins. Solution-based triangular and tetrahedral mesh quality indicator. *SIAM J. Sci. Comp.*, 19(6):2051–2060, 1998.

[7] H. Borouchaki and P. J. Frey. Adaptive triangular-quadrilateral mesh generation. *Int. J. Num. Meth. Engng*, 41:915–934, 1998.

[8] H. Borouchaki, P. J. Frey, and P.-L. George. Unstructured triangular-quadrilateral mesh generation. Application to surface meshing. In *Fifth International Meshing Roundtable* [2].

[9] H. Borouchaki and P.-L. George. Aspects of 2-D Delaunay mesh generation. *Int. J. Num. Meth. Engng*, 40:1957–1975, 1997.

[10] H. Borouchaki, P.-L. George, F. Hecht, P. Laug, and E. Saltel. Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie I: Algorithmes. Technical Report 2741, Institut National de Recherche en Informatique et en Automatique, France, December 1995.

[11] H. Borouchaki, P.-L. George, F. Hecht, P. Laug, and E. Saltel. Delaunay mesh generation governed by metric specification. Part I. Algorithms. *Finite Elem. in Anal. and Design*, 25:61–83, 1997.

[12] H. Borouchaki, P.-L. George, and B. Mohammadi. Delaunay mesh generation governed by metric specification. Part II. Applications. *Finite Elem. in Anal. and Design*, 25:85–109, 1997.

[13] H. Borouchaki, F. Hecht, and P. J. Frey. H-correction. Technical Report 3199, Institut National de Recherche en Informatique et en Automatique, France, June 1997.

[14] H. Borouchaki, F. Hecht, and P. J. Frey. Mesh gradation control. In *Sixth International Meshing Roundtable* [3].

[15] H. Borouchaki, F. Hecht, E. Saltel, and P.-L. George. Reasonably efficient Delaunay based mesh generator in 3 dimensions. In *Fourth International Meshing Roundtable* [1], pages 3–14.

[16] G. C. Buscaglia and E. A. Dari. Anisotropic mesh optimization and its application in adaptivity. *Int. J. Num. Meth. Engng*, 40:4119–4136, 1997.

[17] M. J. Castro-Díaz, F. Hecht, and B. Mohammadi. New progress in anisotropic grid adaptation for inviscid and viscous flows simulations. In *Fourth International Meshing Roundtable* [1], pages 73–85.

[18] M. J. Castro-Díaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaptation for flow simulation. *Int. J. Num. Meth. Fluids*, 25:475–491, 1997.

[19] C. L. Chen, K. Y. Szema, and S. R. Chakravarthy. Optimization of unstructured grid. In *33rd Aerospace Sciences Meeting and Exhibit*, number AIAA–95–0217, Reno, NV, January 1995. AIAA.

[20] T. Coupez. *Grandes transformations et remaillage automatique.* PhD thesis, École Nationale Supérieure des Mines de Paris, November 1991.

[21] J. Dompierre, M.-G. Vallet, M. Fortin, W. G. Habashi, D. Aït-Ali-Yahia, S. Boivin, Y. Bourgault, and A. Tam. Edge-based mesh adaptation for CFD. In *Conference on Numerical Methods for the Euler and Navier-Stokes Equations*, pages 265–299, Montréal, September 1995. CRM-CERCA.

[22] M. Fortin, M.-G. Vallet, D. Poirier, and W. G. Habashi. Error estimation and directionally adaptive meshing. In *25th AIAA Fluid Dynamics Conference*, number AIAA–94–2211, Colorado Springs, CO, June 1994.

[23] L. Freitag, M. Jones, and P. Plassmann. An efficient parallel algorithm for mesh smoothing. [1], pages 47–58.

[24] L. A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *Int. J. Num. Meth. Engng*, 40:3979–4002, 1997.

[25] P. J. Frey and H. Borouchaki. Surface mesh evaluation. In *Sixth International Meshing Roundtable* [3].

[26] P. J. Frey, H. Borouchaki, and P.-L. George. Delaunay tetrahedralization using an advancing-front approach. In *Fifth International Meshing Roundtable* [2].

[27] P. J. Frey, H. Borouchaki, and P.-L. George. Tétraédrisation de Delaunay basée sur une approche frontale. Technical Report 2882, Institut National de Recherche en Informatique et en Automatique, France, May 1996.

[28] J. W. Gaddum. The sums of the dihedral and trihedral angles in a tetrahedron. *Amer. Math. Monthly*, 59:370–371, 1952.

[29] P.-L. George. Génération de maillages par une méthode de type Voronoï. Partie 2: Le cas tridimensionnel. Technical Report 1664, Institut National de Recherche en Informatique et en Automatique, France, April 1992.

[30] P. L. George. Improvements on Delaunay-based three-dimensional automatic mesh generator. *Finite Elem. in Anal. and Design*, 25(3):297–317, 1997.

[31] P.-L. George and H. Borouchaki. *Triangulation de Delaunay et maillage, applications aux éléments finis.* Hermès, Paris, 1997.

[32] P.-L. George and H. Borouchaki. *Delaunay Triangulation and Meshing. Applications to Finite Elements.* Hermès, Paris, 1998.

[33] N. A. Golias and R. W. Dutton. Delaunay triangulation and 3D adaptive mesh generation. *Finite Elem. in Anal. and Design*, 25:331–341, 1997.

[34] W. G. Habashi, J. Dompierre, Y. Bourgault, M. Fortin, and M.-G. Vallet. Certifiable computational fluid dynamics through mesh optimization. *AIAA J.*, 36(5):703–711, May 1998.

[35] W. G. Habashi, M. Fortin, J. Dompierre, M.-G. Vallet, D. Aït-Ali-Yahia, Y. Bourgault, M. P. Robichaud, A. Tam, and S. Boivin. Anisotropic mesh optimization for structured and unstructured meshes. In *28th Computational Fluid Dynamics Lecture Series*. von Karman Institute, von Karman Institute for Fluid Dynamics, March 1997.

[36] D. F. Hawken, J. J. Gottlieb, and J. S. Hansen. Review of some adaptive node-movement techniques in finite-element and finite-difference solutions of partial differential equations. *J. Comp. Phys.*, 95(2):254–302, August 1991.

[37] F. Hecht and B. Mohammadi. Mesh adaption by metric control for multi-scale phenomena and turbulence. In *AIAA 35th Aerospace Sciences Meeting & Exhibit*, number AIAA–97–0859, Reno, NV, January 1997.

[38] B. Joe. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM J. Sci. Comp.*, 16(6):1292–1307, 1995.

[39] A. Liu and B. Joe. On the shape of tetrahedra from bisection. *Math. of Comp.*, 63(207):141–154, 1994.

[40] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *Bit*, 34:268–287, 1994.

[41] A. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on bisection. *SIAM J. Sci. Comp.*, 16:1269–1291, 1995.

[42] A. Liu and B. Joe. Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision. *Math. of Comp.*, 65(215):1183–1200, July 1996.

[43] S. H. Lo. Optimization of tetrahedral meshes based on element shape measures. *Comp. and Struct.*, 63(5):951–961, 1997.

[44] K. Nakahashi and G. S. Deiwert. Three-dimensional adaptive grid method. *AIAA J.*, 24(6):948–954, June 1985.

[45] K. Nakahashi and G. S. Deiwert. Self-adaptive grid method with application to airfoil flow. *AIAA J.*, 25(4):513–520, 1987.

[46] C. F. Ollivier-Gooch. An unstructured mesh improvement toolkit with application to mesh improvement, generation and (de-)refinement. In *AIAA 36th Aerospace Sciences Meeting & Exhibit*, number AIAA–98–0218, Reno, NV, January 1998.

[47] B. Palmerio. A two-dimensional FEM adaptive moving-node method for steady Euler flow simulation. *Comp. Meth. in Appl. Mech. and Engng*, 71:315–340, 1988.

[48] B. Palmerio. An attraction-repulsion mesh adaption model for flow solution on unstructured grids. *Comp. and Fluids*, 23(3):487–506, 1994.

[49] B. Palmerio. Coupling mesh and flow in viscous fluid calculations when using unstructured triangular Finite Elements. *Int. J. Comp. Fluid Dyn.*, 1996.

[50] V. N. Parthasarathy, C. M. Graichen, and A. F. Hathaway. A comparison of tetrahedron quality measure. *Finite Elem. in Anal. and Design*, 15:255–261, 1993.

[51] V. T. Rajan. Optimality of the Delaunay triangulation in R^d. *Disc. & Comp. Geometry*, 12:189–202, 1994.

[52] A. Rassineux. 3D mesh adaptation. Optimization of tetrahedral meshes by advancing front techniques. *Comp. Meth. in Appl. Mech. and Engng*, 141(3):335–354, 1997.

[53] A. Rassineux. Generation and optimization of tetrahedral meshes by advancing front technique. *Int. J. Num. Meth. Engng*, 41:651–674, 1998.

[54] A. Tam. *An Anisotropic Adaptive Method for thr Solution of 3-D Inviscid and Viscous Compressible Flows*. PhD thesis, Concordia University, Montréal, April 1998.

[55] M.-G. Vallet. *Génération de maillages éléments finis anisotropes et adaptatifs*. PhD thesis, Université Pierre et Marie Curie, Paris VI, France, 1992.

[56] P. D. Zavattieri, E. A. Dari, and G. C. Buscaglia. Optimization strategies in unstructured mesh generation. *Int. J. Num. Meth. Engng*, 39:2055–2071, 1996.

[57] H. Zhang and J.-Y. Trépanier. An algorithm for the optimization of directionally stretched triangulations. *Int. J. Num. Meth. Engng*, 37:1481–1497, 1994.

An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes

Scott A. Canann, Joseph R. Tristano, and Matthew L. Staten
ANSYS, Inc.
275 Technology Drive
Canonsburg, PA 15317
{scott.canann, joe.tristano, matt.staten}@ansys.com

Abstract. Automatic finite element mesh generation techniques have become commonly used tools for the analysis of complex, real-world models. All of these methods can, however, create distorted and even unusable elements. Fortunately, several techniques exist which can take an existing mesh and improve its quality. Smoothing (also referred to as mesh relaxation) is one such method, which repositions nodal locations, so as to minimize element distortion. In this paper, an overall mesh smoothing scheme is presented for meshes consisting of triangular, quadrilateral, or mixed triangular and quadrilateral elements. This paper describes an efficient and robust combination of constrained Laplacian smoothing together with an optimization-based smoothing algorithm. The smoothing algorithms have been implemented in ANSYS® and performance times are presented along with several example models.

Keywords. Smoothing, Laplacian smoothing, optimization-based smoothing, triangular, quadrilateral, quad-dominant

1. Introduction

1.1 Importance of work

As mesh generation becomes more automated and as mesh sizes increase, the need to create meshes completely free of unusable or geometrically distorted elements increases. That is, for isotropic solutions, geometrically distorted elements give poor solutions and/or ill-conditioned matrices. For large meshes, visually checking the quality of a mesh can be difficult at best, not to mention error-prone. In order to minimize the user time required to validate that the mesh is acceptable, a priori distortion metrics and automatic correction procedures are needed.

Geometrically distorted elements can be caused by many things, including:

- Model size -- coarse meshes, by their very nature, can result in geometrically distorted elements;
- Tough topological and geometrical configurations common in real-world CAD geometries, including sliver areas, sharp corners, small areas, small holes, thin sections, and areas of high curvature; and
- Algorithmic flaws in the meshing algorithm.

A variety of mesh improvement techniques has been developed to improve the quality of meshes created by automatic techniques. Some of the existing techniques for improving the quality of an existing surface mesh include:

- Topological and quality-based operators – including:
  - Node insertion or local refinement techniques [4,38,46,56,57,70];
  - Edge/face swaps [2,15,16,17,22,26,37,38,44 ,70,73]; and
  - Node removal or element deletion [9-11,15-17,44,70,73].

- Smoothing – modifying node placement so as to improve the elements shape without modifying the mesh connectivity. This can be done for:
  - Corner nodes [1,5,7,14,21,23,25,27,29,30,32,34,35,39-41,55,72]; or
  - Midnodes [62-65] – where the corner nodes of quadratic elements are held fixed and the mid-nodes are repositioned, so as to improve the element's quality.

While ANSYS uses a combination of the techniques listed above to improve the quality of a mesh, this paper will only cover the implementation of the smoothing portion of the overall mesh improvement process.

Since smoothing is independent of the mesh generation technique used to create a mesh, the ideas discussed in this paper can be used to compliment any mesh generation technique.

In addition to being used as an improvement step after creating a mesh, smoothing can also be used to:

- Smooth local collections of elements between steps in an advancing front meshing approach [9-11,52];
- Locally improve elements after each quality-based mesh cleanup operation [2,9-11,26,37,70,73], such as node insertions, deletions and diagonal swaps;
- Improve a mesh after local mesh refinement [4,38,70]; and
- Aid in shape optimization [42].

1.2 Previous work

A significant amount of work has been done in the area of mesh smoothing. This section will present an overview of some of that work, including in the areas of Laplacian smoothing, optimization-based smoothing, and physics-based smoothing. In addition to the overview of smoothing given here, another overview can be found in George [31,32].

1.2.1 Laplacian Smoothing

Laplacian smoothing is by far the most common smoothing technique. Laplacian smoothing, in its simplest form, consists of recursively placing each node at the average of the nodes connected to it. This technique generally works quite well for meshes in convex regions. However, it can result in distorted or even inverted elements near concavities in the model. Many researchers have used and extended the capabilities of Laplacian smoothing [9,10,11,13,18,23,30,32,34,35,39,40]

Some variations on the basic Laplacian technique, include:

- Weighting the contribution of each neighboring node in the averaging function [9,10,11,32,40] by edge length, element area, or other similar criteria;
- Constraining the node movement, so as to avoid the creation of inverted elements [18,23];
- Developing methods to extend it to anisotropic (stretched) meshes [13,34]; and
- Generating a means of combining it with the similar *isoparametric* smoothing technique [35].

1.2.2 Optimization-based Smoothing

A newer form of smoothing, that is receiving more attention lately, is optimization-based smoothing. Instead of moving nodes based on a heuristic algorithm, as is done in Laplacian smoothing, the nodes are moved so as to minimize a given distortion metric. Some of the developments in this area include [1,5,7,14,21,25,27-29,36,54,55]. While optimization-based smoothing is more expensive than Laplacian smoothing, it gives better results – especially near concave regions in the geometry. Several authors [21,27] (including this paper) have implemented schemes that use Laplacian smoothing when possible and only use optimization-based smoothing when necessary.

One of the first optimization-based smoothing algorithms was developed by de Cougny [21]. His technique was designed to improve the distorted tetrahedral elements that can be created near the boundary of a tet mesh

generated using an octree technique [66]. An element distortion metric is presented that is basically the scaled ratio of an element's volume to it's face areas, for which he proves several properties, which indicate that the metric may be well-suited for optimization-based smoothing. He then finds the ideal location for the node to minimize the maximum distortion metric and does a search along the line from the current location to the potentially optimal location.

Parthasarathy [54] developed an optimization-based technique for triangular and tetrahedral meshes -- again created to repair elements created by quadtree and octree mesh generators. He solves a nonlinear, constrained, global optimization problem, using the element's aspect ratio as the objective function to be minimized. He also adds inequality constraints to ensure that none of the elements' surface area (or volume in 3D) drops below a certain threshold. He uses a modified version of the feasible directions algorithm to drive the optimization. Due to the fact that a global optimization was done, Parthasarathy found that as much time was spent in the smoothing process as was spent in the mesh creation phase. Encouraging results were obtained from the smoother.

Canann [14] developed a global optimization method that uses Oddy's distortion metric [51]. Although it was developed mainly for hex meshes, it can be easily extended to any other element type. This method, like that developed by Parthasarathy, is impractical for large meshes, because it uses global optimization. Recursive local optimization has proven to be more feasible. In addition, it was later determined that Oddy's metric was too lenient for angle distortion and too restrictive on aspect ratio distortion.

An approach developed by Freitag [7,25,27-29] works to maximize the minimum angle in triangular or tetrahedral meshes. Since this is a non-continuous function (i.e., the function does not have continuous derivatives defined everywhere), a *nonsmooth optimization* (NSO) is done, using an analogue of the steepest descent method for smooth functions. This approach has been shown to be parallelizable [25], gives good results, and is reasonably efficient. Freitag has also experimented with combining the approach with Laplacian smoothing in [27].

Amenta [1] presents theoretical results showing how some local triangle and tetrahedral shape optimizations can be solved in linear time using generalized linear programming. For the mesh smoothing problems that don't fit into that class of problems, other efficient algorithms are presented. Many distortion metrics are discussed and various optimization techniques are compared.

Other optimization-based methods include ones developed by Riccius [55], Bank [5], and Jacquotte [36]. An optimization-based smoothing algorithm, specifically designed for adaptively improving finite element triangulations by making use of a posteriori error estimates is presented by Bank [5]. Jacquotte [36] developed a distortion metric and optimization-based smoothing approach suitable for 2D & 3D structured grids.

In order to develop a robust optimization-based smoothing algorithm, much thought must go into selecting a good distortion metric. Among the many a priori distortion metrics that have been developed, a representative set can be found in [3,6,8,20,23,36,43,44,45,47-51,54,58-65,74].

1.2.3 Physics-based smoothing

Since a well-graded mesh can be similar to objects found in nature, several authors have developed techniques for smoothing that are based on solving simple physics problems. Lohner et al. [50] developed a smoothing algorithm that views the mesh as a system of springs between nodes, where the force between nodes is a ratio between the actual and desired grid sizes. His technique produces stretched elements based on density gradients. Shimada [67-69] and Bossen [12] developed mesh generation techniques that created the mesh, smoothed it, and cleaned it up, all as a part of a physics-based approach. Shimada viewed the nodes as being the center of bubbles, with each bubble interacting with each of the others. Bossen developed what he called a *pliant* method, which like Shimada's approach uses attraction and repulsion between nodes to determine nodal locations. Neither of these approaches is a true smoothing technique though, because both require node insertions or deletions when the mesh is not graded smoothly enough. Each of these physic-based techniques is capable of retaining stretched elements in an anisotropic mesh.

1.3 Paper Overview

In this work, a complete scheme is presented for smoothing 2D and 3D surface meshes. Some of the advantages include:

- Robustness – improves the mesh and never inverts elements;
- Efficiency – combines constrained Laplacian with selective optimization-based smoothing;
- Ability to repair inverted elements; and
- Ability to smooth 2D and 3D meshes comprised of:
  - triangles only [12,18,20,30-32,48,67-69];
  - quadrilaterals only [9-11,19,53,73];
  - mixed triangles and quadrilaterals [18,30,44,49,52];

Section 2 lists the requirements for the smoother. Section 3 lays out the overall mesh smoothing algorithm. Section 4 discusses the details of the constrained Laplacian smoothing implementation. Section 5 describes the basic optimization-based smoothing algorithm. Section 6 presents the distortion metrics used for both the constrained Laplacian smoothing and the optimization-based smoothing. Section 7 shows some speed and quality results, including a few sample meshes using the presented smoothing scheme. A conclusion is then given in Section 8.

2. Smoothing Requirements

The smoothing algorithms described in this paper were designed for use in the ANSYS® software package. Some of the algorithmic requirements for the smoother include that it:

- Work for triangular, quadrilateral, and quad-dominant meshes, created by any of several mesh generation and refinement techniques;
- Handle severely distorted elements;
- That it be efficient and robust; and
- Be tested on a wide range of models and across several types of physics.

ANSYS creates triangular, all-quadrilateral, and quad-dominant meshes and the smoother must be able to successfully smooth these meshes, without giving undue preference to one element shape over another. Smoothing must also be developed independently from any mesh creation technique, so that is can be used by ANSYS' various triangular mesh generators [18,44,71,75], quad-dominant mesh generators [18,51,75], and mesh refinement capabilities [70,76].

Cases arise where it is necessary to improve severely distorted or even mildly inverted elements with the smoother. This distortion may have been caused by the meshing algorithm, mesh refinement, or a mesh cleanup operation. An example of a case where this type of distortion can be routinely created is when performing an element open (a node insertion) in a quadrilateral mesh as a part of a topological cleanup process [15,16,70]. Sometimes the insertion of such an element can create a very geometrically distorted quad, as shown in Figure 1, which the smoother must then fix.

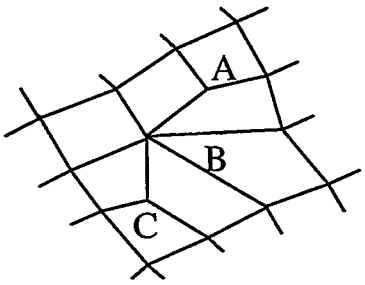


Figure 1a. Before element open

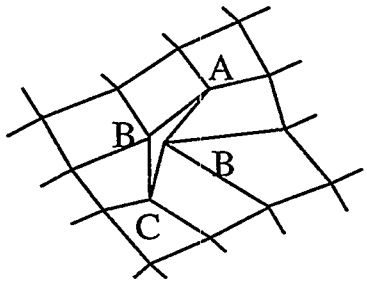


Figure 1b. After element open

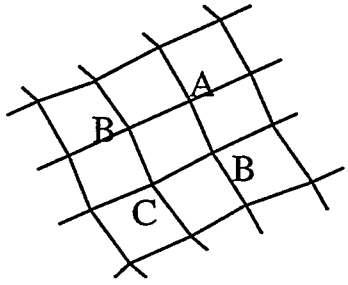


Figure 1c. After smooth

3. Overall Smoothing Algorithm

In this section, the overall smoothing scheme is presented. Algorithmic details for each of the major steps in this scheme will be presented later in the paper.

- Compute initial distortion metrics for all elements.
- If mesh is a layer mesh (see section 4.3), mark the nodes associated with the layer boundary
- Based on model size, compute distance, δ, for numerical gradient computation in optimization based smoothing (see section 5.1)
- Set number of iterations (*niter*) to 1
- REPEAT (MAIN SMOOTHING LOOP):
    - FOR EACH NODE *v* DO:
        1. If node *v* is not movable or has been deactivated from the smooth, then:
            - Break from loop -- go on to the next node in the list
        2. If node *v* wasn't moved by optimization-based smoothing in a previous iteration, then do **Constrained Laplacian Smoothing:**
            a) If node *v* is on layer boundary (see section 4.3):
                - Move node *v* using Laplacian smoothing
                - Project to layer boundary

                Else:
                - Perform a constrained Laplacian smooth -- see section 4.1 and 4.2

            b) If distance moved is less than the move tolerance:
                - Keep the old position for the node
                - Remove node *v* from the list of active smoothable nodes

                Else allow the move:
                - Put the node's neighbors back in the list of smoothable nodes, if they aren't already there
                - Compare to the largest distance moved by a node and keep track of the largest
                - Update the adjacent elements' distortion metric values
        3. If *niter* $\geq$ 2, then invoke **Optimization-based smoothing**. That is, let the Laplacian smoothing have 2 iterations to improve the mesh, before invoking the more expensive optimization-based smoothing):
            a) Find the minimum distortion metric from all facets connected to the node
            b) If the minimum metric <= a given tolerance (e.g., 0.1)
                - Attempt optimization-based smoothing – See section 5 for details
            c) If optimization-based smoothing moved the node, then:
                - Put the node's neighbors back into the smoothable list, if not already there
                - Compare to the largest distance moved by a node and keep track of the largest
                - Update the adjacent elements' distortion metric values
    - END DO (end of loop through nodes)
    - niter = niter + 1
- UNTIL there are no more nodes that have moved enough to warrant another iteration. That is, if NO nodes have moved OR if the maximum distance moved is less than (1.75 * move tolerance). (END OF MAIN SMOOTHING LOOP)

4. Laplacian Smoothing

Basic Laplacian smoothing iteratively repositions each node to be at the average of each of the nodes connected to it. This works extremely well for convex regions. However, it can invert elements near concavities. In order to avoid this, constraints can be placed on the smoother, so that it doesn't do more damage than good. In this section a description of a constrained Laplacian smoothing implementation is given.

4.1 Constrained Laplacian Smoothing Algorithm

The algorithm used for each node entails:

- Computing the location where Laplacian smoothing would place the node & projected new nodal location to its corresponding curve or area
- Loop (repeat up to the arbitrarily selected value of 20 times):
  - Compute distortion metric for neighboring elements
  - If the new location is "acceptable" (see Section 4.2 below), then break out of the loop
  - If the new location is not "acceptable", then cut the proposed move distance in half and set this as the new location

4.2 Acceptance Criteria for Constrained Laplacian Node Movement

Before determining if a new location is acceptable, the following quantities are computed for each element connected to the node:

- μ (i.e., α or β) – Distortion metrics for the elements connected to the node before and after the proposed node movement. (Note that for efficiency, the elements' distortion metrics from the before the move can be stored with each element).
- N+ – Number of elements whose metric improves
- N- – Number of elements whose metric gets worse
- $\Delta\mu = \Sigma\{\Delta\mu_i\} / N$ – Average metric change computed as the sum of distortion metrics changes at the connected elements divided by the number of connected elements
- $N\uparrow$ – Number of elements whose metric improves significantly:
  - Metric goes from negative to positive
  - Metric becomes less negative
  - Metric moves from below acceptable to acceptable quality (a value of $\mu_{min} = 0.05$ is reasonable for the minimal acceptable quality limit).
- $N\downarrow$ – Number of elements whose metric worsens significantly:
  - Metric goes from positive to negative
  - Metric becomes more negative
  - Metric drops below acceptable levels (even if it stays positive).
- $N\downarrow\downarrow$ – Number of inverted elements created. This is the same as $N\downarrow$ for triangles. However, a quad can go negative without becoming inverted. That is, the metric at a corner node goes negative as a corner angle goes past 180 degrees, but inversion only occurs when a corner angle drops below 0 degrees.
- θ -- The largest angle spanned between any two nodes in the element.

Node movement is ruled out if *any* of the following are true:

- (N- = N) – All of the elements get worse; or
- ($N\downarrow\downarrow > 0$) – Any of the elements go inverted; or
- ($N\downarrow > N\uparrow$) – More elements get significantly worse than get significantly better;
- ($\Delta\mu < -\mu_{min}$) – The average change in the metric is poor; or
- ($\theta > \theta_{max}$) – The element spans too great of an angle (θ_{max} = maximum angle any element can span).

If node movement is not ruled out by the checks about, then the node movement is considered to be acceptable if *any* of the following are true:

- $(N+ = N)$ – All of the elements have improved; or
- $(N\uparrow > 0)$ & $(N\downarrow = 0)$ – No elements get significantly worse and some get significantly better; or
- $(N\uparrow \geq N\downarrow)$ & $(\Delta\mu > -\mu_{min})$ – More elements get significantly better than get significantly worse and the average improvement is reasonable.

4.3 Boundary Layer Smoothing

ANSYS provides the ability to create 2D free meshes where the user can request a specified number of rows of very small elements next to a geometric edge and then transition quickly to a much larger element size [53, 75]. This type of mesh is particularly useful for CFD boundary layers or EMAG skin effect studies. See Figure 2 below.

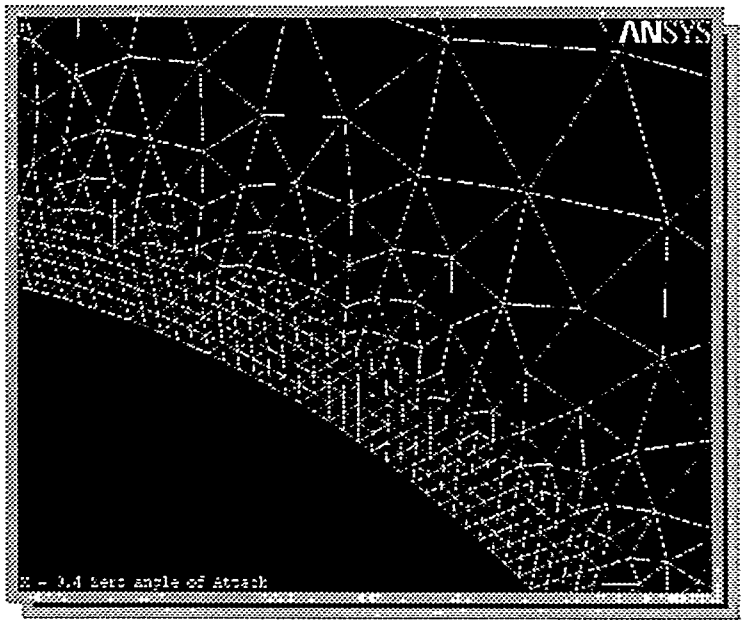


Figure 2. Layer mesh, with four rows of equally sized elements in the boundary layer.

In order to keep the smoother from stretching the elements in the boundary layer out into the rest of the mesh (thereby causing the transition to start at the boundary); several modifications must be added to the smoothing process. Before the smoothing iterations begin, the nodes in the boundary layer that are farthest from the geometric edge are marked. During the smoothing process, these nodes are constrained to remain the same constant boundary layer distance from the edge. This is done by allowing each outside layer node to be smoothed by the Laplacian smoother, projecting it to the geometric edge, and then moving it perpendicularly away from the geometric edge a distance equal to the boundary layer thickness.

5. Optimization-based Smoothing

In some cases – especially near concavities in the model – the constrained Laplacian smoother is unable to improve severely distorted elements. It is often critical that an adequate smoother be invoked to repair these elements, because they will often occur near to the boundary of the mesh – where element shape is typically the most critical.

Optimization-based smoothing directly attacks the element distortion problem. Instead of applying a heuristic-based movement to each node as is done in Laplacian smoothing, optimization-based smoothing seeks to minimize

the distortion of the elements connected to each node. Existing optimization-based smoothing techniques [1,5,7,14,21,25,27-29,36,54,55] are relatively new and vary based on:

- The type of mesh being smoothed (structured or unstructured);
- The element shape (triangle, quadrilateral, etc.);
- The optimization and search technique chosen; and
- The distortion metric selected (see Section 6 below).

5.1 Optimization-based Smoothing Algorithm

In this section, the algorithmic details of a new optimization-based smoothing technique are presented. While only 2D elements (triangles and quadrilaterals) are covered in this paper, the techniques described here can be (and have been successfully) extended to 3D volume elements. The approach presented here is similar to the Freitag's approach [25], but adds two advantages over her approach. The first advantage that this approach has over Freitag's is that it can repair meshes with elements that are severely distorted or even inverted. Freitag's approach requires that none of the elements be severely distorted or inverted. In other words, the distortion metrics used are continuous through element distortion and inversion. The other advantage to this approach over Freitag's approach is that her method has only been implemented for simplicial elements (triangles and tetrahedron). The approach presented here works for triangular, quadrilateral, and mixed-meshes.

Let μ be a distortion metric, with values generally varying between -1 and 1. For example, μ could be the triangle metric, α, or the quadrilateral metric, β presented in Section 6 below. A value of 1 for μ corresponds to a perfectly shaped element, while a value of 0 corresponds to a element that is basically unusable[1]. Negative values correspond to extremely distorted elements, including negatively oriented or inverted elements.

Let $\mathbf{x}$ be a node in the mesh, connected to elements $E_1, E_2, \ldots, E_n$, with (positive) metric values $\mu_1, \mu_2, \ldots, \mu_n$, respectively. In an optimization-based smoothing procedure, the goal is to move $\mathbf{x}$ so that $\mu_{min} = \min(\mu_1, \mu_2, \ldots, \mu_n)$ is increased as much as possible, i.e., the quality of the elements incident on $\mathbf{x}$ is optimized. If $\mathbf{x}$ lies on a boundary edge (possibly curved), then the movement of $\mathbf{x}$ is constrained to lie on that edge. Since optimization-based smoothing is more time-consuming than Laplacian smoothing, it will typically only be invoked for nodes whose value of μ_{min} is sufficiently small.

A steepest descent iterative approach is used to move $\mathbf{x}$ in gradient directions so as to increase μ_{min} as much as possible. Usually, the improvement in μ_{min} decreases with each iteration, so two iterations are usually sufficient in practice. The following is done in an iteration of the steepest descent method:

1. Estimate the gradient vector $\mathbf{g}_i = (g_x^i, g_y^i, g_z^i)$ for each element E_i, $1 \le i \le n$, as follows:
   - Perturb the x-coordinate of $\mathbf{x}$ by δ. The size of δ can be based on the size of vertex coordinates, e.g., 10^{-5} times the maximum model dimension.
   - Compute the perturbed metric value μ_i^+ for E_i.
   - Set $g_x^i = (\mu_i^+ - \mu_i)/\delta$. ($g_y^i$ and g_z^i can be computed similarly.)

2. The gradient direction is taken to be $\mathbf{g} = \mathbf{g}_m$, where m is an index such that $\mu_m = \mu_{min}$. If $\mathbf{g} \approx 0$, then let $\mathbf{g} = \mathbf{g}_{m^+}$, where m^+ is an index, such that $\mu_{m^+} = \mu_{min^+}$, where μ_{min^+} is the next smallest element metric to μ_{min}. Vertex $\mathbf{x}$ is to be moved to $\mathbf{x}^+ = \mathbf{x} + \gamma\mathbf{g}$, where the scalar γ is determined as follows:

   - If μ_i is considered to be a function of $\mathbf{x}$, then the Taylor series expansion yields:

$$\mu_i(\mathbf{x}^+) = \mu_i(\mathbf{x}+\gamma\mathbf{g}) \approx \mu_i(\mathbf{x}) + \gamma\,\mathbf{g} \cdot \mathbf{g}_i \qquad [1]$$

[1] For example, this could be a zero area triangle or a quad with a 180 degree corner angle.

If $\mathbf{g} \cdot \mathbf{g}_i \geq 0$, then μ_i *increases* for a sufficiently small γ. That is, the chosen gradient direction will most likely improve the quality for E_i and we probably won't have to worry about μ_i^+ being less than μ_{min}^+.

- On the other hand, if $\mathbf{g} \cdot \mathbf{g}_i < 0$, then μ_i *decreases* for a sufficiently small γ in the chosen gradient direction. In this case, the value of γ needs to be restricted, so that $\mu_{min}^+ \geq \mu_i^+$ -- or:

$$\mu_{min} + \gamma\, \mathbf{g} \cdot \mathbf{g} \geq \mu_i + \gamma\, \mathbf{g} \cdot \mathbf{g}_i \qquad [2]$$

Therefore, γ is limited to be:

$$\gamma = \min\left\{\frac{(\mu_i - \mu_{min})}{(\mathbf{g}\cdot\mathbf{g} - \mathbf{g}\cdot\mathbf{g}_i)}\right\}, \text{ over the indices, } i, \text{ for which } \mathbf{g} \cdot \mathbf{g}_i < 0 \qquad [3]$$

3. Once the value for γ is obtained and it is sufficiently large, then a move of $\mathbf{x}$ to $\mathbf{x}^+ = \mathbf{x} + \gamma\mathbf{g}$ is attempted as follows:

   - Using the new $\mathbf{x}^+$, the new metric values μ_i^+ are computed for each E_i.
   - The new minimum metric value, μ_{min}^+, is computed as:

$$\mu_{min}^+ = \min(\mu_1^+, \mu_2^+, \ldots, \mu_n^+) \qquad [4]$$

   - The move of $\mathbf{x}$ to $\mathbf{x}^+$ is accepted and μ_i is set to μ_i^+ for each i, if:

$$\mu_{min}^+ \geq \mu_{min} + \text{tol} \qquad [5]$$

   Where "tol" is a tolerance, e.g., 10^{-5}.

   - Since approximations are used, it is possible that the original γ is too large, so if the move is not accepted, then a smaller movement can be tried. The value γ is decreased by a factor of 2, and the above is repeated for the new $\mathbf{x}^+$. At most a fixed number, say 4, of γ decreases is attempted before giving up on moving $\mathbf{x}$.

6. Distortion Metrics

As explained in the previous sections, distortion metrics can be used to do more than just determine the final quality of a mesh. They can also be used to guide smoothing operations by constraining the movement during constrained Laplacian smoothing and to drive the optimization-based smoothing. The selection of a suitable metric is critical. Some of the criteria by which a metric can be judged include:

- Efficiency – it will be called many times
- Suitability for use by an optimization-based smoothing algorithm, i.e.:
  - Continuous, since derivatives are needed to determine the direction which minimizes the metric.
  - Precise in its description of what element distortion is. Nothing can find a flaw in a metric faster than optimization-based smoothing.
  - Preferably, it will remain defined through inversion of the element, so that mildly inverted elements can be repaired.
- Application to mixed meshes. It is advantageous for the metric to be defined and normalized in such a way that the metrics for each element type can be used together. That is, the quad and tri distortion

metrics should be compatible, so that smoothing of quad-dominant meshes does not over-emphasize the quality of one element shape over another at a given node.

The following sections will describe the distortion metrics that have been developed for quantifying triangle and quadrilateral element distortion.

6.1 Triangular Distortion Metric

The distortion metric used for triangle elements is an extension of the one presented by Lee and Lo in [44]:

$$\alpha(ABC) = (I)2\sqrt{3}\frac{\|CA \times CB\|}{\left[\|CA\|^2 + \|AB\|^2 + \|BC\|^2\right]}$$

where : [6]

$$I = \begin{Bmatrix} 1, (CA \times CB) \cdot N_x > 0 \\ -1, (CA \times CB) \cdot N_s < 0 \end{Bmatrix}$$

Where N is the surface normal evaluated at the center of the triangle; and AB, CB, and CA are the edge lengths between those nodes. Note that the metric is signed (with "I") to capture inversion of the element. This distortion metric is basically the area of the triangle divided by the sums of the squares of the lengths of the sides. The $2\sqrt{3}$ term is a normalizing factor so that equilateral triangles will have a maximum α value of 1.

The only change between this metric and Lee's metric, is the inclusion of the "*I*" term, which assigns a negative distortion value to inverted elements. This is important for helping to determine if the proposed Laplacian smoothing node movement will indeed improve the quality of the local mesh. Optimization-based smoothing also gains form a distortion metric that continuously quantifies element distortion as an element goes inverted. This modification makes it possible for the smoother to repair mildly inverted elements.

6.2 Quadrilateral Distortion Metric

The metric used for the quadrilaterals is also based on the metric presented by Lee and Lo in [44], again with a few modifications. The distortion metric, β, is computed by dividing the quadrilateral into triangles along each of its diagonals. The distortion metric, α, (described above) is computed for each of the four resulting triangles and is then sorted in descending order of magnitude. That is:

$$\alpha_1 \geq \alpha_2 \geq \alpha_3 \geq \alpha_4 \quad [7]$$

Lee [44] uses the following as his metric:

$$\beta_{Lee} = \left(\frac{\alpha_3 \alpha_4}{\alpha_1 \alpha_2}\right) \quad [8]$$

Instead of Lee's metric, the distortion metric used here is:

$$\beta = \{\min(\alpha_1, \alpha_2, \alpha_3, \alpha_4)\} - negval \quad [9]$$

Where *negval* = 1 if any of the angles corner angles of the quad are less than 6 degrees, any two of the nodes are coincident within a tolerance, or two of the triangles are inverted (i.e., their α_i's are negative). In addition *negval* is set to 2 or 3, if 3 or 4 α_i's are negative, respectively. This constant allows for a subjective qualification of whether a node movement really improves the quality of the mesh. The use of *negval* here is certainly heuristic, but it has been developed over several years of experiences in smoothing thousands of area meshes.

While this is not a continuous function, it has proven to be effective for both types of smoothing. For optimization-based smoothing, g is correct, but the correct value for γ may require additional iterations, when $\mu_{min} < 0$.

This metric is used, instead of equation 8 above, because negative metric values are now allowed and if two or four α's are negative (inverted), then the β value will be incorrectly signed. The new metric also emphasizes improving minimum metric values over average metric values and analyses are more affected by minimum metrics than by low average metric values. In addition, Lee's metric does not detect aspect ratio problems. His metric will give perfect distortion metric values to long, stretched rectangles.

7. Results and Examples

7.1 Smoothing improvement statistics

The mesh smoothing scheme presented in this paper has been exhaustively tested and has proven to be sufficiently robust and efficient for use in a commercial product. The following distortion metric improvement statistics are based on meshing 23 models at several different sizes, such that the smoother was called a total 124 times. The set is comprised of triangle, quadrilateral, and mixed meshes.

Table 1. Improvements to distortion metric values from smoothing

	Average value before smoothing	Average value after smoothing	Average increase
Average Metric	0.82	0.86	0.04
Minimum Metric	0.58	0.70	0.12
When minimum metric was	< 0.1	~0.55	0.48

In Table 1, the rows are listed in increasing order of importance. The minimum metric is the most important number to improve – especially when it drops below 0.1 (our arbitrary minimum quality allowed). One of the tests started with a minimum metric as low as -0.49, but ended up with a minimum metric of 0.12.

Of the 124 times that the smoother was called, the smoother was called the minimum metric got slightly worse 8 times (by a maximum of 0.06). There is no cause for alarm in these cases, however, because the worst minimum metric that any of these was only 0.42 (nowhere near the poor quality cutoff of 0.1) and the maximum that any got worse was 0.06. The reason for these minor variations in minimum metric are due to the way in which the acceptance criteria for the Laplacian smooths are set up.

7.2 Speed

The average amount of time spent in the smoother is less 30% of the total meshing time. Over 160 models were selected and meshed at several mesh sizes, resulting in over 850 calls to the smoother, so as to extract the following timing statistics.

Table 2. Timing statistics

1. Percent of total meshing time spent in the smoother	29
2. Percent of smoothing time spent computing distortion metrics	38
3. Percent of smoothing time performing Laplacian constraint	55
4. Percent of nodes requiring optimization-based smoothing	< 1.0

Note that in Table 2, the time spent on performing the Laplacian constraint is largely a superset of the time spent computing the distortion metrics. Therefore, the actual Laplacian constraint checks take less than 20% of the total smoothing time.

Since the vast majority of nodes do not require optimization-based smoothing, nearly all of the expense incurred for optimization-based smoothing is in checking to see if it is required. For most models, less than 1% of the nodes end up requiring optimization-based smoothing. Moreover, since the same metric is used for both types of smoothing, it does not have to be recomputed for the optimization-based smoothing checks.

7.3 Examples

Figures 3a and 3b below show a triangular mesh on a doubly curved surface both before and after smoothing. Note how the curvature is nicely maintained.

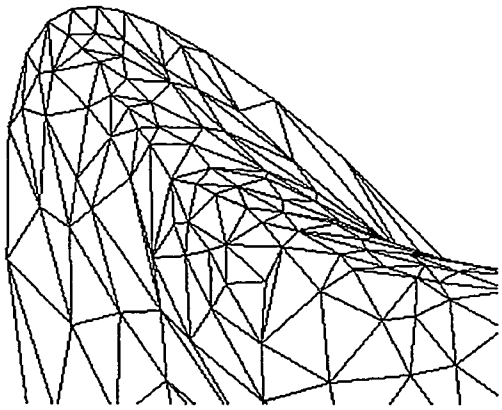


Figure 3a. Tri mesh before smoothing

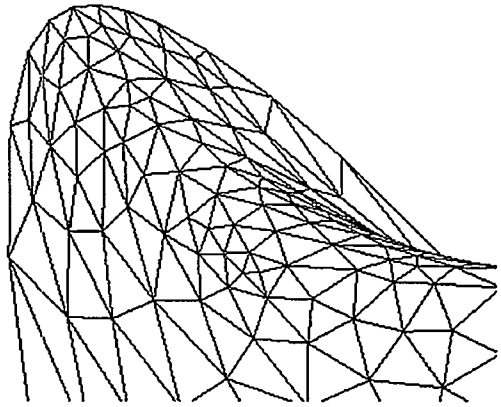


Figure 3b. Tri mesh after smoothing

Figures 4a and 4b below show another triangular mesh with and without smoothing. This tri mesh will be used as to create the quad mesh in Figure 5.

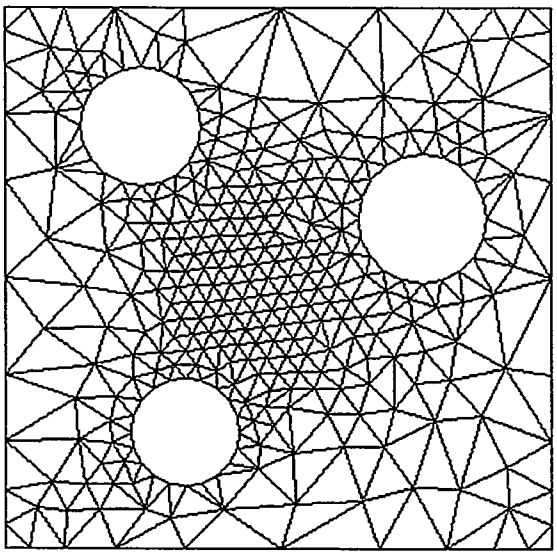


Figure 4a. Tri mesh before smoothing

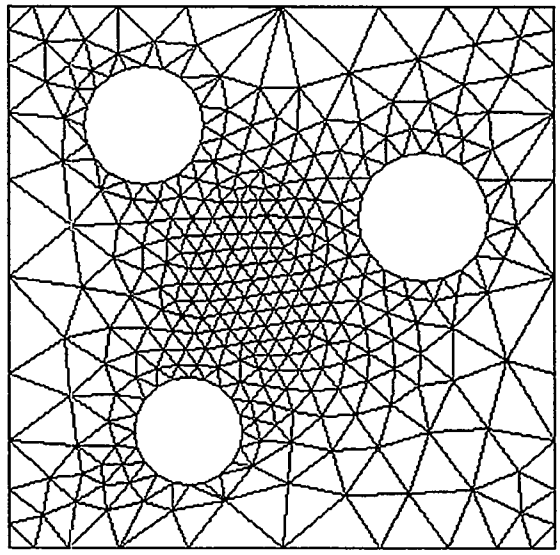


Figure 4b. Tri mesh after smoothing

Figures 5a and 5b show a quad mesh before and after smoothing. Note how severely distorted some of the quads are that must be handled by the smoother.

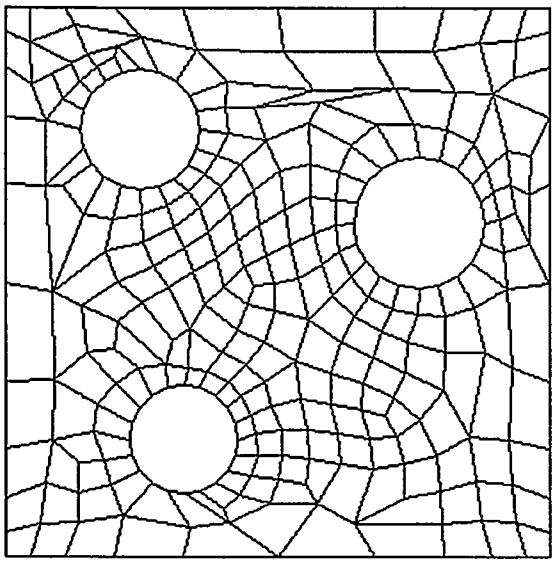


Figure 5a. Quad mesh before smoothing

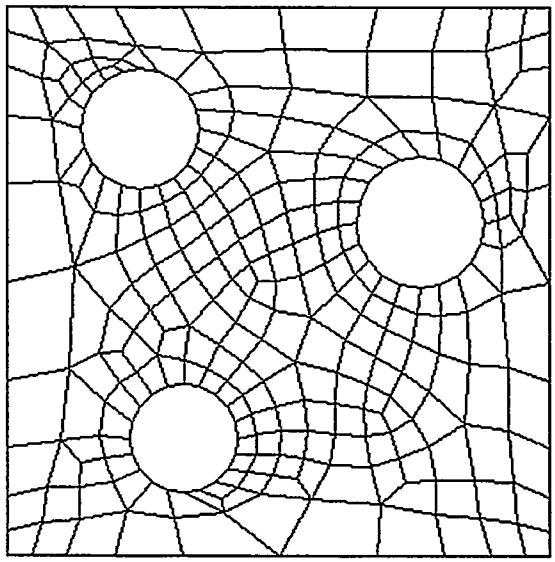


Figure 5b. Quad mesh after smoothing

Figures 6a and 6b show a simple quad mesh before and after smoothing.

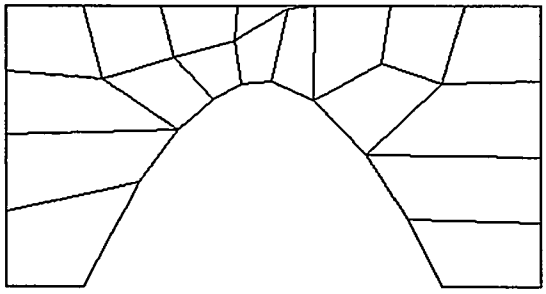


Figure 6a. Quad mesh before smoothing

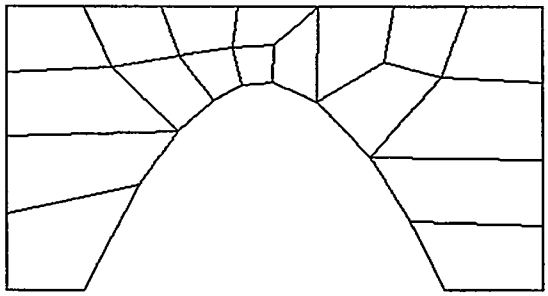


Figure 6b. Quad mesh after smoothing

8. Conclusion

An efficient and robust mesh smoothing scheme for 2D and 3D surface meshes is presented which works for tri, quad, and mixed meshes. This technique includes the combination of both constrained Laplacian and optimization-based smoothing. A distortion metric is presented that is well suited to both types of smoothing. The technique has been tested extensively and is in production use in ANSYS. The technique can (and has) been extended to 3D elements.

Acknowledgements

The authors wish to express gratitude to Steve Owen and Barry Joe for their help in the development and implementation of several of the ideas presented in this paper. We would also like to thank ANSYS, Inc. for its support of the research, development and publication of this work.

References

[1] Amenta, N., M. Bern, and D. Eppstein, "Optimal point placement for mesh smoothing," In *8th ACM-SIAM Symposium on Discrete Algorithms*," New Orleans, p. 528-537, 1997.

[2] Amezua, E., M.V. Hormaza, A. Hernandez, and M. B. G. Ajuria, "A method of the improvement of 3D solid finite element meshes," *Advances in Engineering Software*, Vol. 22, p. 45-53, 1995.

[3] Babuska, I., and A.K. Aziz, "On the angle condition in the finite element method, *SIAM Journal on Numerical Analysis*, Vol. 13, p. 214-227, 1976.

[4] Bank, R.E., A.H. Sherman, and A. Weiser, "Refinement algorithms and data structures for regular local mesh refinement," In R. Stepleman et al., editor, *Scientific Computing*, p. 3-17, IMACS/North-Holland Publishing Company, Amsterdam, 1983.

[5] Bank, R.E., and R.K. Smith, "Mesh smoothing using a posteriori error estimates," *SIAM Journal on Numerical Analysis*, Vol. 34, No. 3, p. 979-997, 1997.

[6] Barlow, J., "More on optimal stress points-reduced integration, element distortions and error estimation," *International Journal for Numerical Methods in Engineering*, Vol. 28, No. 7, p. 1487-1504, 1989.

[7] Batdorf, M., Freitag, L.A., and C. Ollivier-Gooch, "Computational study of the effect of unstructured mesh quality on solution efficiency," *Presented at the 13th Annual Computational Fluid Dynamics Meeting*, Snowmass Village, CO, 1997.

[8] Berzins, M., "Solution-based mesh quality for triangular and tetrahedral meshes," *Proceedings of the Sixth International Meshing Roundtable*, Sandia National Laboratories, p. 427-436, 1997.

[9] Blacker, T.D., M.B. Stephenson, and S.A. Canann, "Analysis automation with paving: a new quadrilateral meshing technique, *Proceedings of Design Productivity Institute and International Conference*, University of Missouri-Rolla, Honolulu, Feb. 1991.

[10] Blacker, T.D., and M.B. Stephenson, "Paving: A new approach to automated quadrilateral mesh generation," *Sandia Report No. SAND90-0249*, Sandia National Laboratories, Albuquerque, N. M., 1990.

[11] Blacker, T.D., and M.B. Stephenson, "Paving: A new approach to automated quadrilateral mesh generation," *International Journal for Numerical Methods in Engineering*, Vol. 32, p. 811-847, 1991.

[12] Bossen, F.J., and P.S. Heckbert, "A pliant method for anisotropic mesh generation," *Proceedings of 5th International Meshing Roundtable*, Sandia National Laboratories, p.63-74, October 1996.

[13] Briere de L'Isle, E. and P.L. George, "Optimization of tetrahedral meshes," *IMA Volumes in Mathematics and its Applications*, I. Babuska, W. D. Henshaw, J. E. Oliger, J. E. Flaherty, J. E. Hopcroft, and T. Tezduyar (Eds.), Vol. 75, p. 97-128, 1995.

[14] Canann, S.A., M.B. Stephenson, and T.D. Blacker, "Optismoothing: An optimization-driven approach to mesh smoothing," *Finite Elements in Analysis and Design*, Vol. 13, p. 185-190, 1993.

[15] Canann, S.A., S.N. Muthukrishnan, and R. Phillips, "Topological improvement procedures for triangular and quadrilateral finite element meshes," *In Proceedings of 3rd International Meshing Roundtable*, Sandia National Laboratories, October 1994.

[16] Canann, S.A., S.N. Muthukrishnan, and R. Phillips, "Topological refinement procedures for quadrilateral finite element meshes," to appear in *Engineering with Computers*, submitted and accepted, 1994.

[17] Canann, S.A., S.N. Muthukrishnan, and R. Phillips, "Topological refinement procedures for triangular finite element meshes," *Engineering with Computers*, Vol. 12, Nos. 3-4 combined, p. 243-255, 1996.

[18] Canann, S.A., Y.C. Liu, A.V. Mobley, "Automated 3D surface meshing to address today's industrial needs," *Finite Elements in Analysis and Design*, Vol. 25, Nos. 1-2 combined, p. 185-198, Mar 1997.

[19] Cass, R.J., S.E. Benzley, R.J. Meyers, and T.D. Blacker, "Generalized 3-D paving: an automated quadrilateral mesh generation algorithm," *International Journal for Numerical Methods in Engineering*, Vol. 39, p. 1475-1489, 1996.

[20] Cuilliere, J.C., "An adaptive method for the automatic triangulation of 3D parametric surfaces," *Computer-Aided Design*, Vol. 30, p. 139-149, 1998.

[21] de Cougny, H.L., M.S. Shephard, and M.K. Georges, "Explicit node point smoothing within the octree mesh generator," *SCOREC Report #10-1990*, 1990.

[22] Edelsbrunner H., and N. Shah, "Incremental topological flipping works for regular triangulations. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, p. 43-52, 1992.

[23] Field, D., "Laplacian smoothing and Delaunay triangulations," *Communications in Numerical Methods in Engineering*, Vol. 4, p. 709-712, 1988.

[24] Field, D., "Give me a good mesh", *Proceedings of 5th International Meshing Roundtable*, Sandia National Laboratories, p.3, October 1996.
[25] Freitag, L., M. Jones, and P. Plassmann, "An efficient parallel algorithm for mesh smoothing," In *Proceedings of the Fourth International Meshing Roundtable*, Sandia National Laboratories, p. 47-58, 1995.
[26] Freitag, L., and C. Ollivier-Gooch, "A comparison of tetrahedral mesh improvement techniques." In *Proceedings of the Fifth International Meshing Roundtable*, Sandia National Laboratories, p. 87-100, 1996.
[27] Freitag, L., "On combining Laplacian and optimization-based mesh smoothing techniques." *AMD Trends in Unstructured Mesh Generation*, ASME, Vol. 220, p.37-43, July 1997.
[28] Freitag, L., and C. Ollivier-Gooch, "The effect of mesh quality on solution efficiency," In *Proceedings of the Sixth International Meshing Roundtable*, Sandia National Laboratories, p. 249, 1997.
[29] Freitag, L., "Tetrahedral mesh improvement using swapping and smoothing," to appear in *International Journal for Numerical Methods in Engineering*, submitted, 1997.
[30] George, P.L., Automatic Mesh Generation, Paris, Masson, p. 234-236, 1991.
[31] George, P.L., and H. Borouchaki, Triangulation de Delaunay et Maillage Applications aux Elements Finis, Paris, Editions Hermes, 1997.
[32] George, P.L., and H. Borouchaki, Delaunay Triangulation and Meshing, Application to Finite Elements, Paris, France, Editions Hermes, p. 230-234, 1998.
[33] Gutierrez, M.A., and R. de Borst, "An algorithm for quadrilateral mesh rezoning," *Presented at Finite Elements in Fluids; New Trends and Applications*, Vol. 2, Eds. M. Morandi Cecchi et al, Univ. of Padova, Padova, 1995.
[34] Hansbo, P., "Generalized Laplacian smoothing of unstructured grids," *Communications in Numerical Methods in Engineering*, Vol. 11, p. 455-464, 1995.
[35] Herrmann, L.R., "Laplacian-isoparametric grid generation scheme," *Journal of Engineering Mechanics*, EM5, p. 749-756, Oct. 1976.
[36] Jacquotte, O.P., and G. Coussement, "Structured mesh adaption: space accuracy and interpolation methods," *Computer Methods in Applied Mechanics and Engineering*, Vol. 101, p. 397-432, 1992.
[37] Joe, B., "Three-dimensional triangulations from local transformations," *SIAM Journal on Scientific and Statistical Computing*, Vol. 10, p. 718-741, 1989.
[38] Joe B., "Construction of three-dimensional improved quality triangulations using local transformations," *SIAM Journal on Scientific Computing*, Vol. 16, p. 1292-1307, 1995.
[39] Jones, N.L., and S.G. Wright, "Algorithm for smoothing triangulated surfaces," *J. Comput. Civil Eng.*, ASCE 1, p. 85-102, 1991.
[40] Jones, R.E., "QMESH: A self-organizing mesh generation program," *Sandia Report SLA-73-1088*, Sandia National Laboratories, p. 23, 1974.
[41] Khamayseh, A., and A. Kuprat, "Anisotropic smoothing and solution adaption for unstructured grids," *International Journal for Numerical Methods in Engineering*, Vol. 39, p. 3163-3174, 1996.
[42] Kohli, H.S., and G.F. Carey, "Shape optimization using adaptive shape refinement," *International Journal for Numerical Methods in Engineering*, Vol. 36, p. 2435-2451, 1993.
[43] Lautersztajn-S, N., and A. Samuelsson, "Distortion measures and inverse mapping for isoparametric 8-node plane finite elements with curved boundaries," *Communications in Numerical Methods in Engineering*, Vol. 14, p. 87-101, 1998.
[44] Lee, C.K, and S.H. Lo, "A new scheme for the generation of a graded quadrilateral mesh," *Computers and Structures*, Vol. 52, No. 5, p. 847-857, 1994.
[45] Lewis, R.W., Y. Zheng, and D.T. Gethin, "Three-dimensional unstructured mesh generation: Part 3. Volume meshes," *Computer Methods in Applied Mechanics and Engineering*, Vol. 134, p. 285-310, 1996.
[46] Liu, A. and B. Joe, "On the shape of tetrahedra from bisection," *Mathematics of Computations*, Vol. 63, p. 141-154, 1994.
[47] Liu, A. and B. Joe, "Relationships between tetrahedron shape measures," *BIT*, Vol. 34, p. 268-287, 1994.
[48] Lo, S.H., "A new mesh generation scheme for arbitrary planar domains," *International Journal for Numerical Methods in Engineering*, Vol. 21, p. 1403-1426, 1985.
[49] Lo, S.H., "Generating quadrilateral elements on plane and over curved surfaces," *Computers and Structures*, Vol. 31, No. 3, p. 421-426, 1989.
[50] Lohner, R., K. Morgan, and O.C. Zienkiewicz, "Adaptive grid refinement for the compressible Euler equations," in I. Babuska, O.C. Zienkiewicz, J. Gago, and E.R. de A. Oliviera (Eds.), Accuracy Estimates and Adaptive Refinements in Finite Element Computations, Wiley, p. 281-297, 1986.

[51] Oddy, A., J. Goldak, M. McDill, M. Bibby, "A distortion metric for isoparametric finite elements," *Trans. CSME*, No. 38-CSME-32, Accession No. 2161, 1988.
[52] Owen, S.J., M.L. Staten, S.A. Canann, and S. Saigal, "Q-MORPH: An indirect approach to advancing front quad meshing," submitted to *International Journal for Numerical Methods in Engineering*, 1998.
[53] Owen, S.J., and S. Saigal, "Neighborhood-based element sizing control for finite element surface meshing," *Proceedings of 6th International Meshing Roundtable*, Sandia National Laboratories, p. 143-154, Oct. 1997.
[54] Parthasarathy, V., and S. Kodiyalam, "A constrained optimization approach to finite element mesh smoothing," *Finite Elements in Analysis and Design*, Vol. 9, p. 309-320, 1991.
[55] Riccius, J., K. Schweizerhof, and M. Baumann, "Combination of adaptivity and mesh smoothing for the finite element analysis of shells with intersections," *International Journal for Numerical Methods in Engineering*, Vol. 40, p. 2459-2474, 1997.
[56] Rivara, M.C., "New mathematical tools and techniques for the refinement and/or improvement of unstructured triangulations," *Proceedings of 5th International Meshing Roundtable*, Sandia National Laboratories, p. 77-86, October 1996.
[57] Rivara, M.C., "Non-obtuse boundary Delaunay triangulations," In *Proceedings of 6th International Meshing Roundtable*, Sandia National Laboratories, p. 391, October 1997.
[58] Robinson, J., and G. Haggenmacher, "Element Warning Diagnostics," *Finite Element News*, Jun/Aug, 1982.
[59] Robinson, J., "Some new distortion measures for quadrilaterals," *Finite Element Analysis*, Vol. 3, p.183-197, 1987.
[60] Robinson, J., "Distortion measures for quadrilaterals with curved boundaries," *Finite Element Analysis*, Vol. 4, p. 115-131, 1988.
[61] Roddeman, D.G., and Jansen, L.F., "An a priori geometry check for a single isoparametric finite element," *Computers and Structures*, 47, p. 69-72, 1993.
[62] Salem, A. Z. I., S.A. Canann, and S. Saigal, "Robust distortion metric for quadratic triangular 2D finite elements," *ASME Applied Mechanics Division*-Vol. 220, p.73-80, 1997.
[63] Salem, A.Z.I., S.A. Canann, and S. Saigal, "Mid-node admissible space for quadratic triangular 2D finite elements with one curved edge," *submitted to International Journal for Numerical Methods in Engineering*, 1998.
[64] Salem, A.Z.I., S.A. Canann, and S. Saigal, "Mid-node admissible space for quadratic triangular 2D finite elements," *submitted to International Journal for Numerical Methods in Engineering*, 1998.
[65] Salem, A.Z.I., S.A. Canann, and S. Saigal, "Mid-node admissible space for 3D quadratic tetrahedral finite elements," *submitted to International Journal for Numerical Methods in Engineering*, 1998.
[66] Shephard, M.S., and M.K. Georges, " Automatic three-dimensional mesh generation by the finite octree technique," *International Journal for Numerical Methods in Engineering*, Vol. 32, No. 4, p. 709-749, 1991.
[67] Shimada, K., "Physically-based mesh generation: Automated triangulation of surfaces and volumes via bubble packing, Ph.D. thesis, ME Dept., Massachusetts Institute of Technology, Cambridge, MA, 1993.
[68] Shimada, K., and D.C. Gossard, "Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing." *In ACM Third Symposium on Solid Modeling and Applications*, p. 409-419, May 1995.
[69] Shimada, K., "Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles," *Proceedings of the Sixth International Meshing Roundtable*, Sandia National Laboratories, p. 375-390, 1997.
[70] Staten, M.L., and S.A. Canann, "Post refinement element shape improvement for quadrilateral meshes," AMD-Vol. 220 Trends in Unstructured Mesh Generation, ASME, p. 9-16, July 1997.
[71] Tristano, J.R., S.J. Owen, and S.A. Canann, "Advancing front surface mesh generation in parametric space using a Reimannian surface definition," submitted to the 7th International Meshing Roundtable, Dearborn, MI, Oct 1998.
[72] Winslow, A.M., "Equipotential zoning of two-dimensional meshes, Lawrence Livermore Laboratory, University of California, Livermore, California, *Report No. UCRL-7312*, 1963.
[73] Zhu, J.Z., O.C. Zienkiewicz, E. Hinton, and J. Wu, "A new approach to the development of automatic quadrilateral mesh generation," *International Journal for Numerical Methods in Engineering*, Vol. 32, p. 849-866, 1991.
[74] Zienkiewicz, O.C., The Finite Element Method, London, McGraw Hill, 1971.
[75] Meshing your solid model, *ANSYS Modeling and Meshing Guide*, Release 5.4, 000862, 2nd Edition, SAS IP, Inc., Chapter 7, September 1997.
[76] Revising your solid model, *ANSYS Modeling and Meshing Guide*, Release 5.4, 000862, 2nd Edition, SAS IP, Inc., Chapter 8, September 1997.

Novel Meshing Approaches

Genetic Algorithms, Another Tool for Quad Mesh Optimization?

Mike Holder, Ph.D. Student
Civil Engineering Dept., University of Alabama
Box 870205, Tuscaloosa AL 35407
mikehii@traveller.com

Jim Richardson, Assoc. Professor
Civil Engineering Dept., University of Alabama
jrichardson@coe.eng.ua.edu

Abstract

The purpose of this paper is to investigate the use of a genetic algorithm (GA) to perform the finite element analysis mesh smoothing process. It is the goal of this paper to take a simple quadrilateral mesh and smooth it using a GA into a useful model that will provide a correct solution. The GA smoothing technique is demonstrated on two simple quadrilateral mesh examples. Only one node is moved at a time in the examples.

A distortion metric is used to quantify the "goodness" of a quadrilateral element and serves as the fitness function for the GA. Other implementation details such as convergence criteria, population size, cross-over probability and mutation rate are discussed in the paper. Early results from the two simple check problems are presented. Finally, planned future work is outlined and possible GA smoothing applications are discussed.

Keywords: Genetic Algorithm, Artificial Intelligence, FEA Mesh Smoothing, Mesh Optimization

1. Introduction

The goal of most finite element analyses (FEA) is to verify the suitability of an engineering design. The challenge is to build a sufficiently accurate model in the available time. One of the most time-consuming tasks in building a finite element model is generating and optimizing the finite element mesh.

Many algorithms have been developed to create the elements of the model, a process called meshing, but the meshes are often constructed of poorly shaped and thus error prone elements. Smoothing, a technique of correcting the poorly shaped elements, is an emerging science with several useful methods available. No one smoothing technique works all of the time.

It is the purpose of this paper to explore the use of a simple genetic algorithm (GA), an artificial intelligence (AI) method, to control the execution of the smoothing process. GAs will take longer to run than most conventional methods such as Laplacian smoothing but may prove useful for smoothing difficult meshes (non-convex meshes for example). This paper describes application of a GA to smooth a poorly formed simple convex quadrilateral mesh. Future work includes application of a GA to solve a non-convex mesh and other more complex meshes. The final goal of this project is to apply GAs to smooth non-convex 3D surfaces. It is hoped that GA smoothing may ultimately be able to smooth hexahedral meshes.

Quadrilateral Elements Quadrilateral elements are more accurate, in general, than triangular elements for 2D stress analysis. For the same reasons, hexahedral elements are more accurate than tetrahedral elements for 3D stress analysis, especially when working with materials with a high Poisson's ratio such as rubber (Key, 1997). Quad and hex meshes are harder to generate than comparable triangle and tet meshes.

Smoothing Smoothing or mesh adjusting is an operation where nodes are repositioned to produce elements more closely resembling the theoretical element (for example, a square for quadrilateral elements). In addition to repositioning nodes, nodes can be added and deleted and elements can be added and deleted. The most common smoothing algorithm is Laplacian smoothing (Hansbo, 1995; Field, 1988; Frey, 1991; Bossen, 1996). Other smoothing methods include the spring method (Hansbo, 1995), mesh relaxation (Field, 1995), pliant method (Bossen, 1996) and the genetic algorithm used in this paper (Holland, 1975; Goldberg, 1989, Smith, 1994).

Laplacian Smoothing In the Laplacian smoothing technique, nodes are moved in the areas of poorly conditioned elements. The Laplacian algorithm locates an offending node and moves it to the centroid of the surrounding nodes thus helping to improve the shape of the elements in most cases (Bossen, 1996). On some non-convex domains, nodes are pulled outside the boundary.

A more sophisticated variation is the Laplace-Delaunay smoothing (Bossen, 1996), which performs edge swaps and Laplacian smoothing in a loop, adjusting the topology and geometry of the mesh at the same time. This method does not have the ability to add or subtract nodes.

2. Artificial Intelligence (AI)

Artificial intelligence is the study of the computations that make it possible to perceive, reason and act (Winston, 1992). This paper briefly describes three forms of AI previously used to smooth meshes. Smoothing meshes with artificial intelligence is not currently the best way to smooth meshes. It is a different approach to mesh smoothing that has not been presented at the previous meshing conferences. Given enough time on a fast machine (measured in terms of hours) one of these AI methods may prove capable of solving the arbitrary 3-D hex meshing problem. Perhaps some clever person can figure out a way to reduce the solution to minutes instead of hours.

Expert Systems An expert system is a knowledge-based system programmed to make decisions an expert would make. An expert system typically includes a sizable database, consisting of facts about the domain and rules for applying those facts (Mishkoff, 1985). It is a large list of do's and don'ts containing as many rules as required to solve the problem in question. For example, the expert system may be programmed to partition a region into subregions. There would be rules about thin sections, fillets, holes, boundary conditions, etc., to partition the region just as the engineer would do it.

Genetic Algorithms A Genetic algorithm (GA) is a procedure for improving results of some operation by repeatedly trying several possible solutions and reproducing and mixing the components of the better solutions. The process is inspired by, and crudely similar to, the process of evolution in the biological world, where the fittest survive to reproduce (Evolver, 1995). According to Goldberg (Goldberg, 1989), genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. According to Pal (Pal, 1996), GAs are adaptive and robust computational procedures modeled on the mechanics of natural genetic systems. GAs have been successfully used to optimize or smooth 2D FEA meshes populated with triangles (Guesta, 1994). Unfortunately, we were unable to communicate directly with Guesta or his staff to discuss the details of his paper.

GAs act as a biological metaphor and try to emulate some of the processes observed in natural evolution. They are viewed as randomized, yet structured, search and optimization techniques. They express their ability by efficiently exploiting the historical information (remembering good results) to speculate on new offspring (producing new generations of solutions) with higher fitness (expected improved performance). GAs are executed iteratively on a set of coded solutions, called a population, with three basic operators: selection or reproduction, crossover, and mutation.

In order to find out the optimum solution of a problem, a GA starts from a set of chromosomes (assumed solutions) and evolves different but better sets of chromosomes (sets of solutions) over a sequence of iterations. In each generation (iteration) the fitness measuring criterion (objective function) determines the suitability of each chromosome and, based on these values, some of them are selected for reproduction. The number of copies

reproduced by an individual parent is expected to be directly proportional to its fitness value, thereby embodying the natural selection procedure, to some extent. The procedure thus selects more fit solutions; less fit solutions are eliminated.

GAs were developed by John Holland at the University of Michigan in the mid 70s and are the subject of much research today. GAs are robust and, although they may not find a perfect solution, they come close enough for most engineering work for a wide verity of problems. Multimodal and highly discontinuous problems are taken in stride by GAs. There are numerous variations on GAs including different types of crossover algorithms, hybrids combining GAs with fuzzy logic, simulated annealing and neural networks.

Neural Networks. GAs and neural networks (NN) are both considered artificial intelligence technologies, and are often lumped together because both were inspired by biological systems in nature; the GA mimics Darwinism evolution, and the NN mimics the brain.

In a GA problem, we usually want to find the inputs to pump through a given model that will produce the optimal output. In a NN problem, we usually have many sets of inputs and their related outputs, but we are looking for the model that ties them together. The NN is trying to make sense of your inputs and outputs by building an internal black box model of its own. This makes NN good for pattern recognition and prediction applications, where you have just raw data, and you are looking for a model to make sense of the data.

NNs must first be trained and tested before they are used. Once built, we take a subset of the inputs and related outputs and feed it through the NN. This training data allows the NN to try and create a model that accurately relates those inputs to their output. A NN is only as good as its training data.

GAs and NNs can be joined together. A GA can help turn a NN inside out, crunching through thousands of possibilities to find the best set of inputs to feed the network to achieve a desired output result. NNs can be trained to do optimization problems like the traveling salesman's problem, design optimization, cost optimization, job-shop scheduling and FEA mesh generation (Dyck, 1992).

Of the three AI methods described above and previously used by others to smooth meshes, genetic algorithms was considered by the authors to be the most promising. The following sections describe initial implementation efforts of a GA for smoothing quadrilateral meshes.

3. Implementation of GA for Mesh Optimization

Each node of the 2D mesh is represented by an x and a y coordinate. In the simple GA used for this project, the chromosome is a binary representation of one x-y nodal location in the 2D region. The value of each chromosome is compared with a fitness function. The more fit solutions reproduce and the less fit solutions die off.

In our chromosome representation, 30 binary characters or bits represent the nodal location, fifteen binary locations for the x coordinate and fifteen for the y coordinate. Fifteen binary locations make 32767 equal steps between the maximum and minimum coordinates determined for each search region.

Other decisions in the GA are the size of the initial population, the type of selection, crossover possibility, and the beginning and ending mutation probabilities. Figure 1 is a flow chart of a simple GA and the solution method taken in this approach to the smoothing process.

A random number generator creates the initial population. Each number represents a x-y nodal location inside the search region. The initial population is a balancing act, too many may result in a loss of efficiency and too few may result in a loss of accuracy.

Pairs of individuals are selected using a tournament selection. The population is broken down into random tournaments of size k ($k = 2$ is most commonly used). A copy of the best individual in each tournament is assigned to the mating pool. The process is repeated k times. Using this method, the best individual always receives k copies in the mating pool. One other method is roulette wheel selection, but it produces less than desirable results.

Crossover or recombination comes in a variety of choices. Single, uniform and multi-point crossovers are a few of the types. In this paper single-point crossover is used. Here crossover is used on two mates, producing two children. An integer k is selected at random between one and the string length minus one (1 and 14 in our case). Two new strings are created by swapping all characters (bits) between positions k + 1 and 14.

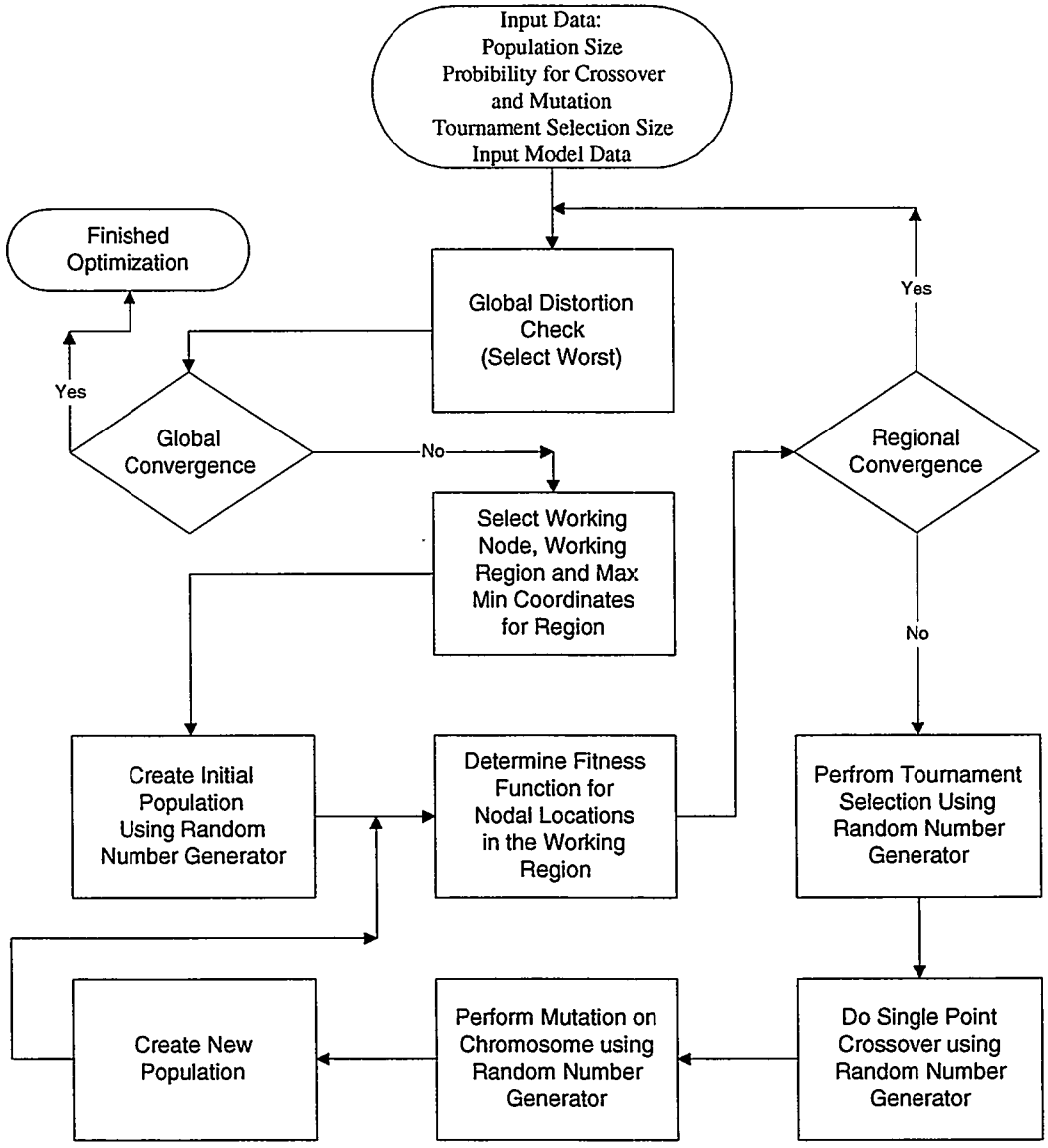


Figure 1. Flow chart of solution procedure.

Mutation adds diversity to the GA. It is a random walk in the search space. Two mutation rates are used in this GA written by the University of Alabama (Smith, 1994). The first is a higher rate with a probability of 0.533 and ending at 0.0333. Higher mutation in the beginning of the GA and less near the end for less disruption as the GA converges.

The core of this quad optimization problem is the fitness function. Here a distortion metric developed by Carleton University in Ottawa, Ontario, Canada was selected (Oddy, 1988). What was desired for this problem, was a single number that represents the overall goodness of a quad element. This metric is a function of the element Jacobian

(averaged for the four Gauss points for each element). Green's strain is examined, deviatoric strains are considered with a resulting function in terms of the Jacobian to the fourth power. The distortion metric grows rapidly as the distortion increases. We are experimenting with GAs in this paper, not goodness criteria, with details of Oddy's metric given in his paper (Oddy, 1988).

The distortion metric for each element in the overall model is computed and the worst element is selected. The program next examines the distortion of each of the surrounding elements with the algorithm selecting one of the four nodes of the worst element as a working node. The elements common to this node constitute the working region. The max/min coordinates are determined for the region and the GA then proceeds to "move" the working node to minimize the distortion of the elements in the working region. Once an optimum is achieved for the region (either a minimum distortion metric is reached or an error value is achieved) the distortion metric is then recalculated for the entire model and the process repeated until a global minimum is achieved.

Unlike Laplacian smoothing, the working region can be non-convex since the GA is looking at the individual element goodness and is not necessarily looking for a centroid of a region to position a node. A non-convex region using Laplacian smoothing could possibly place the node outside of the region.

4. Implementation of a GA for Mesh Smoothing

Check Problems. The first check problem used in this project was a simple four quad rectangle with one movable node in the interior of the region (Figure 2). The solution is trivial, but it provided an important opportunity to check out how the GA behaved. Depending on the error cutoff value (0.00 error being four perfect rectangles and 0.01 for an acceptable approximation), the GA typically stumbled into a solution quickly. The lower the error cutoff, the longer it took to run.

An exact solution would be x = 5.0 and y = 5.0 for node 5 since the overall size of the model is 10 X 10. The GA quickly found a solution of x = 4.6001 and y = 5.0255 with the worst distortion metric going from 0.4754 to 0.009065 with an error cutoff of 0.01. This occurred in the 1st generation with a population of 100. At an error cutoff at 0.1, the GA found a solution in the initial population.

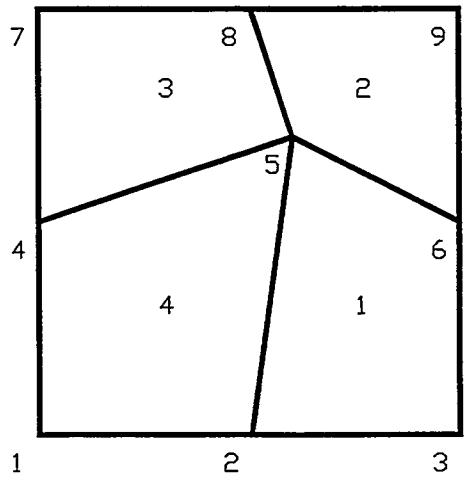


Figure 2. Example initial mesh #1.

Figure 3 is a 32 element convex geometry example. This example checked out the basic search algorithm and the GA. Here again the solution is obvious which makes a good test example.

Convergence Tolerance. The biggest problem in solving the 32 element problem is dealing with the error function or convergence criteria. Since only one node is moving at a time and adjacent elements are poorly shaped, a fixed error function set at a low number, i.e., 0.01, won't converge since the location of other nodes prevents a well shaped element.

The error function needs to be set sufficiently high, 0.5 for example (the worst error metric in the above model is 0.98892) and reduced as the GA optimizes the particular working region. Once this region is improved as much as possible, the GA moves on to the next worst region. This process of moving to the worst area of the model continues until a global model minimum has been reached. In this model, a global minimum of 0.01 was set and finally reached. Here a global minimum considers the distortion metrics for all elements in the model. More work needs to be done in fine tuning the GA's parameters and the convergence criteria since all of the input parameters influence the convergence of the GA.

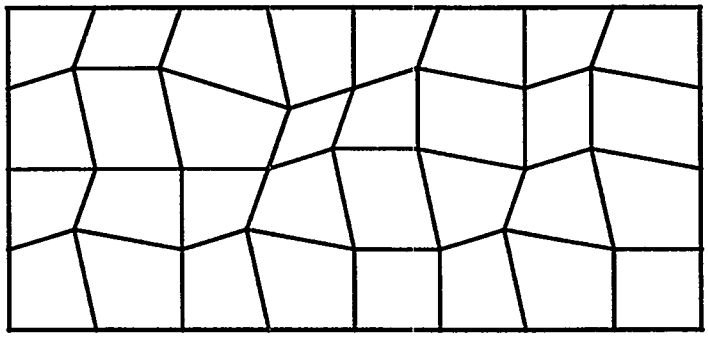


Figure 3. Example initial mesh #2.

Population Size. Population size is an example of a parameter affecting the algorithm's convergence. The larger, more diverse population takes longer to converge on a solution, but is more likely to find a solution because of its diverse gene pool. For these check problems, a population of 100 (100 trial points in the region of the working node) works well. Smaller populations typically didn't converge as quickly as larger populations. In some instances the smaller populations did not converge at all. Here time is measured in more than an hour using a 486-50 PC.

Crossover Probability. Crossover probability usually ranges from 0.01 to 1.0. Crossover reflects the likelihood that future population of nodes will contain a mix of information from the previous generation of nodes. A rate of 0.5 means that a child node will contain about 50% of its location from one parent node and the rest from the second parent node. A rate of 1.0 means that no crossover will occur, or only clones of the parents will be evaluated. We are using a crossover probability of 0.75 at this time.

Mutation Rate. The mutation rate can vary from 0.0 to 1.0. The GA software engine we're using allows a variable mutation rate. The suggested initial rate (Smith, 1994) is a function of the overall chromosome length.

$$((L/2) + 1)/L = (30/2 + 1)/30 = 0.533 \quad \text{for a chromosome length of } L = 30.$$

The suggested end mutation rate is

$$1/L = 1/30 = 0.0333.$$

The higher the mutation rate, the more likely future nodal chromosomes will contain some random values. Since mutation occurs after crossover, a mutation rate that is too high will prevent crossover from having much if any effect.

Fitness Function. The fitness function in this paper is the distortion metric generated from the four corner Jacobian values for each quadrilateral element. The convergence criteria that checks the individual fitness functions is perhaps the biggest challenge in setting up a good GA. The first method used in this paper was to simply define a relatively small error value, i.e. 0.01, and subtract the fitness value the GA calculated from 0.0 (a perfect quad) and compare it with this error value of 0.01. If it is less that the preset value, we have a solution. This works great for a mesh that will ultimately converge to a square, but what about geometry that doesn't or can't because of poorly shaped adjacent elements?

In this last case, the GA will eventually converge to an usable nodal value, but not necessarily a preset value like 0.01. Here we need to check the convergence process. We can select the best answer after a predetermined number of generations or we can select the value that the GA seems to be converging to and move on to the next "working region."

The convergence process may be a good application for fuzzy logic. We are not so lucky as to have a simple "go" or "no go" situation. We have a "maybe" or "maybe not" which is a natural for fuzzy logic. The way the GA works, a gradual relaxation of the convergence is required.

We can also create penalty functions to force the GA toward convergence. If the GA creates a value with an acceptably low distortion metric, the value passes it on as is (and say, "Good GA!"). If the GA creates a error value that is not acceptably low, you could take the square or cube of the error times a penalty coefficient and added it to the error value (and say, "Bad GA!").

The penalty coefficient allows the operator to tweak the penalty function for optimum performance. A penalty coefficient might vary from 0.1 to 1,000,000 or more depending on the results. This penalty function takes an unacceptable nodal value and makes it very unacceptable forcing the GA in the direction of convergence.

5. Conclusions and Future Work

Laplace smoothing moves the "working node" to the centroid of a group of "working elements." In a non-convex geometry situation, the working node may end up outside of the region. This won't happen with the GA smoothing method provided the element selection algorithm is properly written. Future work for this project will demonstrate this advantage.

Other future work will focus on taking advantage of the inherent flexibility and robustness of GAs as problem solvers. One idea is to have the GA simultaneously adjust multiple nodes for multiple elements. The chromosomes can be expanded to include the coordinates for multiple nodes, and the objective function can be modified to measure the distortion over several elements. Also, an "outer-loop" GA could be used to adjust the mesh density. This would allow for a higher density mesh in regions with distorted elements due to irregular geometry.

The goal of this project is to use the GA to smooth 3D non-convex surfaces. The next logical step after 3D surfaces is for someone to tackle the 3D solid geometry problem of hexahedrals. The most interesting feature of the GA is that is almost guarantees a workable solution for the elements present. Perhaps elements can be added or subtracted in the future for an even better solution.

The GA adjusts and readjusts all movable nodal locations (as opposed to fixed nodes, i.e. corner nodes and load points) continually checking and optimizing the fitness functions of the elements present. There is no apparent reason why this method can't be expanded to the 3D hexahedral problem.

List of References

Bossen, F. J., Heckbert, P. S., 1996, "A Pliant Method for Anisotropic Mesh Generation," 5th International Meshing Roundtable, Pittsburgh, PA, pp. 63-74.

Dyck, D. N., et al., March 1992, "Determining an Approximate Finite Element Mesh Density Using Neural Network Techniques," IEEE Transactions on Magnetics, Vol. 28, No. 2.

Evolver, 1995, The Genetic Algorithm Problem Solver, Version 3.1, Axcelis, Seattle, Washington.

Field, D. A., 1995, "The Legacy of Automatic Mesh Generation From Solid Modeling," Compute Aided Geometric Design, Vol. 12 pp. 651-673.

Field, D., 1988, "Laplacian Smoothing and Delaunay Triangulation," Communications in Applied Numerical Methods, Vol. 4, pp. 709-712.

Frey, W. H., Field, D. A., 1991, "Mesh Relaxation: A New Technique for Improving Triangulations," International Journal for Numerical Methods in Engineering, Vol. 31, pp. 1121-1133.

Goldberg, D. E., 1989, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA.

Guesta, P., et al., 1994, "Mesh Generation Using Genetic Algorithms," Advances in Structural Optimization, Civil-Comp Ltd., Edinburgh, Scotland, pp. 225-231.

Hansbo, Peter, 1995, "Generalized Laplacian Smoothing of Unstructured Grids,: Communications in Numerical Methods in Engineering, Vol. 11, pp. 455-464.

Holland, J. H., 1994 (1975), Adaptation in Natural and Artificial Systems, 3rd printing, MIT Press, Cambridge, MA.

Key, Samuel, Sandia National Laboratory, April 1997: Personal Correspondence.

Mishkoff, H. C., 1985, Understanding Artificial Intelligence, Howard Sams, Indianapolis.

Oddy, A., et al., 1988, "A Distortion Metric for Isoparametric Finite Elements," Department of Mechanical and Aeronautical Engineering, Carleton University, Ottawa, Canada.

Pal, S. K., and Wang, P. P., 1996, Genetic Algorithms for Pattern Recognition, CRC Press, Boca Raton, FL.

Smith, R. E., 1994, The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa, AL 35487.

Winston, P. H., 1992, Artificial Intelligence, 3rd ed. Assison Wesley, New York.

NEXT-GENERATION SWEEP TOOL: A METHOD FOR GENERATING ALL-HEX MESHES ON TWO-AND-ONE-HALF DIMENSIONAL GEOMTRIES

PATRICK M. KNUPP
PARALLEL COMPUTING SCIENCES DEPARTMENT
SANDIA NATIONAL LABORATORIES*
M/S 0441, P.O. BOX 5800
ALBUQUERQUE, NM 87185-0441
PKNUPP@SANDIA.GOV

Abstract. Placement of interior node points is a crucial step in the generation of quality meshes in sweeping algorithms. Two new algorithms were devised for node point placement and implemented in Sweep Tool, the first based on the use of linear transformations between bounding node loops and the second based on smoothing. Examples are given that demonstrate the effectiveness of these algorithms.

1. Introduction

It has been demonstrated that geometries that are two-and-one-half dimensional (e.g., generalized cylinders) can be meshed with all-hexahedral finite elements [1], [4]. Because all-hexahedral mesh generation on general three-dimensional geometries remains an elusive goal, algorithms to mesh two-and-one-half dimensional geometries, generally referred to as "sweeping" or "projecting" methods, continue to be important. Although mesh sweeping is simple in concept, the first-generation algorithms lacked robustness and often produced poor mesh quality.

There are several approaches to mesh generation via sweeping but common to all is the idea of identifying surfaces on a volume to serve as "sources" or "caps" and a complemetary set to serve as "linking sides." Source surfaces may be arbitrarily meshed with quadrilaterals via paving or multi-block structured meshing and then "swept" along the linking sides towards a "target" surface which may or may not be pre-meshed. This is feasible provided the linking side surfaces are meshed with a type of multi-block, structured mesh known as "submap" [5]. Identification of these source and linking surfaces has been automated in the CUBIT code [6].

Interior mesh connectivity is entirely determined once these surfaces have been identified and meshed. Placement of the interior nodes in space is then a crucial step in generating a quality mesh. For geometries that are sweepable via source mesh translation or rotation, the spatial location of the interior nodes is unambiguous. For more general geometries, however, there is no uniquely correct location for the nodes. The guiding principle in the general case is that nodes are placed so that hexahredral elements of a swept mesh are of good quality. As a minimal requirement, the "Jacobian" at the eight corners of all hexahedrons should be positive [2]. Another reasonable requirement is that qualities of the source surface mesh such as biasing or the relative areas of quadrilaterals should be transferred, in so far as possible, to the interior layer meshes and to the target. Even with these requirements, flexibility remains in the placement of the interior nodes in sweeping general geometries. This paper reports on a robust method for uniquely specifying the interior node positions in a way that directly addresses the mesh quality issue.

* SANDIA IS A MULTIPROGRAM LABORATORY OPERATED BY SANDIA CORPORATION, A LOCKHEED MARTIN COMPANY, FOR THE UNITED STATES DEPARTMENT OF ENERGY UNDER CONTRACT DE-ACO4-94AL85000.

The outline of this paper is as follows: section 2 describes how point placement was done in the first-generation Project Tool [4] in order to illustrate the challenge posed by general two-and-one-half dimensional geometries. Section 3 treats the point placement algorithm based on linear transformations that is used in Sweep Tool while Section 4 describes results of layer smoothing as a point placement method, both in Sweep and Project Tools. Of course, interior point placement is not the only issue that arises in designing a good sweeping algorithm, so Section 5 briefly covers some additional topics of importance. In section 6 we offer our conclusions.

2. Interior Point Placement in the Project Scheme

Interior points are located in the Project scheme by an advancing front mechanism that builds a layer of hexahedrons on the front from the previous layer, begining with the source surface meshes. An individual hexahedron on the front is built using the four nodes x_0, x_1, x_2, and x_3 on the previous layer plus three other nodes x_4, x_5, and x_6 belonging to the hexahedron under construction. The latter are known from the boundary data on the linking surface and from pre-existing nodes on the layer containing the advancing front. In the original algorithm, the final node x_7 of the hexahedron was computed by averaging the three vectors $x_4 - x_0$, $x_5 - x_1$, and $x_6 - x_2$ that approximately point in the direction in which the front advances:

$$x_7 = x_3 + \frac{1}{3}\{(x_4 - x_0) + (x_5 - x_1) + (x_6 - x_2)\}. \tag{1}$$

This approach gives the desired nodal position under a translation: if $x_{4+k} = x_k + c$ for some constant vector $c \in R^3$ and $k = 0, 1, 2$, it immediately follows that $x_7 = x_3 + c$. However, not all meshes on sweepable geometries can be translated so equation (1) is inadequate for many important cases. Foremost among these are rotatable geometries; rotations are not preserved under (1) and very poor meshes result. An improved "planar-intersection" algorithm was suggested in [4] in which the point x_7 was determined by the intersection of the three planes containing the points (x_4, x_5, x_6), (x_0, x_3, x_4), and (x_2, x_3, x_6), respectively. While this alternate algorithm improved the quality of the projected meshes, it still did not work effectively on rotatable geometries.

Two approaches to solve this difficulty were suggested in [4]. In the first approach, a separate program module (known as scheme Rotate) was developed for geometries whose source meshes can be rotated. The planar-intersection algorithm was discarded in favor of one that works only in the rotational case. The result was a highly efficient module that consistently achieved good quality meshes on a limited set of geometries. A minor disadvantage of adding Rotate to the suite of tools was that there were then two modules to maintain when only one is actually needed. More importantly, however, is that other commonly encountered geometries such as a frustum needed development of additional special modules because the heuristically-based planar-intersection algorithm failed to give quality meshes on them as well.

To overcome this piecemeal approach, the authors of Project tried mesh smoothing. In some respects this was premature because, as is demonstrated in the next section, the planar-intersection algorithm can be replaced with a unifying technique involving linear transformations that handles all geometries which require translation, rotation, and scaling of layer meshes.

3. Interior Point Placement via Linear Transformations

Given a layer mesh of points $p_j \in R^3$ with bounding loop L consisting of the points $\{x_k\}$, $k = 1, 2, \ldots, K$ with $K \geq 3$, and given a second loop $\tilde{L}$ consisting of K points $\{\tilde{x}_k\}$ in R^3, we seek a vector $b \in R^3$ and a 3×3 non-singular linear transformation $\mathcal{T}$ between loops such that

$$\tilde{x}_k = \mathcal{T} x_k + b \tag{2}$$

for all k. Then, for any point p_j belonging to the layer bounded by loop L, we determine the point $\tilde{p}_j$ in the layer bounded by loop $\tilde{L}$ by $\tilde{p}_j = \mathcal{T} p_j + b$.

The vector b can be readily determined from the loop data. Define loop center points

$$c = \frac{1}{K} \sum_k x_k , \tag{3}$$

$$\tilde{c} = \frac{1}{K} \sum_k \tilde{x}_k . \tag{4}$$

It is not necessary that these center points lie "inside" the loops. If we sum (2) over all k, we find:

$$b = \tilde{c} - \mathcal{T} c ; \tag{5}$$

thus (2) becomes

$$\tilde{x}_k - \tilde{c} = \mathcal{T} (x_k - c) . \tag{6}$$

For convennience, let $u_k = x_k - c$ and $\tilde{u}_k = \tilde{x}_k - \tilde{c}$ so that $\tilde{u}_k = \mathcal{T} u_k$.

Define K matrices $\mathcal{U}_k = [u_k, u_{k+1}, u_{k+2}]$ and $\tilde{\mathcal{U}}_k = [\tilde{u}_k, \tilde{u}_{k+1}, \tilde{u}_{k+2}]$.[1] The former matrices will be non-singular provided the three column vectors are linearly independent, i.e., each portion of the loop must be non-planar. If $\mathcal{U}_k$ is non-singular, then we can uniquely define K linear transformations $\mathcal{T}_k = \tilde{\mathcal{U}}_k \mathcal{U}_k^{-1}$ which satisfy $\tilde{u}_k = \mathcal{T}_k u_k$. In general, however, there may not be a single linear transformation $\mathcal{T}$ between the loops. If $\mathcal{T}$ exists, then $\mathcal{T}_k = \mathcal{T}$ for all k.

Since, in general, such a transformation between arbitrary loops does not exist, we perform a least-squares fit to the bounding loop data by minimizing the non-negative function

$$F(\mathcal{T}) = \frac{1}{2} \sum_{k=1}^{K} | \tilde{u}_k - \mathcal{T} u_k |^2 . \tag{7}$$

Clearly, if $\mathcal{T}$ in (2) exists, then it mimimizes F. To find $\mathcal{T}$, set $\partial F / \partial \mathcal{T}_{ij} = 0$ to get three uncoupled linear systems that can be written as

$$\mathcal{T} \mathcal{M} = \mathcal{F} \tag{8}$$

with

$$\mathcal{M} = \sum_k (u_k \otimes u_k) , \tag{9}$$

$$\mathcal{F} = \sum_k (u_k \otimes \tilde{u}_k) , \tag{10}$$

[1] Since the loops are closed we can achieve periodicity by letting $x_{k+K} = x_k$ and $\tilde{x}_{k+K} = \tilde{x}_k$.

where $\otimes$ is the vector outer product. $\mathcal{T}$ is determined provided $\mathcal{M}$ is non-singular but, unfortunately, $\mathcal{M}$ is singular for the important case of planar loops.

Definition: a loop L is **planar** if there exists $n \in R^3$ such that $n \cdot u_k = 0$ for all k. Suppose that n exists for loop L. Then $\mathcal{M}n = \sum(u_k \cdot n)u_k = 0$. Hence the null-space of $\mathcal{M}$ is non-trivial and so $\mathcal{M}$ is singular.

To circumvent the problem of planar loops, redefine the set of vectors u_k, $\tilde{u}_k$ by

$$u_k = x_k - (2c - \tilde{c}), \tag{11}$$

$$\tilde{u}_k = \tilde{x}_k - c. \tag{12}$$

It is easy to show that $\sum u_k = \sum \tilde{u}_k = K(\tilde{c} - c)$. Note that $u_k = (x_k - c) + (\tilde{c} - c)$, giving a clear geometric interpretation. Note that for the sweep problem there will not be a vector n that is normal to all the u_k unless the geometry is pathologic. We wish to apply the least-squares fit above to find a relation of the form

$$\tilde{u}_k = \mathcal{T} u_k. \tag{13}$$

Summing this relation over k we find that $\mathcal{T}(\tilde{c} - c) = \tilde{c} - c$. Putting (11-12) into (13), we find that

$$\tilde{x}_k - \tilde{c} = \mathcal{T}(x_k - c). \tag{14}$$

We have thus arrived at relationship (6), which in turn implies (2), as desired. Note that if $\mathcal{T}$ is the identity matrix, then one gets loop translation. Note also that if F is not zero at the minimum, then $\mathcal{T}$ does not necessarily send $\{x_k\}$ to $\{\tilde{x}_k\}$.

A new linear transformation is calculated in Sweep Tool for each layer of the advancing front based on consecutive boundary loops from the linking surfaces.

4. Layer Smoothing

The algorithm outlined in the previous section was implemented in Sweep Tool and was found effective on a wide variety of problems, including rotatable geometries (see Figure 1).

Despite the effectiveness of linear transformations, another algorithm is needed to sweep general two-and-one-half dimensional geometries. To handle general geometries, we resort to layer smoothing techniques in which each layer of quadrilaterals on the advancing front is smoothed independently of connections to nodes on the previous layer. Although fully three-dimensional smoothing is appropriate for the hexahedral meshes generated in sweep, it would be very inefficient to incorporate such a procedure into the advancing front method. An alternative is layer smoothing which provides a relatively efficient, if somewhat less robust, approach. A variety of layer smoothing methods, including Laplacian and isoparametric, were tried in Project with mixed success. Although Project smoothing was generally an improvement over the Project point placement algorithm described in section 2, two difficulties adversely impact mesh quality.

The first difficulty is illustrated by the example of sweeping a half-torus with the source surface meshed with a radially-biased circle primitive (Figure 2). Because the Project algorithms of section 2 produce meshes with negative Jacobian elements on rotatable geometries, layer smoothing is necessary. Laplace layer smoothing eliminates the negative Jacobians but the biasing on the source mesh is lost. More importantly, distorted hexahedral elements are created on the first layer of the advancing front

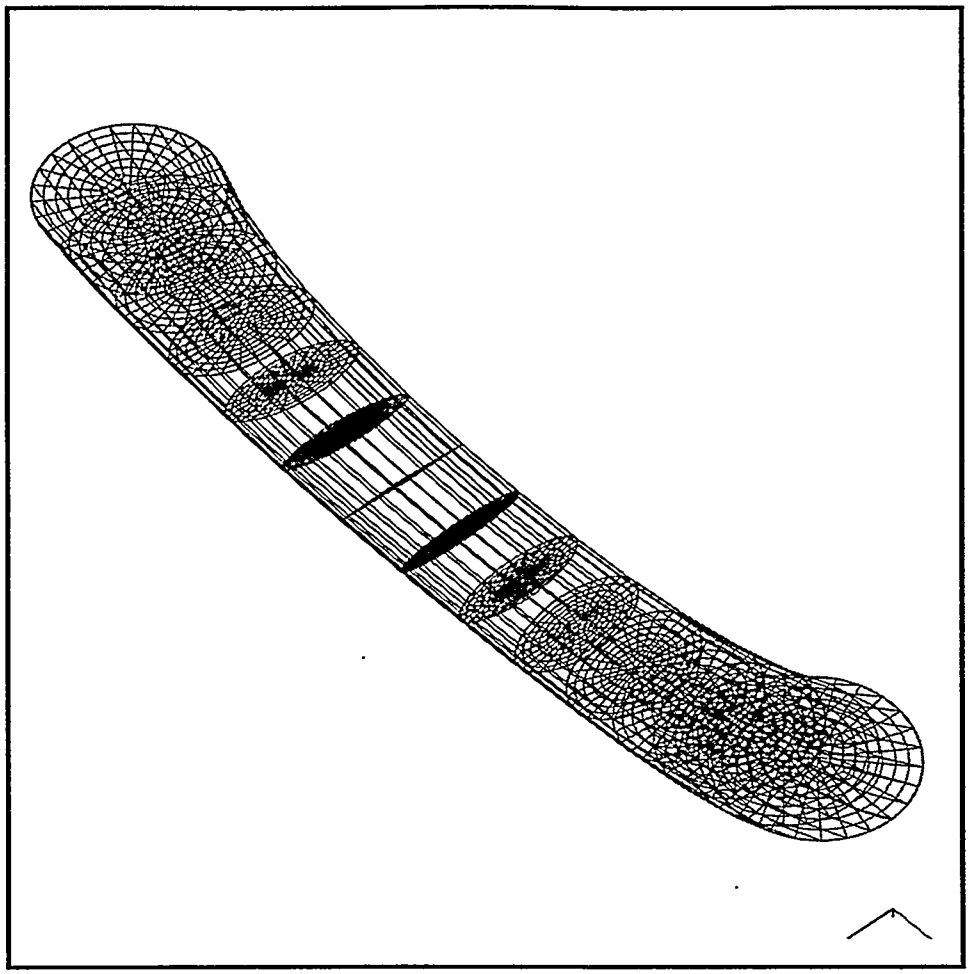


FIG. 1. Sweep Tool on Rotatable Geometry via Linear Transformation Approach

because there is not a smooth transition between the source and target meshes. In this example, the need for the original biasing of the source mesh to be preserved by the layer smoothing is obvious.

The second difficulty occurs on volumes having non-convex or non-simply connected source surfaces. Layer smoothing is absolutely necessary on the example in Figure 3 because no linear transformation between the source and target loops exists. Project with Laplacian layer smoothing produces a bad target mesh in this example (note the compression of the quadrilaterals next to the inner boundary). The second difficulty arises in many other important sweepable geometries.

Both difficulties are overcome by devising a robust smoother for Sweep Tool that takes mesh biasing into account and performs mesh copying/morphing[2]. For example, the mesh in Figure 1 can be obtained with Sweep Tool, with or without layer smoothing. Sweep Tool smoothing also solves the diffulty illustrated in Figure 2. Fig-

[2] Space limitations prevent discussion of the smoothing algorithm here. See [3] for a detailed presentation

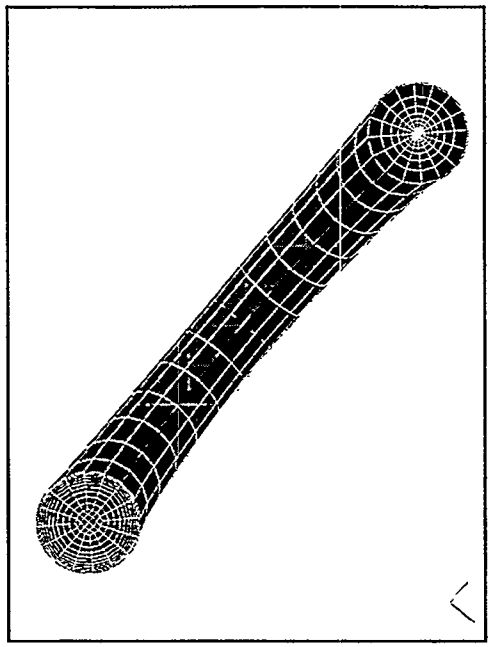


FIG. 2. Project Tool: Loss of Mesh Biasing due to Smoothing (Target on Right)

ure 4 shows the result of sweeping with Sweep Tool layer smoothing on the example in Figure 3. Experience with a series of realistic problems has shown it is possible to sweep a wide variety of general two-and-one-half dimensional geometries with this smoother.

If the curvature of a source surface is significantly different than the target surface, poor mesh quality may result even with layer smoothing. Curvature is somewhat preserved by layer smoothing if the bounding loop reflects the surface curvature but, in general, this is not the case. Research on methods to achieve good mesh quality for curved source and target surfaces is presently being pursued. The approach takes into account both source and target meshes.

Sections 3 and 4 give two alternative approaches to placement of the interior nodes, one involving linear transformations and the other layer smoothing. Although linear transformations are less general than layer smoothing, they remain useful because, when they exist, it is computationally much faster to generate them than to perform layer smoothing. Fortunately, (7) gives a direct measure of whether or not such a transformation exists: if $F = 0$, then the linear transformation exists and smoothing need not be used because good quality is guaranteed, as shown in the following result.

Proposition. Assume that a layer mesh has positive Jacobian at all mesh nodes. Let $\mathcal{T}$ be a linear transformation with positive determinant. Then $\mathcal{T}$ applied to this layer in the manner of (14) will result in another layer mesh which also has positive Jacobians at all of its nodes.

Proof. For each quadrilateral of the mesh let $\mathcal{U}$ be the matrix $[x_1 - x_0, x_2 - x_0, n]$ for some unit normal. We have $\mathcal{T}\mathcal{U} = \tilde{\mathcal{U}}$ with the determinant of $\mathcal{U}$ positive by assumption. Then the determinant of $\tilde{\mathcal{U}}$ is positive because it is the product of the

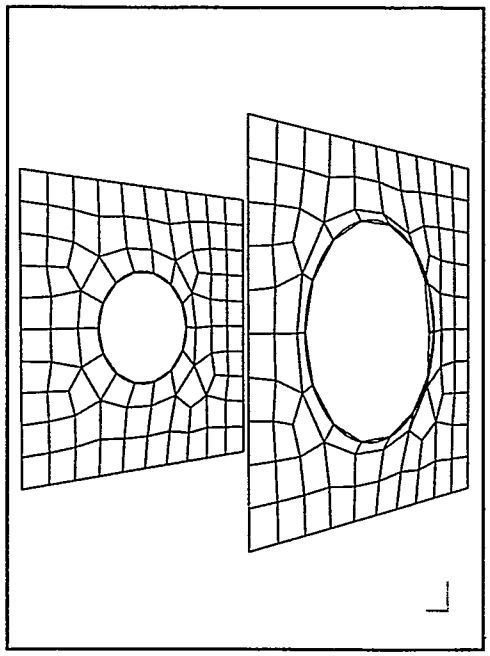


FIG. 3. Project Tool: Poor Quality Target Mesh (right) Resulting from Laplacian Layer Smoothing

determinant of two matrices which each have positive determinants. §

There are thus two criteria to determine whether or not smoothing is needed. If (i) $F < \epsilon$ where ϵ is some tolerance criterion and (ii) the linear transformation has positive determinant, then smoothing is not needed.

Occasionally, layer smoothing will fail to produce adequate mesh quality even with the new smoother because, although the geometry and topology are two-and-one-half dimensional, the mesh is fully three-dimensional. When layer smoothing fails, the situation can sometimes be salvaged by invoking the fully three-dimensional smoother as a post-processing procedure.

5. Additional Topics

We briefly discuss in this section some Sweep topics not directly related to that of interior node placement.

Poor quality meshes on the linking surfaces generally result in poor quality hexadrons generated by sweeping. It is therefore crucial to generate good meshes on the source and linking surfaces. Because surface meshing is outside the control of Sweep Tool, Sweep Tool checks the Jacobians of all the quadrilaterals on the linking surfaces before sweeping. If any Jacobians are negative, then sweep aborts because negative hexahedron Jacobians will surely result. Even when there are no bad quadrilaterals on the linking surfaces, one can still sweep out bad hexahedra due to problems with the linking surface meshes. Highly skewed meshes on the linking surface is a common cause of this problem.

Sweep Tool contains a topology-based mesh-matching algorithm because in some

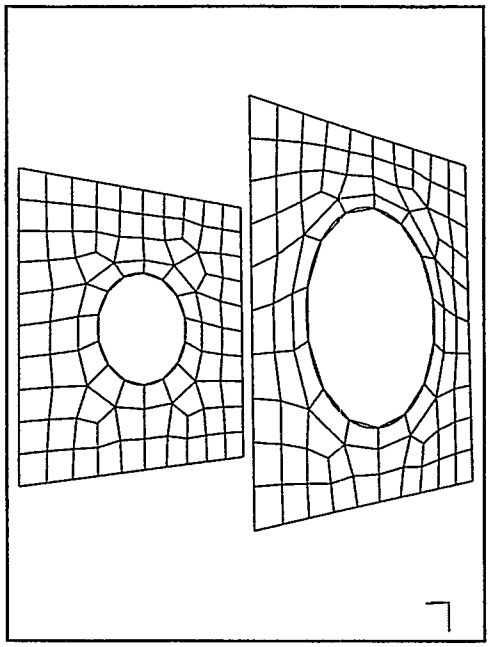


FIG. 4. Sweep Tool: Good Quality Target Mesh (right) Resulting from new Layer Smoothing Algorithm

instances it is advantageous to have the target surface meshed prior to sweeping. If the target mesh topology is incompatible with the source mesh topology, the sweep is aborted, otherwise each quadrilateral on the last layer of the advancing front is matched with the appropriate quadrilateral of the target mesh.

Multiple source surfaces are allowed in sweep, but multiple targets are not presently allowed.

Mesh generation rates of up to 3000 hexahedrons per second have been generated with Sweep Tool on a 300 MHz workstation, compared to 1800 hexahdrons per second for Project Tool. It is possible to achieve this through careful memory-speed tradeoffs to eliminate quadratic-time calculations.

6. Summary and Conclusions.

Sweep Tool provides a relatively fast and semi-automatic means of generating volume meshes on general two-and-one-half dimensional geometries. Interior point placement is crucial to the success of the algorithm.

Two approaches for the placement of interior node points have been suggested that are robust and highly effective. The linear transformation approach is fast and gives high quality meshes on translatable, rotatable, and scalable geometries. The layer smoothing approach is slower but can give high quallity on more general geometries. Two criteria are given which form the basis for an automatic method for determining if smoothing is needed.

Some geometries are border-line sweepable, i.e., even though the geometry is two-and-one-half dimensional, meshes on the linking surfaces conspire to make layer smoothing inadequate. Full 3D smoothing applied as a post-processing step may give

the desired mesh quality.

Further development of Sweep Tool technology is underway to improve mesh quality in the case of highly curved source and target surfaces. Research on improved algorithms for sweeping to multiple targets is also in progress. Of course, not all geometries are two-and-one-half dimensional so research must continue on meshing the general problem.

Acknowledgements

The author would like to thank the members of the CUBIT team and the codes users at Sandia National Laboratories for many helpful suggestions.

REFERENCES

[1] T. Blacker, *The Cooper Tool*, Proceedings of the 5th International Meshing RoundTable, pp13-29, Pittsburg, Pennsylvania, October 10-11, 1996.

[2] P. Knupp, *On the invertibility of the isoparametric map*, Comp. Meth. Appl. Mech. Engr., **78**, p313-329, 1990.

[3] P. Knupp, *Applications of Mesh Smoothing: Copy, Morph, and Sweep on Unstructured Quadrilateral Meshes*, submitted for publication.

[4] L. Mingwu, S. Benzley, G. Sjaardema, T. Tautges, *A Multiple Source and Target Sweeping Method for Generating All Hexahedral Finite Element Meshes*, Proceedings of the 5th International Meshing RoundTable, p217-225, Pittsburg, Pennsylvania, October 10-11, 1996.

[5] D. White, L. Mingwu, S. Benzley, G. Sjaardema, *Automated Hexahedral Mesh Generation*, Proceedings of the 4th International Meshing RoundTable, pp165-176, October 16-17, Albuquerque, New Mexico, 1995.

[6] David White, private communication.

The Geode Algorithm: Combining Hex/Tet Plastering, Dicing and Transition Elements for Automatic, All-Hex Mesh Generation

Robert W. Leland[1]
Darryl J. Melander[1]
Ray W. Meyers[1]
Scott A. Mitchell[1]
Timothy J. Tautges[1]

Abstract. A new all-hexahedral meshing algorithm, referred to as "Geode", is described. This algorithm is the combination of hex/tet plastering, dicing, and a new 26-hex transition element template.The algorithm is described in detail, and examples are given of problems meshed with this algorithm.

keywords. Geode; hexahedral mesh generation; transition template.

1. Introduction

Finite element analysis is used to model physical phenomena in a wide variety of disciplines, including structural mechanics and dynamics, heat transfer, and computational fluid dynamics (CFD). To perform these analyses, the problem domain must first be discretized into one-, two- or three-dimensional elements. While the finite element method allows many types of elements, tetrahedra and hexahedra are typically used; furthermore, hexahedra are considered more accurate for a given cost for some types of analyses, particularly in the non-linear regime. However, generating all-hexahedral meshes for typical problems has proven quite difficult.

To date, no automatic all-hexahedral meshing algorithm has been found which delivers both complete automation and high-quality meshes, both of which are critical to acceptance of such an algorithm in the FEA field. There have been many research efforts to find a suitable algorithm. Whisker weaving[1] operates in the mesh dual to find the topology of an all-hex mesh, then smooths to locate nodes in physical space. While this algorithm has shown promise, it has not yet achieved the required robustness for typical problems. Other algorithms investigated by Schneiders et. al[2] and by researchers at Cray Research[3] have proven robust, but have generated meshes which tend to be large and which have poorest quality near region boundaries. Recent efforts at Sandia National Laboratories have shown that hex meshes can be obtained by slicing automatically-generated tetrahedral meshes into hexahedra, and that these meshes may be suitable for some analyses, but these meshes have yet to be tested for a wide variety of non-linear analyses[4].

This paper describes the Geode algorithm, which generates all-hexahedral meshes automatically for arbitrary geometries. The algorithm is based on hex-tet plastering, an advancing-front, hex-dominant meshing algorithm which generates hex elements from the boundary inward followed by tetrahedra on the interior[5]. Geode inserts a "necklace" layer of hex elements between the hexes and tets, then dices the ensemble to produce an all-hex mesh. A special transition element is used to dice the necklace layer of hexes [13]. The algorithm is in the early stages of development, and some issues remain regarding the quality of the elements generated in the transition layer in realistic problems. These are being diligently researched, and our feeling is that the algorithm shows much promise.

The remainder of this paper is organized as follows. Section 2 describes the hex-tet plastering algorithm and the dicing techniques in more detail. Section 3 describes the combination of these algorithms and the necessary additions for the

1 Parallel Computing Sciences Dept., Mail Stop 0441, Sandia National Laboratories, Albuquerque, NM 87185. The authors were supported by the Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy, Office of Energy Research, and work at Sandia National Laboratories, operated for the U.S. DOE under contract No. DE-AL04-94AL8500. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. DOE.

Geode algorithm. Section 4 gives examples of the Geode algorithm, and Section 5 gives conclusions and future directions.

2. Key Meshing Technology

The Geode algorithm can be thought of as combining two key meshing algorithms, hex/tet plastering and hex and tet dicing, with a special transition element layer. In this section, we describe each of these pieces.

2.1 Hex-Tet Plastering

In the past, Sandia has researched two very different all-hex meshing algorithms, plastering[6] and Whisker Weaving[1]. After comparing the relative merits of the two, we decided to discontinue research on plastering and devote our efforts to Whisker Weaving. While there has been substantial progress on the Whisker Weaving algorithm since then[7], we still do not have a fully robust and general algorithm. In recognition of this, Sandia recently began pursuing a shorter-term goal of developing an algorithm which would give hex-dominant meshes while retaining the characteristics of suitable (but not optimal) mesh quality and complete automation. This effort resulted in the hex-tet plastering algorithm[5]. The hex-tet plastering algorithm consists of three steps, which are described below.

A. Plastering

The plastering algorithm can be considered a volume analogy to the paving algorithm[8]. Starting with a closed boundary of quadrilateral elements, each element is projected into the volume to form a hexahedron. Elements are projected layer by layer to maintain an advancing-front mesh, which has the desirable characteristic of placing highest quality elements near volume boundaries. As the meshing front proceeds inward, special merging operations are done to stitch the front(s) together (see [6] for more details of this process.)

At some point in this process, the plastering algorithm is unable to proceed further without producing intersections with another part of the front or forming hexes whose quality is below a prescribed minimum. At this point, the plastering algorithm exits and the hex-tet plastering algorithm proceeds to the next stage.

B. Hex-Tet Transition

There are several possible methods to transition between a hex and tet mesh; these methods can be classified by whether they are conformal or non-conformal[2]. We have implemented one conformal transition method and two non-conformal methods; the choice of which method to use can be made at runtime in our implementation.

A non-conforming interface between hexes and tets can be obtained simply by converting each quadrilateral bounding the void to several triangles; we have implemented both a two- and four-triangle transition. These transitions differ in their robustness and in the number of tetrahedra ultimately generated [5], but leave a triangle-bounded void on which to start the tet mesher.

To transition between hex and tet meshes in a conformal manner, additional element types are necessary. The two most common elements used in this case are triangular prisms and pyramids. We choose to use only pyramids for the transition in Hex-Tet Plastering (the Geode algorithm uses a different transition method described later), since they are simpler to use in this context. In particular, we have implemented the method proposed by Canann et. al [9]. In this method, tetrahedral meshing is performed from a two-triangle non-conforming interface; afterwards the tetrahedra next to the interface are converted to pyramids using a combination of transition primitives and tetrahedra recombinations.

2 A conformal mesh is one where elements sharing more than one node also share all the edges and faces comprised by the shared nodes. Such meshes are also said to be contiguous. A non-conformal mesh is one where these conditions do not hold.

C. Tetrahedral meshing

Following the transition region generation, the remaining region is meshed with tetrahedra, using the tetrahedral meshing engine in CUBIT[10].

Studies have shown the hex-tet plastering algorithm to be quite robust while generating hex and tet elements of high quality[5].

2.2 Hex and Tet Dicing

The hex dicing algorithm was implemented in CUBIT to allow the generation of large (multi-million element) meshes[11]. This algorithm simply divides each hexahedral element in a mesh into smaller elements, optionally projecting the newly generated nodes onto the geometric boundary where appropriate. If the coarse mesh is conformal and all-hexahedral, the fine mesh will have the same characteristics.

Tetrahedral meshes can be subdivided into hexahedra as well by using a simple primitive. Each triangle is subdivided into three quadrilaterals, a node is inserted in the interior of the tetrahedron, and edges are made which connect that node to the four mid-face nodes (see Figure 1). This splits the tetrahedron into four hexahedra. If the initial mesh is all-tetrahedral and fully conformal, the fine mesh will be conformal as well, but will consist only of hexahedral elements.

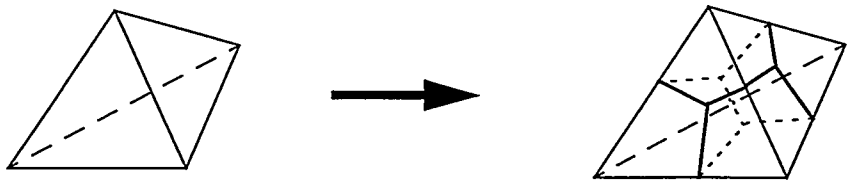


Figure 1. Tetrahedral dicing or subdivision.

It is interesting to note that tet dicing could be considered as an alternative method for generating all-hexahedral meshes (when combined with an automatic tetrahedral meshing algorithm). Conventional wisdom has in the past held that these meshes would have insufficient quality for finite element analysis; however, recent studies have shown otherwise for selected analysis types[4].

2.3 Transition Layer

Hex-tet plastering has been shown to yield hex-dominant meshes of sufficient quality, and we described earlier how hex and tet elements can each be split into finer, all-hexahedral elements. A natural question is, then, can these methods be combined to yield an automatic, all-hexahedral meshing algorithm that generates fully conformal meshes? We emphasize that this method would generate high-quality, well fitted hexes along the boundary, a very desirable property that has eluded previous attempts to produce an automatic hex meshing algorithm.

If a hex-tet plastered mesh which uses a two-triangle transition is diced using the methods described above, the result is a non-conformal interface. Dicing a hex-tet plastered mesh which uses pyramids to get a conformal interface gets us closer to the goal, yielding diced hex and tet regions separated by pyramids whose faces are diced into quadrilaterals. We initially thought this would lead to the desired algorithm; the only missing piece was a transition element corresponding to this quad-bounded pyramid. Unfortunately, obtaining an all-hexahedral mesh for this geometry and topology has proven problematic. Currently, the minimal all-hexahedral mesh for this arrangement consists of approximately 128 hex elements. In fact, this has been posed as an "open problem" to the meshing community for some time[12], and has not been solved to date.

We therefore developed an alternative approach to using pyramids on the hex-tet interface. We use a "necklace" layer of hexes consisting of a single hex for each quadrilateral on the initial void boundary. Each necklace has one face on the original hex boundary, and the opposite face on the new, tet-facing boundary. The remaining four faces are shared by other necklace hexes only. The transition problem then becomes one of transitioning between the diced hex-facing quad and the two-triangle-then-diced tet-facing quad (see Figure 2, left). The solution to this problem is described in another paper[13], however, the result is the transition element shown in Figure 2, center, right.

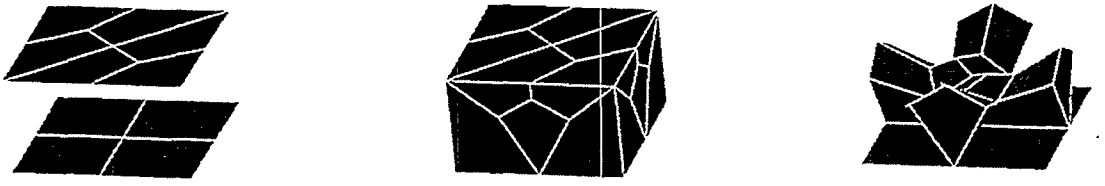


Figure 2. Constraints on transition element (left); proposed transition element meeting those constraints (center, right); from [13].

3. The Geode Algorithm

Using the algorithms described earlier, we can now construct the Geode algorithm. The required steps, as implemented in CUBIT, are:

1. Plaster
2. Generate necklace layer of hexes
3. Convert void-bounding quads to two triangles
4. Tet mesh remaining void
5. Dice hexes and tets
6. Insert transition template into each necklace hex and smooth locally

The sequencing of these steps can be modified; for example, we experimented with generating the necklace layer either before or after the generation of tets. In addition, we have explored the generation of the necklace by pulling faces back from the void instead of projecting the faces into the void. We have found that the procedure described above, using projection-based necklace generation, works best with the plastering algorithm available in CUBIT[14].

4. Examples and Applications

Examples of meshes generated by the Geode algorithm are shown in Figure 3 and Figure 4. These are early results meant to characterize the sort of geometries we can presently mesh automatically and with reasonable quality using the Geode algorithm. The current limitation is that we have difficulty ensuring that all elements in the transition layer have positive Jacobians. We are working on advanced smoothing algorithms to remedy this situation. Since the underlying components of the Geode algorithm are already capable of automatically meshing substantially more complex geometries than those shown, we are optimistic that, with improved smoothing, the Geode algorithm will have practical impact on difficult meshing problems.

It has been shown by Meyers et. al[5] that hex-tet plastering generates an increasing volume fraction of hexes as initial bounding mesh size is decreased, as expected; this characteristic also benefits the Geode algorithm, by minimizing the number of transition elements and concentrating them in region interiors, away from principal loads and/or wall effects.

There are many potential applications of the Geode algorithm which we intend to explore. The most obvious is as an automatic, all-hexahedral meshing algorithm for arbitrary geometries. This algorithm would be appropriate for generating hex meshes for optimization loops or for parallel meshing. Previously only tet-based hex meshes could be used in these settings because of the level of automation required.

There is another application of Geode which is not as obvious, but which we feel is perhaps more significant, and this is as another tool in the hex-meshing toolbox. Since Geode generates hex meshes which have suitable but not optimal quality, it should be used only when necessary. Given the availability of simpler algorithms which generate meshes of higher quality, e.g. mapping and sweeping, along with techniques and tools for decomposing geometry into sub-regions, Geode could be used only on the difficult sub-regions, leaving other regions to be meshed with simpler algorithms. We believe that this combination of algorithms will substantially reduce the amount of geometry decomposition (and hence time) required to mesh complex assemblies, while maintaining the high overall mesh quality which motivates the need for hex meshes in the first place. We note that a hex meshing algorithm based on tet-dicing could not be used in this type of application.

A key capability for this application is that of generating an all-hexahedral mesh starting from a fixed, all-quadrilateral boundary. In this context, volumes meshed with Geode would need to be meshed first, or a method to de-refine a quadrilateral mesh would be necessary, since Geode dices the initial quadrilaterals bounding the void.

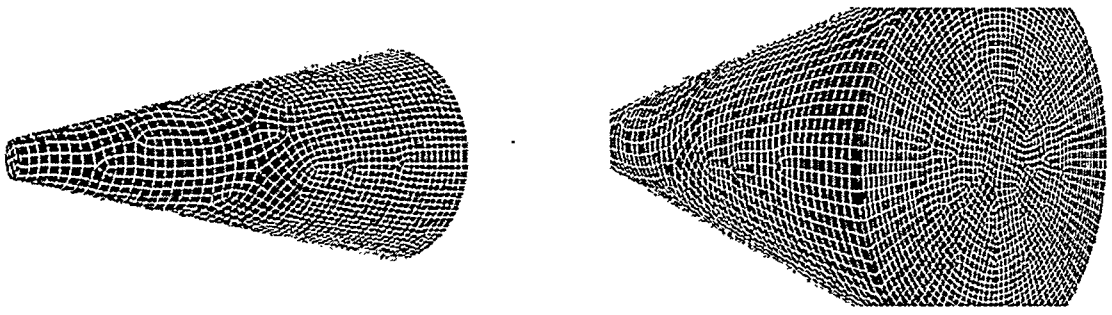


Figure 3. Two views of a conical geometry automatically hex meshed by the Geode algorithm.

Figure 3 demonstrates application of the Geode algorithm to a conical geometry; two views of the same geometry are shown. Axial sweeping is possible but not desirable here because the high aspect ratio between the source and target surfaces would result in elements of inappropriate size on one of these surfaces. Azimuthal sweeping is also possible but leads to wedge shaped elements also considered undesirable for the intended analysis. The Geode algorithm was able to generate a valid all-hex mesh with fairly uniform size starting from a Paved surface mesh. In this instance no Plastering was used, so the surface mesh displays one face from each of the elements in the transition layer. When Plastering was employed, the surface to which the transition layer is attached becomes less regular, and construction of valid transition elements becomes more difficult, resulting in the generation of small fraction with negative Jacobians.

On the left in Figure 4 a three dimensional duct geometry is shown. This could, after minimal decomposition, be meshed with traditional sweeping, so this is merely a demonstration problem. The Geode algorithm was able to generate an all-hex mesh of about 24k elements for this geometry. Two of these elements had negative Jacobians in the case where no plastering was used, and four did when Plastering was used. In the latter case, the worst Jacobian was 10e-3, and 16 elements had Jacobians less than 10e-2.

On the right in Figure 4, we show a cube geometry with circular imprints. The imprints prevent straight-forward sweeping and make this a difficult problem requiring substantial decomposition with the standard approach. The Geode algorithm was able to automatically generate a valid, all-hex mesh with no negative Jacobians for this geometry.

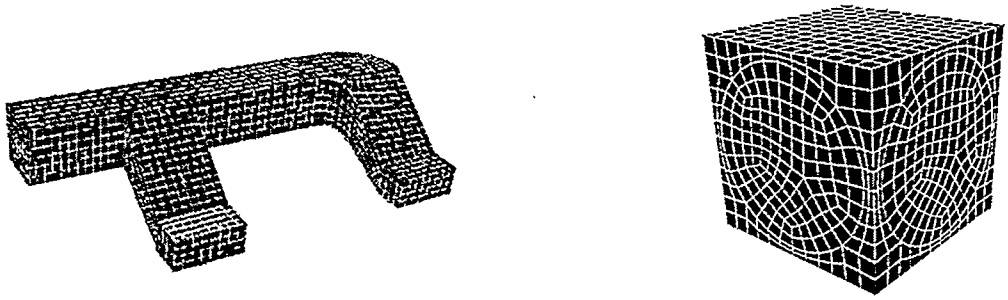


Figure 4. Examples of all-hex meshes generated automatically be the Geode algorithm.

Here again it was necessary to turn off Plastering to generate a fully valid mesh. With Plastering, one of the 11.5k hex elements generated had a negative Jacobian.

5. Conclusions and Future Work

The hex-tet plastering algorithm has been shown to be robust and to generate mixed-element meshes of suitable quality for a variety of example geometries. Combining this tool with a diced-hex to diced-tet transition template results in an automatic all-hexahedral meshing algorithm. After investigating a number of transition template options, including pyramids and necklace layers of hexes, we have used a transition template based on a necklace layer of hex elements. The resulting Geode algorithm has been implemented in CUBIT, and preliminary results are encouraging. We believe that with additional work on associated smoothing techniques, the Geode algorithm will be capable of automatically generating all-hex meshes of quality suitable for FEM analysis for a meaningful class of geometries.

The Geode algorithm can be used where automatic hex-meshing is required, for example in optimization loops and for parallel mesh generation. Another powerful application is in combination with geometry decomposition and other well-known meshing algorithms; this toolbox approach shows potential in substantially reducing the time to mesh for complicated assemblies.

Going forward, we intend to refine the combination of algorithms in Geode in order to increase the quality of the resulting meshes. In particular, the methods used to generate the necklace transition layer need further investigation. We also intend to use Geode meshes for typical analysis problems at Sandia in order to explore their suitability for engineering analyses of various types. Finally, we will also explore the criteria used to determine when to use Geode versus performing further geometry decomposition to get more mappable and sweepable sub-regions.

References

[1] Timothy J. Tautges, Ted D. Blacker, Scott Mitchell, "The Whisker Weaving Algorithm: a Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes", Int. J. Numer. Methods Eng., 39:3327-3349 (1996).

[2] R. Schneiders, R. Schindler, F. Weiler, "Octree-based Generation of Hexahedral Element Meshes", Proceedings of the 5th International Meshing Roundtable, Sandia National Laboratories report SAND96-2301, Oct. 10-11, 1996, Pittsburgh, Pennsylvania.

[3] R. Taghavi, "Automatic, Parallel and Fault Tolerant Mesh Generation from CAD", Eng. with Comp. (1996) 12:178-185.

[4] Ed Boucheron, Randy Weatherby, "ALEGRA - Meshing Approaches", http://www.sandia.gov/1431/mesh_showcase.html.

[5] Ray J. Meyers, Timothy J. Tautges, Philip M. Tuchinsky, "The 'Hex-Tet' Hex-Dominant Meshing Algorithm as Implemented in CUBIT", to appear in 7th International Meshing Roundtable, Detroit, MI, October 1998.

[6] Stephenson, M. B., S. A. Canann, T. D. Blacker, "Plastering: A New Approach to Automated 3D Hexahedral Mesh Generation -- Progress Report I", SAND89-2192, Sandia National Laboratories, 1990.

[7] N. Folwell, S. Mitchell, "Reliable Whisker Weaving via Curve Contraction", submitted to 7th International Meshing Roundtable, Detroit, MI, October 1998.

[8] T. D. Blacker, M. B. Stephenson, "Paving: A New Approach to Automated Quadrilateral Mesh Generation", Int. J. Numer. Methods Eng., 32:811-847 (1991).

[9] Owen, S. J., Canann, S. A., Saigal, S., "Formation of Pyramid Elements as a Means of Maintaining Tetrahedra to Hexahedra Conformability," AMD-Vol. 220 Trends in Unstructured Mesh Generation, ASME, p. 123-129, 1997.

[10] D. R. White, Timothy J. Tautges, "Tetrahedral Meshing Using CUBIT", CU-TIPS #3 (to be written), Sandia National Laboratories, 1998.

[11] Darryl J. Melander, Timothy J. Tautges, Steven E. Benzley, "Generation of Multi-Million Element Meshes for Solid Model-Based Geometries: The Dicer Algorithm," AMD-Vol. 220 Trends in Unstructured Mesh Generation, ASME, p. 131-135, 1997.

[12] R. Schneiders, "An interesting open problem", http://www-users.informatik.rwth-aachen.de/~roberts/open.html.

[13] Scott A. Mitchell, "The All-Hex Geode-Template for Conforming a Diced Tetrahedral Mesh to any Diced Hexahedral Mesh", to appear in 7th International Meshing Roundtable, Detroit, MI, October 1998.

[14] T. D. Blacker et al., "CUBIT Mesh Generation Environment, Volume 1: User's Manual", SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994.

Practical Industrial Experiences

WHEN MESH GENERATORS REQUIRE ENHANCEMENTS

Gregory J. Elbert
Midsize Car Division Headquaters
Mail Code: 480-210-152
General Motors Corporation
300001 Van Dyke Avenue
Box 9020
Warren, MI 48090-9020
lzxbsv@hqs.mid.gmeds.com

David A. Field and William H. Frey
Research and Development Center
mail Code: 480-106-285
General Motors Corporation
30500 Mound Road
Warren, MI, 48090-9055
aes4ur@daf.ma.gmr.com
frey@.gmr.com

Abstract

Advances in automatic mesh generation have made the generation of finite element meshes easily available. Commercial mesh generators can create two dimensional meshes composed of triangles and quadrilaterals on simple planar polygons or complex three dimensional meshes of tetrahedra and hexahedra. However, they do not always work properly or guarantee adequate element quality. Due to the inability of a commercial preprocesser to correctly develop a mesh, this paper relates a revisit to triangular meshes on simple planar polygons. This return to basics led to new measures and visualizations of the quality of meshes and to the use of mesh relaxation, a method of improving initial two dimensional triangulations. Ultimately more than one hundred thousand elements were created in less than two minutes.

1. Introduction

A serious problem in analyzing the conjugate heat transfer inside an automotive engine compartment occurred when a commercial software package could not produce a mesh without crashing a powerful graphics workstation. During a 24 hour attempt at generating approximately 120,000 triangles the software generated anomalies which were removed and handled individually thereby enabling the software to generate approximately 99.8% of the mesh.

An impressive 99.8% success rate does not give much incentive for a supplier to remedy the problem. This familiar scenario of generally very satisfactory software and of little effort to correct a bug related to a specific user (as opposed to a bug encountered by a large number of users) gives incentive for users to have local backup mesh generators available. Fortunately the availability of such mesh generation software enabled us to resolve our difficulties.

This paper relates our experiences with our own industrial strength mesh generator. Without divulging specific details that would identify the commercial software, this paper discusses execution times of generating meshes, visualizing the quality of meshes, numerical measures for the shape of triangles and quadrilaterals, and the use of mesh relaxation as a method for improving initial planar triangulations.

2. Simple Domains and Initial Meshes

In the automobile industry the configurations of automotive components under the hood of cars and trucks represent complicated geometries. Mathematical models that simulate the distribution and flows of heat and air in these engine compartments must first represent the geometry of the surfaces of each automotive component under the hood as well as the walls of the compartment. The model under consideration here represented each relevant surface as the union of relatively large planar triangles and quadrilaterals whose four edges did not have to be coplanar. Call these triangles and quadrilaterals macro triangles, macro quadrilaterals and macro polygons.

Each macro polygon underwent further decomposition into smaller triangles. The decomposition started by a partition of each edge of the macro polygons into segments of equal length determined by dividing the edge by a nominal length and rounding to the nearest integer number of segments. The vertices along each edge of the macro polygons served as the input to the commercial generator of triangular meshes. In addition to the aforementioned failure mode, this commercial mesh generator also generated some triangles with no area. We have ascertained that the mesh generator uses an advancing front in the three dimensions in which the macro polygons lie. For some macro polygons, typically macro triangles, the code continues to create triangles until it exceeds storage limits.

Since the heat transfer and finite element analysis codes in use prefer equilateral triangles and since planar Delaunay triangulations avoid extremely acute angles, our own planar triangulations used the Delaunay based selective mesh refinement technique [2]. At the heart of this technique lies a spacing function which measures at each vertex the average length of edges connected to it. As its first step this method creates a Delaunay triangulation of a polygon using only the vertices on its boundary. Triangles that have poor shapes as measured by the normalized shape ratio (twice its inradius divided by its circumradius) become candidates for inserting a new vertex into its interior. Evaluating the spacing function at each vertex of the triangle then determines whether to insert a new node and whether to create an equilateral triangle if the triangle has an edge on the boundary of the macro polygon. The selective mesh refinement technique also imposes an upper bound on the number of vertices inside the macro polygon as well as linearly or nonlinearly scaling the distribution of vertices away from the boundary of the macro polygon. The location of a new vertex inside a triangle involves the spacing function, the inradius and the circumradius of a triangle. After inserting the final vertex into the macro polygon the selective mesh refinement technique performs a Laplacian smoothing of the triangulation. Figures 1 and 2 illustrate the use of linear and quadratic spacing functions on a macro triangle and a macro quadrilateral.

The four edges of a macro quadrilateral often do not lie in the same plane. Consequently we first map the quadrilateral onto a central plane determined by the cross product of the two diagonals and the centroid of the macro quadrilateral. The mapping of the macro quadrilateral onto the plane proceeds as follows. Since a bilinear finite element mapping originally defines the macro quadrilateral, every vertex on its boundary has an easily calculated parametric coordinate. At each vertex calculate the normal to the macro quadrilateral directly from the bilinear mapping and along this normal project the vertex onto this central plane. The polygonal image of the macro quadrilateral then serves as the

input to the selective mesh refinement algorithm. The inverse mapping of the triangulation onto the macro quadrilateral uses a Newton-Rhapson iteration to locate interior vertices. The location of the boundary vertices are already known.

When the discretized boundary of a macro quadrilateral has the same number of nodes on each edge, an extremely quick alternative triangulation method exists. Use this number to subdivide the unit square into a grid of squares whose diagonals completes a triangulation of the unit square. The bilinear finite element mapping which defined the macro quadrilateral then maps the triangulation onto the macro quadrilateral. Since the bilinear mapping can grossly distort a good triangulation of its domain, this alternative was not considered. The macro quadrilateral can also be too warped, i. e. deviate too much from planarity even for the mapping that uses local normals. This possibility prompted the definition of warpage presented in the next section.

3. Quality of results

The bane of triangulating thousands of macro polygons is that invoking a single method such a Laplacian smoothing to improve the Delaunay triangulations may not suffice for every one of the thousands of macro polygons. Many opportunities for the improvement of the Delaunay triangulation exist at the end of the selective refinement method. Having codes for Laplacian smoothing [7], Laplace-Delaunay smoothing [1] and mesh relaxation [3] gave us many options. Whereas for special meshes choosing one or combinations of options usually becomes a matter of visual observations and subjective approval, one choice for thousands of meshes cannot wait on treating special cases however few they may be percentage wise. How to improve initial triangulations and when to stop improving them are major decisions to be made.

To help decide on a course of action for improving initial meshes it helps to study representative examples and use visual displays. Keeping geometries simple, as done in this paper, emphasizes the algorithms and separates them from geometric anomalies. Since the macro polygons in our application also had this simplicity we used them. Our scheme for improving initial meshes used mesh relaxation as its last step.

Except for Laplace-Delaunay smoothing and mesh relaxation a literature search does not reveal much insight beyond using Laplacian smoothing and its variants [4,5]. Laplacian smoothing moves vertices to the centroid of the triangles that share it. The fact that Laplacian smoothing does not change the connections of vertices to one another limits its effectiveness. This limit motivated the development of Laplace-Delaunay smoothing which still moves a vertex to the centroid of the triangles that share it but alters the connections if the location does not guarantee a Delaunay triangulation. The Delaunay part of the smoothing creates equilateral triangles whenever possible and carries over into three dimensions. More importantly for three dimensions, this type of smoothing allows incremental Delaunay triangulations algorithms to continue inserting additional vertices.

Mesh relaxation carries the idea of establishing new connections to the extreme by acting on the observation that vertices inside a triangulation having only equilateral triangles connect to six other vertices. Mesh relaxation tries to give every vertex inside a triangulation a connection with exactly six other vertices whenever possible. Figure 3 and the following formulas summarizes the methodology.

Let d_1 and d_2 denote the degrees of the endpoints of the j^{th} edge and let d_3 and d_4 be the degrees of the vertices opposite the j^{th}edge in the two triangles sharing the j^{th} edge. The formula,

$$E_j = d_1 + d_2 - d_3 - d_4 \ , \tag{1}$$

defines the *relaxation index* , E_j, of the j^{th} edge. Swapping the j^{th} edge as shown in Figure 3, changes the degrees of its endpoints so that its new new relaxation index, E_j', is given by

$$E_j' = d_3' + d_4' - d_1' - d_2' = (d_3 + 1) + (d_4 + 1) - (d_1 - 1) - (d_2 - 1) \ . \tag{2}$$

In a triangulation of equilateral triangles, the relaxation index of each interior edge equals 0. It can be shown that the process of swapping an edge whenever its index *equals or exceeds* a threshold of 3 guarantees that the process of swapping edges ends. In [3] it was demonstrated that using a threshold of 2 can further improve a mesh but no longer guarantees convergence.

Mesh relaxation clearly swaps only interior edges. However, if an endpoint of an edge lies on the boundary of the macro polygon then its index as defined in (1) must reflect that no triangles exist outside the polygon. Mesh relaxation resets the degree of the boundary node by adding to the degree the largest number of possible equilateral triangles that could share the node and lie outside the macro polygon.

Since swapping edges takes no consideration of the local geometry, in order to reproduce a valid triangulation after terminating the swapping of edges, mesh relaxation relocates vertices via Laplacian smoothing. Furthermore, if three endpoints of the swappable edges lie on the the boundary of the macro polygon a swap can produce a triangle with no area or even a triangle outside the macro polygon; see Figure 4 which uses a threshold of 2. Special precautions must be taken to avoid this possibility.

Figures 5 and 6 show examples of applying mesh relaxation applied after selective mesh refinements from strictly linear to strictly quadratic spacings. Figures 7 and 8 show surfaces which plot for each vertex an average shape of the triangles that share the vertex. The expression

$$\frac{8\sqrt{3}A^3}{(abc)^2} \ , \tag{3}$$

where A denotes the area to the triangle and a, b and c denote the lengths of its edges, determines the numerical measure of shape. Geometrically this measures divides the area of a triangle by the area of the equilateral triangle that shares the same circumcircle. This measure satisfies the properties of a fair measure. It yields a value of zero for all degenerate triangles, it is scale invariant, it is bounded and it is normalized to 1 for the equilateral triangle. For visualization however the Figures invert the surfaces so that the highest peaks correspond to the worst triangles.

Since the four edges of macro quadrilaterals may not lie in the same plane, the deviation from planarity can be extreme. An extremely warped macro quadrilateral manifests itself when its mapping into its midplane produces a self intersecting planar polygon. Triangulation algorithms fail on such polygons. Measure this deviation from planarity, called *warpage*, by the larger of the two dihedral angles between the pair of triangles formed by each diagonal of the macro quadrilateral. These are also the angles between the normals

to the triangles as shown in Figure 9. Note that this warpage differs from Robinson's [6] definition which applies to hexahedral finite elements. As far as we know, our definition of warpage is new. Some macro quadrilaterals have so much warpage that their presence compromises the integrity of any numerical simulation based on the model. In these cases the creator of the geometry must correct the geometry.

As measured by the commercial code, subtracting the fair measure in (3) from one defines *skewness*. Table 1 gives statistics on the skewness for a sample set of substructures such as an oil filter, solenoid etc.. These substructures consist of a set of macro polygons that describe the exterior geometry of an automotive component.

3. Conclusions

Our algorithm decreased the wall clock time for generating approximately 120,000 triangles from almost 24 hours to less than one and a half minutes. We anticipate that some even more complicated problems may take up to 5 minutes.

REFERENCES

1 D. A. Field, *Laplacian smoothing and Delaunay triangulations*, Comm. Appl. Num. Meth, 4, 709-712 (1988).

2 Frey, W. H., *Selective refinement: a new strategy for automatic node placement in graded triangular meshes*, International Journal for Numerical Methods in Engineering, **24**, 2183-2200 (1987).

3 W. H. Frey, and D. A. Field, *Mesh relaxation: a new technique for improving meshes*, International Journal for Numerical Methods in Engineering, **31**, 1121-1133 (1991).

4 S. R. Kennon and D. A. Anderson, *Unstructured grid adaptation for non-convex domains*, in Numerical Grid Generation in Computational Fluid Mechanics '88, S. Sengupta, J. Hansin, P. R. Eiseman and J. F. Thompson, eds., Pineridge Press, Ltd., Swansea, U. K., 599-609 (1988).

5 V. N. Parthasarathy and S, Kodiyalam, *A constrained optimization approach to finite element smoothing*, Finite Elements in Analysis and Design, **9**, 309-320 (1991).

6 J. Robinson, *Quadrilateral and hexahedron shape parameters*, Finite Elements in Analysis and Design , **16**, 43-52 (1994).

7 Winslow, A. M., *Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh*, Journal of Computational Physics, Vol. 2, 149-172 (1967).

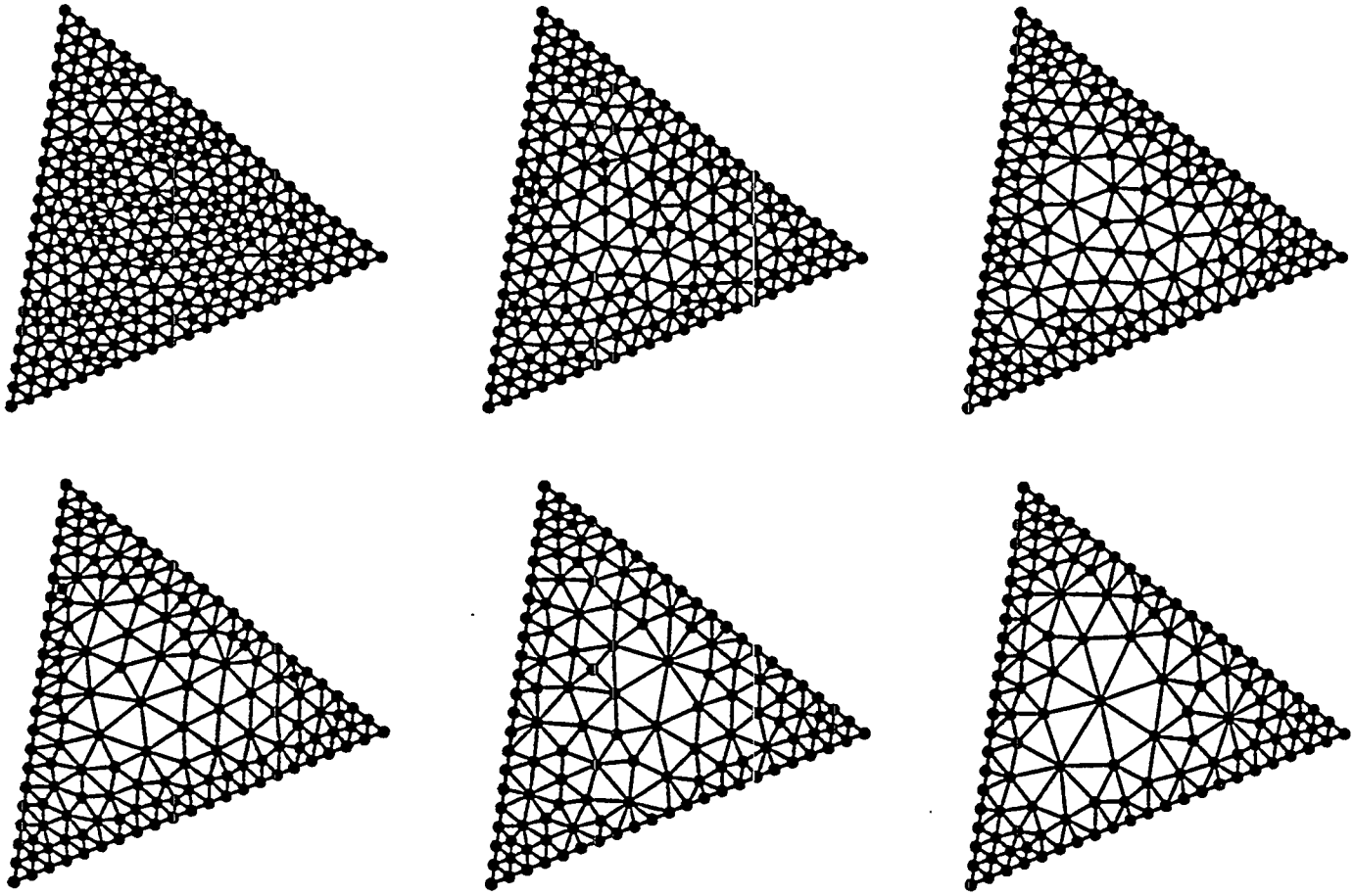


Figure 1 Six convex combinations of the linear and quadratic spacing functions for a macro triangle. Starting from strictly linear at the top left to strictly quadratic at the lower right, the corresponding number of triangles are 485, 329, 273, 235, 197 and 183.

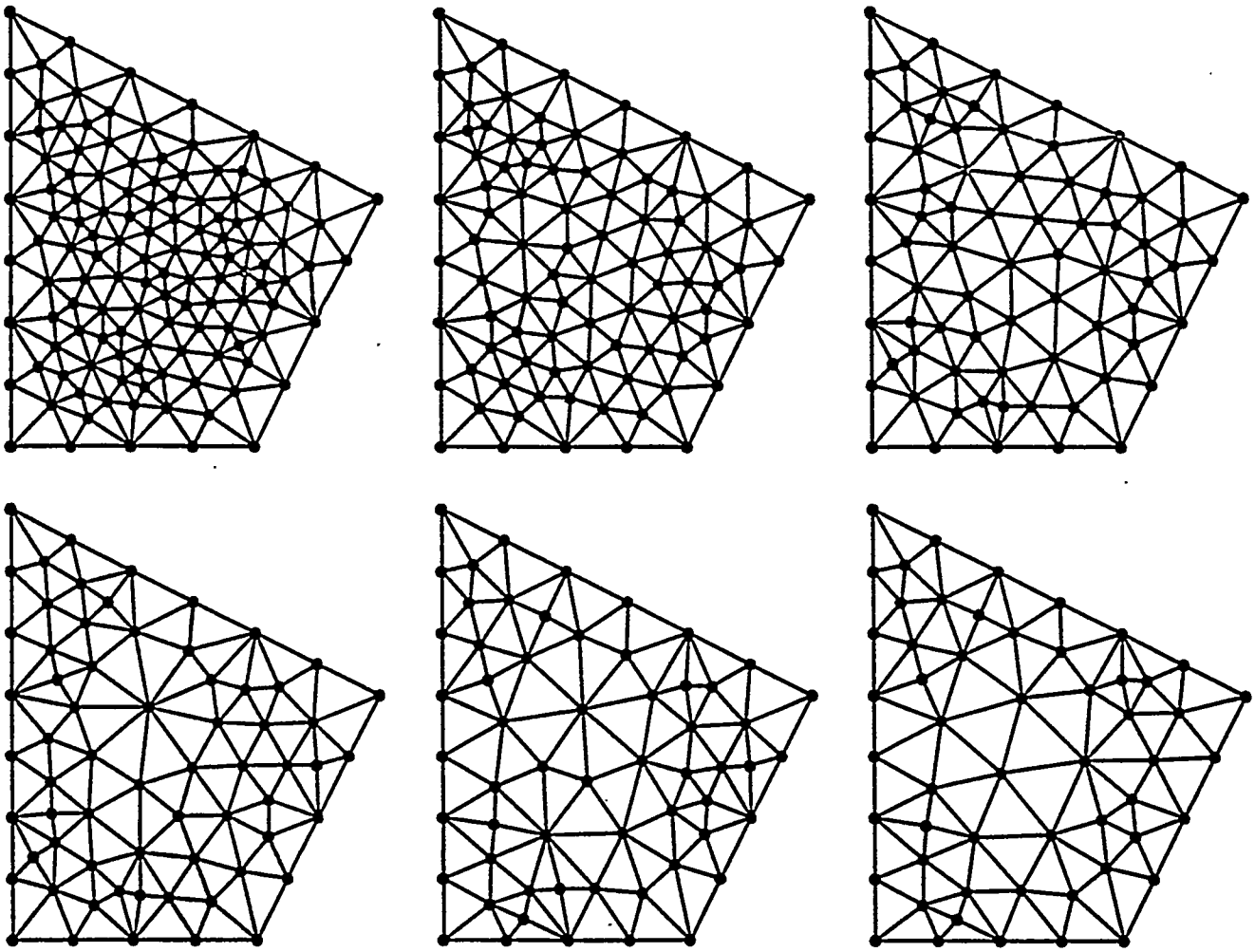


Figure 2 Six convex combinations of the linear and quadratic spacing functions for a macro quadrilateral. starting from strictly linear at the top left to strictly quadratic at the lower right, the corresponding number of triangles are 225, 153, 123, 107, 93 and 85.

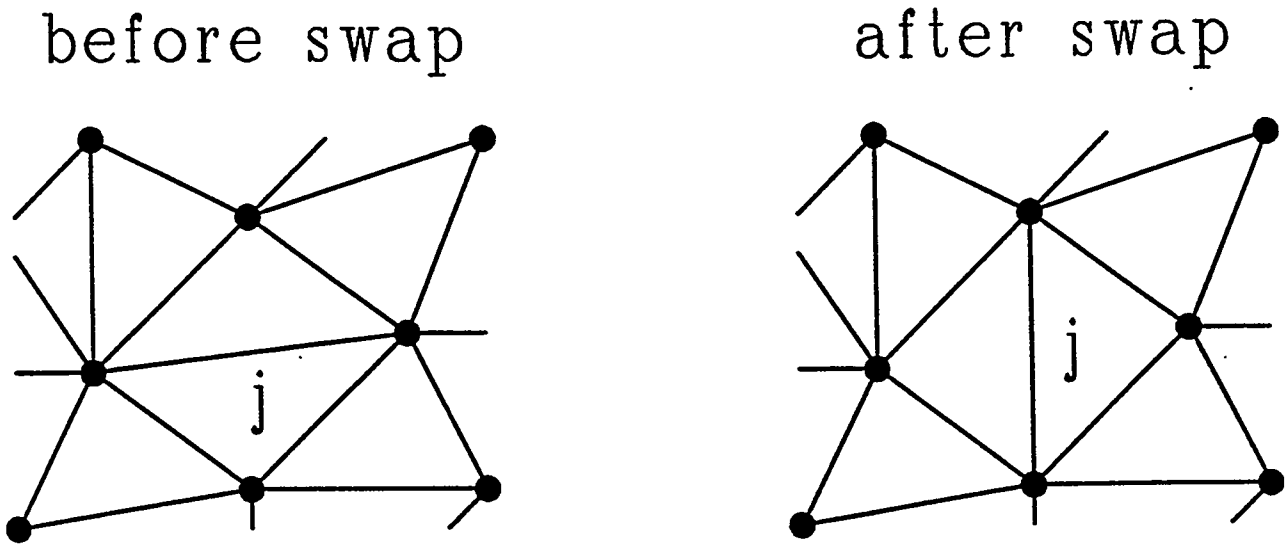


Figure 3 Swapping the j^{th}-edge.

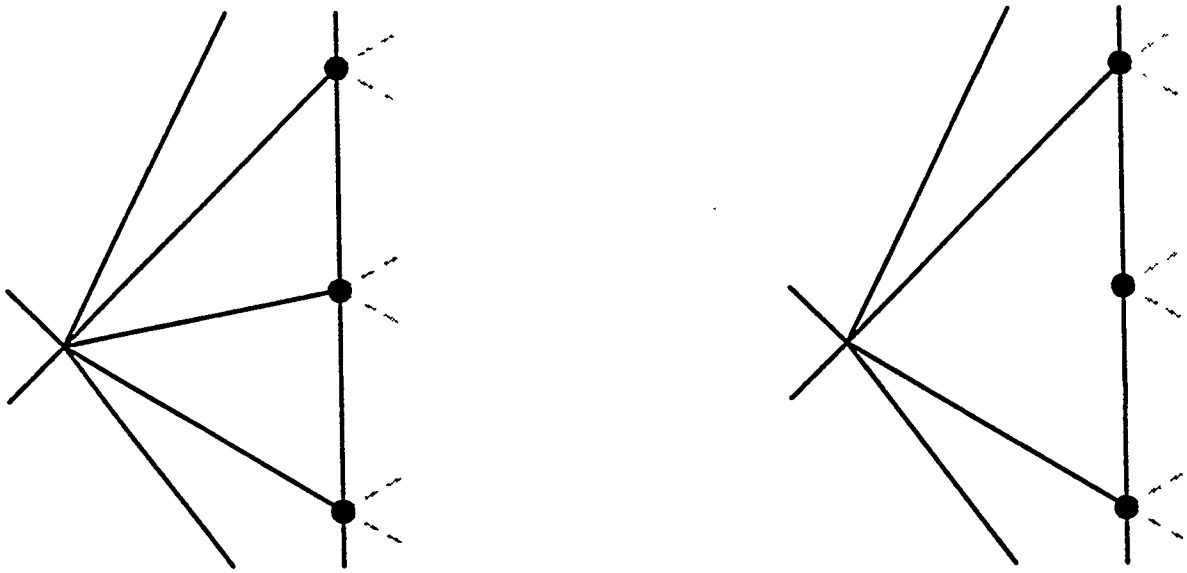


Figure 4 An illegal swap using a threshold equal to two. The three vertices on the boundary, shown with their pseudo edges to indicate their degree, form a triangle with no area after the swap.

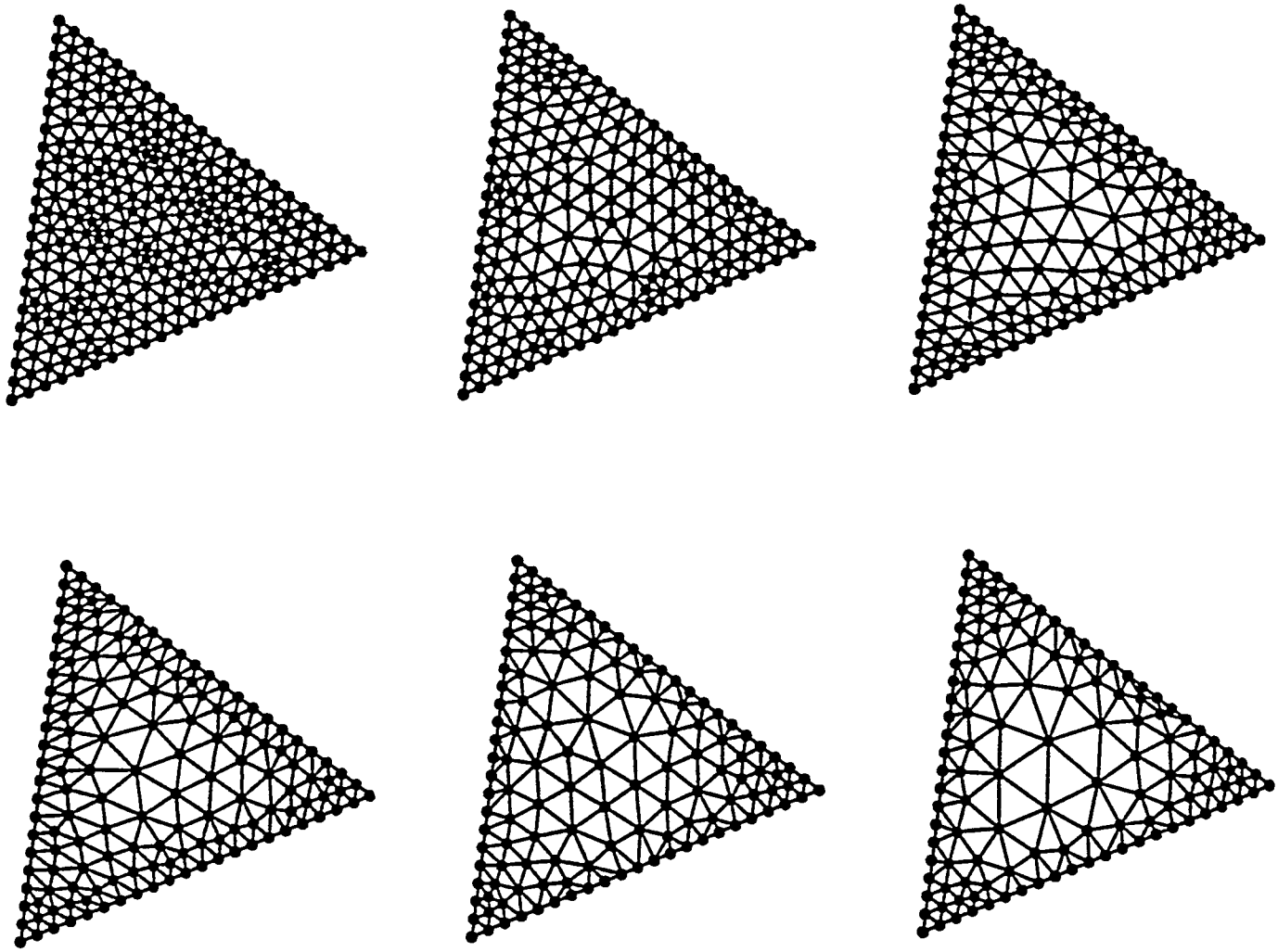


Figure 5 Mesh relaxation set at a threshold of 3 applied to the meshes in Figure 1.

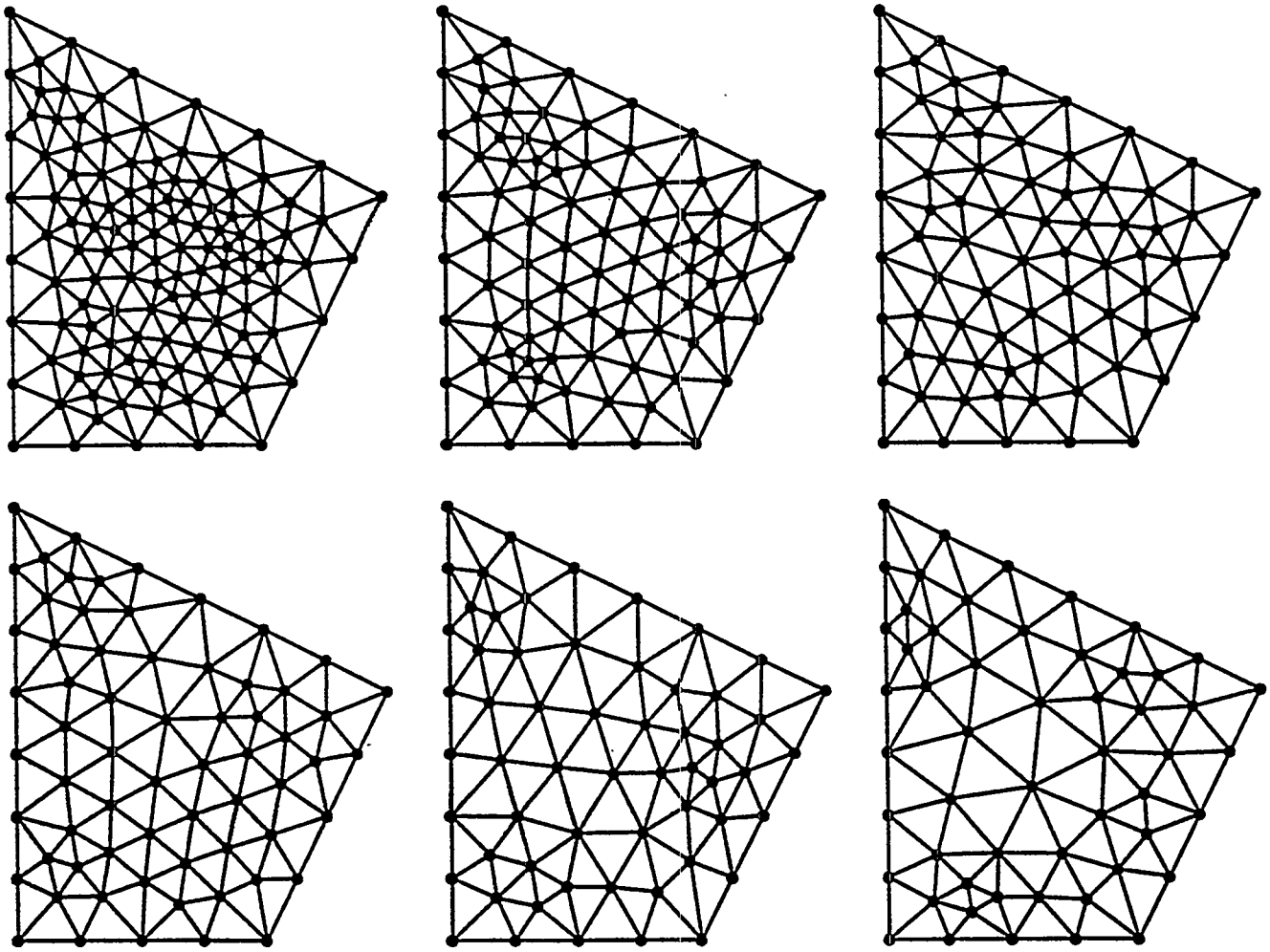


Figure 6 Mesh relaxation set at a threshold of 3 applied to the meshes in Figure 2.

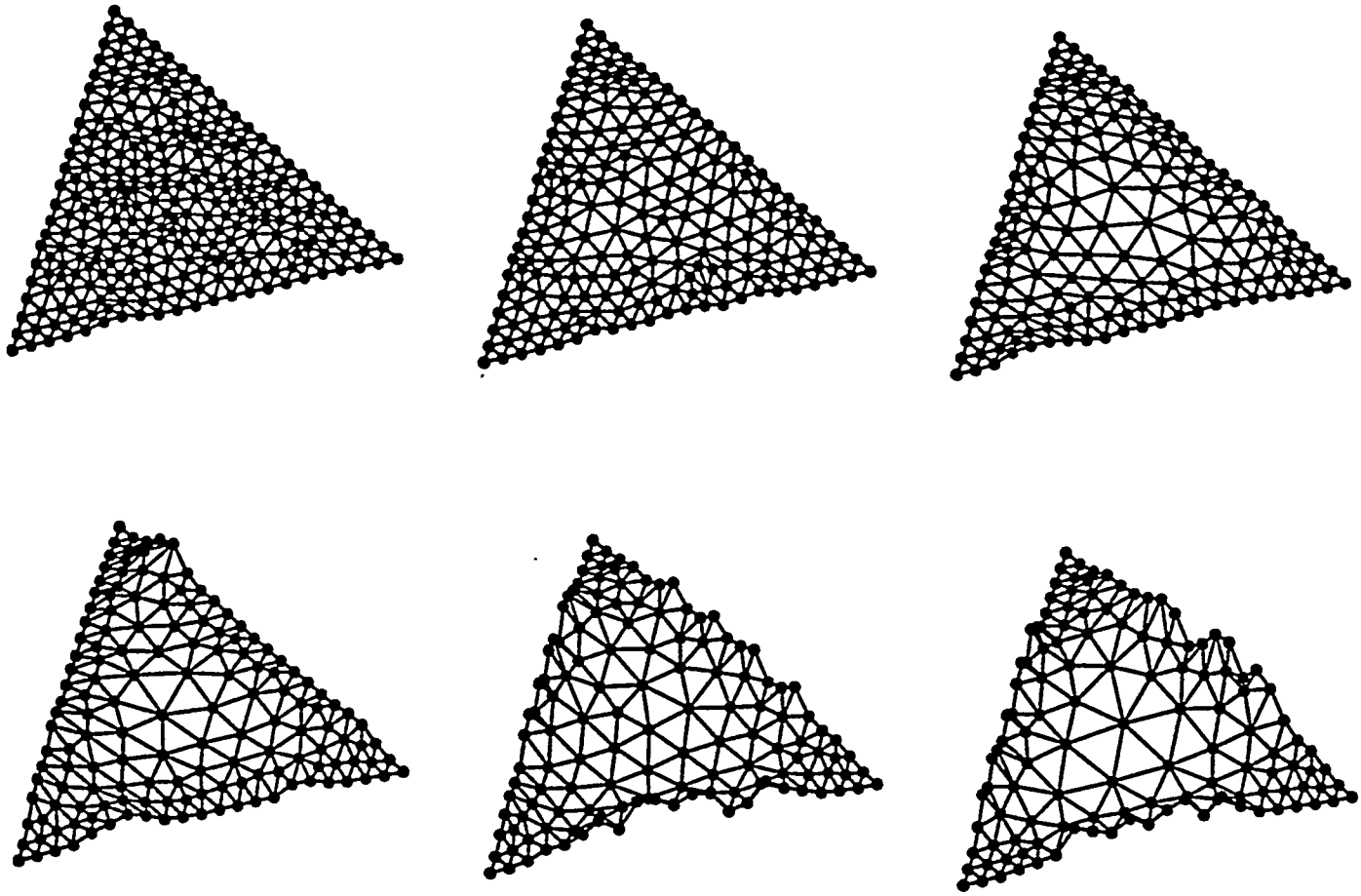


Figure 7 Surface representations of the average shape of the triangles in the meshes in Figure 5.

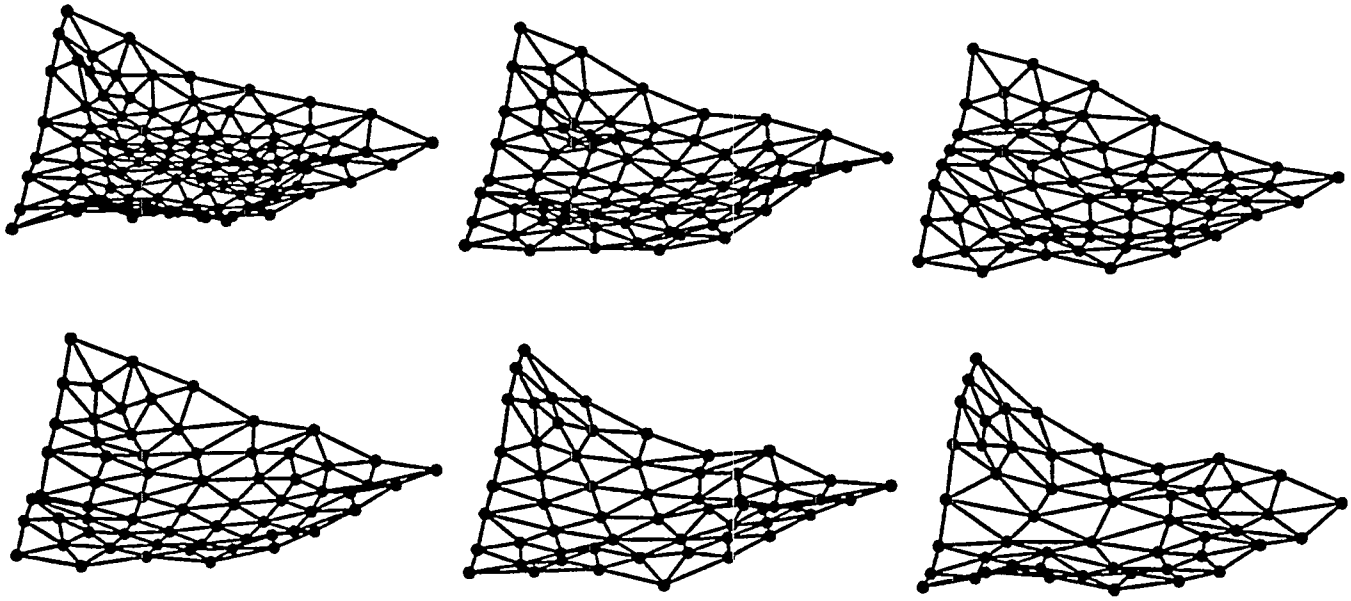


Figure 8 Surface representations of the average shape of the triangles in the meshes in Figure 6.

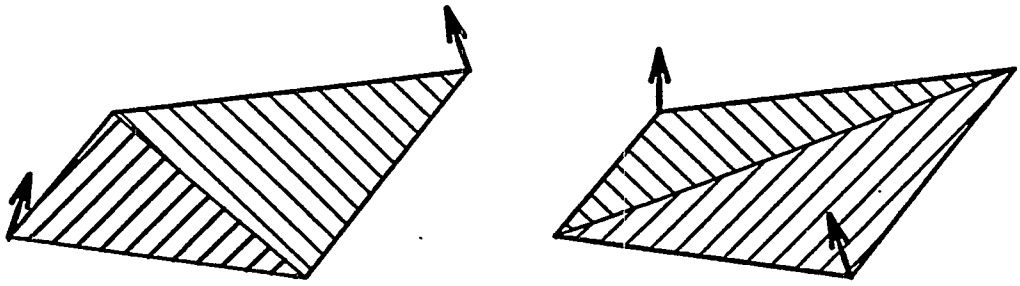


Figure 9 The maximum angle between the vectors determines the warpage of the macro quadrilateral.

Values of Skewness				
component	*minimum*	*average*	*maximum*	*no. triangles*
1	0.00037	0.14365	0.58624	1370
2	0.00059	0.18148	0.49137	464
3	0.00039	0.18084	0.70366	424
4	0.00005	0.13742	0.66515	776
5	0.00000	0.11219	0.60481	1921
6	0.00027	0.12176	0.46121	912
7	0.00005	0.10144	0.55485	1157
8	0.00067	0.13638	0.43044	875
9	0.07223	0.23888	0.53271	72
10	0.00002	0.06295	0.53217	4538
11	0.00017	0.12032	0.54666	472
12	0.00014	0.16449	0.69480	324
13	0.06348	0.21831	0.24316	88
14	0.00807	0.06673	0.23164	28
15	0.00009	0.11715	0.35934	280
16	0.00010	0.11355	0.56067	1241
17	0.00259	0.13071	0.39600	282
18	0.00116	0.09888	0.56517	640
19	0.00028	0.14951	0.62304	426
20	0.00115	0.16466	0.52079	15
21	0.00024	0.11256	0.30759	406
22	0.00045	0.16846	0.62780	693
23	0.00123	0.25564	0.60610	819
24	0.00031	0.23000	0.65333	1857
25	0.00035	0.15548	0.76621	1113
26	0.00000	0.12691	0.62588	2371
27	0.00004	0.20021	0.60771	642
28	0.00001	0.08590	0.44569	2528
29	0.00000	0.07407	0.51914	2696
30	0.00010	0.05732	0.43333	729
31	0.00010	0.11120	0.90035	1430

Table 1

CAD Model Quality
Holds the Key for Analysis

Doug Cheney
International TechneGroup Incorporated
Milford, OH

FEA users have experienced their share of horror stories when performing analysis on problem CAD models. Hidden errors in these files represent a major obstacle for the Finite Element Analysis process. Informal studies reveal that analysis users are wasting up to 70% of their time re-working CAD files before analysis can even begin. The bad news is that with the growing use and complexity of these models the situation will only get worse. The good news is that a solution now exists to solve these problems.

Today 3D CAD models are driving many downstream product development processes. Finite Element Analysis, Rapid Prototyping, NC programming, data exchange, and other downstream applications rely to a growing extent on the direct use of CAD models to streamline processes. This delivers significant savings in time and money while boosting quality. Unfortunately many of these models contain hidden errors or anomalies. The results of bad CAD files can be unprecedented levels of inefficiency, days of lost time and productivity, loss of design intent, and ultimately inferior product quality. Fortunately there is a solution.

Regardless of cost or the vendor who developed it, every CAD system in use today is susceptible to producing invalid files when creating complex 3D surfaces and solids. At the same time, even the most experienced designer can occasionally create CAD models containing hidden errors or anomalies. Often these problems do not surface until well downstream of design presenting more than just minor inconveniences for those who receive them. CAD model problems can bring product development processes to a grinding halt as the corrupt model is shipped back to the designer to be fixed. A more

common scenario, however, might have the downstream user forced to make corrections or even reconstruct the model entirely. Obviously this practice can have a detrimental impact on altering design intent along with the obvious ramifications associated with lost time and cost over-run.

Milford, Ohio's International TechneGroup Incorporated (ITI) met recently with manufacturers throughout industry to determine the impact of model quality problems on the product development process. It was found that up to 70% of the man-hours spent during Finite Element Analysis and Simulation, for example, were wasted correcting geometry problems. In addition, Rapid Prototyping, Numerical Control tool path generation, and Product Data Exchange functions were likewise spending 50%, 20%, and 20% of their time respectively reworking geometry problems. With this in mind it is easy to see why CAD model quality issues have been identified as the biggest hurdle facing industry today.

Common Types of Model Quality Problems

As computer-aided design, engineering, and manufacturing tools continue to expand in terms of usage and complexity, downstream waste attributed to poor model quality will become an even larger problem. Such model quality problems can be generally categorized into three areas: Structure, Accuracy, and Realism.

Structure

Structural problems include loop orientation inconsistencies, missing geometry and self-intersecting geometry among others. Structural errors violate the solid modeling application's own rules for what constitutes a correct model.

Structural errors can also cause modeling programs to crash without warning. Often this can occur some time after the actual error has been made. Structural errors can cause programs for finite element mesh generation, numerically controlled toolpath generation, and intersystem translation software to behave unpredictably.

An example of a structural error is a face with an edge that isn't shared by another face. In a manifold solid volume, such edges shouldn't exist because they cannot physically be manufactured. They occur because some solid modeling programs allow non-manifold topology as a midpoint to creating complete volumes. If these errors are not fixed, manufacturing and analysis programs will reject the models.

Accuracy

Accuracy requirements place limits on gaps between geometric entities such as vertices, edges, and faces that are adjacent. They can also limit the minimum sizes of trimmed entities such as edges, faces, and regions.

Non-trivial gaps occur because intersections of curves between non-planar surfaces are approximated in most solid modelers. Approximations are used when the precise intersection between two geometric entities (faces, curves of intersection, vertices where intersection curves meet) is too complex to compute exactly. Solid modelers use different tolerances to compute the maximum deviation allowed between topological entities.

If the deviations between entities are too large, toolpath and finite element mesh generation programs can fail. They can uncover gaps in geometry that are too small to be seen in shaded or hidden line images of a model. The translation between programs can also fail if the maximum allowable tolerances between surfaces and edges in the exporting program are larger than those of the importing program.

All CAD modeling systems must balance the accuracy (precision) of models with the amount of geometric information required to define them. Extremely precise models require complex and large data structures to define them. In general, the smaller the gaps, the smaller edges and faces may become in complex models.

Realism

Realism errors render a part "unmanufacturable" due to physical limitations. Realism errors include transition cracks and sliver faces. Transition cracks in solid models, like physical cracks in engineering materials, are nearly invisible gaps between features of a model. Like physical cracks, they may not extend completely through the object. Slivers are small, elongated faces that are generated by the system to patch between larger surfaces in a model.

Additional restrictions on the realism of model features are added by many concurrent engineering applications such as FEM, NC toolpath generation, and rapid prototyping. For example, these tools are very sensitive to unrealistic features such as sliver faces, minute edges, and very acute angles between edges at a vertex.

The Causes of CAD Model Problems

Obviously not every model has problems, but as you see, those models that do contain errors can cause significant delay and additional effort. Model quality problems are rooted in a variety of contributing factors that range from bugs in the CAD modeling system to data translation software errors. Additionally, modeling techniques that do not anticipate the needs of downstream shape-based applications can create anomalies. These human-error types of problems arise when CAD designers don't fully understand the geometric requirements of downstream applications or have no efficient way to validate CAD models against those requirements to identify potential problems.

CAD model problems or anomalies are caused by a series of three factors:

- *User technique*
- *CAD application algorithms*
- *Part design and manufacturing requirements*

User technique can be the order in which you add a feature or create geometry. There are some commercial products that attempt to address this. In general, each company and type of product requires a unique set of rules. For example, a rule could be that the

distance between parallel features for machined parts must be greater than 0.001 inches. This is great for machined bulkheads on airplanes, but not for microcircuits. So each company and each product has to have unique rules. This same approach has been tried with electronic drawing checkers over the last 10-15 years. None of these products were very successful due to the expense of customization. Also, even when all rules are satisfied the anomalies can still occur. Additionally, the user can unintentionally introduce problems as a result of schedule constraints combined with last minute design changes. The errors can easily go undetected or are allowed to be released in the model rather than run the risk of missing the release schedule.

A second factor contributing to these anomalies is the algorithms within the CAD applications themselves. This is especially true when users approach the limits of the mathematics behind the CAD system. For example, it's easy to understand how a round off error can cause a gap between two lines that are supposed to intersect 10,000" out in space. One line might say the end point is 10,000.00001" and the second line might say the end point is 9,999.99999". The result is a gap of 0.00002". For an airplane or a skyscraper this is probably acceptable - for a microcircuit, it is not.

The third factor that contributes to these anomalies is the product design and manufacturing requirements. Sometimes a justifiable anomaly exists that prevents the use of a downstream application. This anomaly is necessary to support the intent of the design. An example could be a very small face to transition between two other features. In this case, the anomaly must exist in the design and may need to be removed in a secondary/reference model to support the downstream process. However, the removal or modification of geometry needs to be the product team's decision and not the decision of a single person in the process.

What's the Answer?

Regardless of the reasons why they exist, the fact remains that if model quality problems aren't effectively resolved, downstream process simply can't work. The solution is to

implement a Model Quality program. Through such a program designers can better identify and resolve the source of these problems through a combination of improved modeling techniques, better software bug reports, and "real world" user requirements for current research in this field.

Likewise the downstream software user should implement Model Quality as well. This will enable the recipient of the file to quickly analyze model and locate problem areas before production begins. The file can now be returned to the designer with errors highlighted for quick turnaround. The downstream user may choose to make the modifications depending on the severity of the errors.

Model Quality Results

Implementing a model quality testing program can yield breakthrough levels of improvement. Validating CAD files prior to release significantly reduces model rework time. ITI estimates that model rework time can be cut by 50% in downstream Finite Element, Product Data Exchange, and Numerical Control applications. That number jumps to savings of up to 80% for Rapid Prototyping functions.

As CAD models continue to take on a more broad and significant role in the development of new products, it is naturally imperative that these files flow smoothly into downstream applications. Today technology exists to ensure the integrity, reliability and interoperability of CAD models throughout the product development process. By implementing a Model Quality program organizations can begin to reap the benefits of a tangible concurrent engineering environment.

Model Quality Tools

Off the shelf software now exists making CAD model quality a reality. This technology analyzes CAD models detecting problems that may prohibit a smooth flow into downstream applications. The software also allows users to present their CAD system provider with detailed reports pertaining to bugs in their software. Once identified many

of these problems can be resolved by the designer in the early stages of development where changes can be incorporated quickly and at less cost with the full knowledge of the design intent of the part. Other problems, such as software bugs or translation errors, will require longer term efforts by CAD/CAM/CAE vendors and researchers. However, whereas in the past, these bugs or errors were communicated through user phone calls and hotline reports, a comprehensive model quality control program can provide more detailed and valuable insight into identifying these problems.

It is important that these tools do more than just check models against "rules". They should provide the CAD user with the power and flexibility to analyze the model for conformance to a wide variety of applications *and* specific CAD/CAM/CAE system requirements. This allows the designer to anticipate system restrictions and ensure that models created will flow seamlessly into all downstream applications. In short, this allows unrestricted interoperability to be designed into the model.

A Big Payoff for Finite Element Analysis

While a solution for model quality is of obvious benefit to the designer and in fact holds value for all CAD/CAM/CAE software users throughout the product development process, those involved in analysis specifically stand to benefit from such technology. It is not uncommon for FEA software users to spend an inordinate amount of their time cleaning up or recreating problem CAD files – up to 70% of their time in some instances.

Implementing model quality checks allows FEA software users to efficiently locate problems before analysis significantly reducing model rework. By pinpointing problem areas the user can provide precise detailed information back to the designer to expedite changes. Additionally, if the user chooses he/she may make the necessary corrections at a fraction of the time usually required. This provides the user a method to then recheck the model to ensure that the problem was corrected and that no new problems were inadvertently introduced into the design. Those employing the process today also find

that such a program provides a means to better gauge the amount of time required for downstream functions allowing them to quote jobs more accurately.

Now, for the first time, downstream software users can utilize CAD models with greater confidence and be assured of spending less time cleaning up bad CAD files. Proven model quality software provides a practical solution for CAD/CAM/CAE software users throughout industry and bridges the interoperability gap enabling CAD models to continue to take on an expanded role throughout the product development process.

About the Author

Doug Cheney is Senior Model Quality Consultant for Milford, Ohio's International TechneGroup Incorporated (ITI). Doug works with organizations throughout industry to solve CAD model quality problems. Doug is also the author of ITI's CAD/IQ™. CAD/IQ is the world's premier CAD model quality and interoperability testing tool. You can read more about CAD Model Quality and CAD/IQ on the Internet at www.cadiq.com You can learn more about CAD Model Quality and CAD/IQ through on-line conferencing via the web. Contact Doug Cheney at 800-783-9199 for more details.

An Object Oriented Approach to Geometry Defeaturing for Finite Element Meshing

Anton V. Mobley, Michael P. Carroll, and Scott A. Canann
ANSYS, Inc.
275 Technology Drive
Canonsburg, PA 15317
{tony.mobley, michael.carroll, scott.canann}@ansys.com

Abstract. Automatic finite element mesh generation of CAD generated data has been a goal of finite element meshing codes for years. However, the lack of accuracy and the amount of detail in this data have made this a daunting task. In essence, the CAD data needs to be defeatured to overcome accuracy deficiencies and to remove excessive detail. In this paper, an object-oriented approach to automatic geometry defeaturing is presented. The geometric and finite element data abstractions are given, along with the basic algorithms used. These algorithms deal with near tangencies, coincident edge precision discrepancies, poor intersection curve accuracy, and small geometrical features. Along with this discussion, examples of these types of defeaturing are given.

Keywords. Defeaturing, dirty geometry, object-oriented FEM, CAD import, meshing

1. Introduction

1.1 Importance of work

Though computer aided design (CAD) and computer aided engineering (CAE) can trace their origins to the advent of digital computing, their evolution since have taken them on quite different paths. CAD was originally created to automate the laborious job of 2-D drafting and it remained in the 2-D realm for quite some time. In the CAE world, the finite element analysis (FEA) technique became one of the most popular methods, in which the model was idealized by breaking it down into simple triangular and quadrilateral shapes (know as elements). However, it was realized early that 2-D physical phenomena idealizations were of limited use at best. Applying the same techniques to the tetrahedral and hexahedral shaped elements, full 3-D elements were constructed. The construction of these idealization models, especially in 3-D, becomes tedious and error prone. Specialized software was written to facilitate the construction of these types of models, but the only geometric definition was the nodal locations and element connectivity.

With the development of better solid, parametric, and feature-based modeling techniques, the creation of complex 3-D solid models dominated the next stage of CAD development. The FEA packages had also grown in their sophistication, but they had been created with a bias for creating models that could easily create finite element models. Moreover, modeling was not as high a priority for FEA packages, as was the development of more complex idealization techniques. The solid modeling in these packages lagged behind the more focused CAD packages.

While the creation of solid geometry from both FEA and CAD sound very similar, in fact the created models are quite different. The CAD packages are more interested in creating visually acceptable models that can display a model in its "computer" environment with all the details of the finished product. In addition, CAD models are loaded with details that bring no value to the FEA idealization. Most of the time, these details inhibit the creation of an ideal FEA model by introducing areas that require high element size transitions. Usually, this occurs away

from the FEA operator's area of interest in the model. The FEA package does a great deal of work for little or no gain in the idealization's accuracy. In addition, FEA programs want "clean" geometry and topology in which FEA meshes can easily be generated. Since no real standard form of geometric representation is used by any of the major CAD packages, the transfer of the geometry and topology from one package to another is difficult, if not impossible. The job of reading CAD models into FEA packages is difficult and error prone. Some CAD packages made forays into the analysis realm by creating meshing capabilities inside their modelers. This bypassed the translation problems but generated new ones. Often, CAD packages create FEA idealizations that are too large to be solved by the current technology, that create invalid FEA elements, or that make assumptions that make FEA solvers fail.

However, engineering management sees the creation of two models (one for CAD and one for FEA) as a duplicitous effort and is constantly looking for solutions to this fundamental problem. In addition, since FEA is the end user of the CAD geometry, the inability to transfer the models and operate on them was perceived to be an FEA problem. This paper examines some of the techniques that can be used to handle CAD data in the FEA process.

1.2 Background and previous work

In the August 1997 issue of Engineering Automation Report [5], the readers' survey identified the ability to exchange geometric data between the different vendors' software packages as the most important current issue in the CAD/CAM/CAE community.

Marc Halpern from DH Brown Associates [7] presents the findings of an extensive study that shows that the median time to complete full analyses, including convergence studies, has been reduced 48% from 1991 to 1996. This speedup is credited to advances in mesh generation techniques, the impact of adaptivity, and improvements in CAD-FEA integration. Although this progress has been impressive, Halpern states that the state-of-the-art has still not yet gotten to the point where it can satisfy the requirements of the early design practice. As proof, his study shows that the median time to perform a finite element analysis to verify a design is 4 days and the average time is 7 days. Designers make changes to their designs at a faster pace than that. Therefore, significant advances are still required.

Although dealing with CAD solid models in FEA products is a fairly recent phenomenon, an impressive amount of work has already been done. In this section, a brief synopsis is given for many of the articles and products that include such capabilities, emphasizing those that address one or more of the following:

- Geometric validity checking and repair – especially for geometry import
- Dimensional reduction
- Model idealization or simplification, i.e., defeaturing

Many FEA packages, including ANSYS [8], Firmin and Walsh [6], provide a limited set of manual, bottom-up repair and rebuilding geometry tools. This type of geometry repair involves deleting the volumes, surfaces, and curves near the problem, creating new geometry, and then creating new volumes from the newly modified curves and surfaces.

Dey [3] developed a method that searches for small edges that create small angles, and then collapses all of the small edges that won't invalidate the mesh or reduce the dimensionality of the model. He also presents techniques for collapsing triangular faces or tetrahedral elements that have extremely large dihedral angles. He combines edge and face swaps with these two operators, to increase the method's success rate.

Armstrong [1], Price [10], Jones [9], and Donaghy [4], et al, use the medial object[1] transform of the geometry as a tool to:

- Aid in the decomposition of general solids into sub-regions for mapped meshing,
- Identify slender regions for dimensional reduction, and
- Recognize small features for suppression.

Butlin [2] uses geometric type feature suppression to eliminate small features.

The bulk of this work is at the geometry/topology level. As such it is similar to the feature suppression found in Section 3. However, the authors have found that geometry level feature suppression is not sufficient to guarantee meshing success. This paper extends this work to mesh level suppression that is presented in Section 5. Although this mesh level suppression has some similarities to the dimension reduction methods found above, it does provide a unique tool for the meshing solution.

1.3 Paper overview

This paper presents an object-oriented approach to performing automatic validation, repair, and small feature suppression at the CAD and FEA model level.

In section 2, a brief overview of the class structures is presented for the CAD topological and geometric entities used for the boundary representation. Section 3 presents geometry-based defeaturing, which performs small feature recognition and suppression for specified topological and geometric entities. Section 4 describes the object-oriented C++ classes used to represent the FEA model. Section 5 presents FE model-based defeaturing, where feature suppression is done using the representation of the FEA model described in section 4. Section 6 shows some results, including example meshes using the presented defeaturing scheme. Conclusions are then given in Section 7.

2. Base Classes for CAD Topology and Geometry Representation

2.1 Overview

Although many packages share the same underlying solid modeling kernel, almost every CAD package has it own internal topological and geometric representation. This adds to the complexity of maintaining connections from CAD to FEA. Since, as stated previously, FEA is an end user of CAD models, so a *superset* of the possible topological data structures has to be chosen for the FEA topological representation. If the FEA topological structure chosen does not support all of the CAD's topological representations, then data modification of the CAD model has to also be performed in addition to the data transfer. These modifications can further decrease the reliability of the CAD to FEA model connection.

A topology data structure is an ideal candidate for the data abstractions that object oriented languages such as C++ provide. A well-chosen abstraction is essential since all geometric and topological information must be able to be represented. The model itself can be abstracted as a boundary representation, **Brep**. In addition, each entity (vertex, edge, face, or body) has been abstracted into a base class that was given the name **Cell**. The **Cell** class contains four different types of data (each of which will be discussed in detail in the subsequent sections):

1. Attributes
2. Topological Traversal

[1] The medial object is a medial axis in 2D and a medial surface in 3D. In 2D, it can be thought of as the collection of the centers of the largest circles that can be placed entirely inside the model, without crossing the boundary curves. In 3D, instead of circles, spheres are used.

3. Geometric Query
4. Geometric Data

This scheme matches the paradigm that is used by most CAD systems. In particular, the authors have worked with ACIS, Parasolid (UG), ProEngineer, and Computervision.

2.2 Attributes

Attributes contain information such as the dimensionality and type of the entity and a topological identification number (*topoid*). The *topoid* uniquely identifies the entity in the model. The **BRep** contains a global function that when given a *topoid* can return a pointer to the **Cell** for that entity. Conversely, each **Cell** has a member function that will return its *topoid*. Data base support such as saving and resuming the entity are also categorized as an attribute of the **Cell**.

2.3 Topological Traversal

The ability to move up or down the topological tree in the model is accomplished by another class called **CellUse**. **Cell** has a "has a" relationship to the **CellUse** class. A "has a" relationship describes the situation where one class contains the other, thus the **Cell** class "has a" **CellUse** class as one of its data members. Upward traversal is accomplished by calling the **Cell** member function **use** and using the member function **bound** (see Figure 1) allows for downward traversal in the structure.

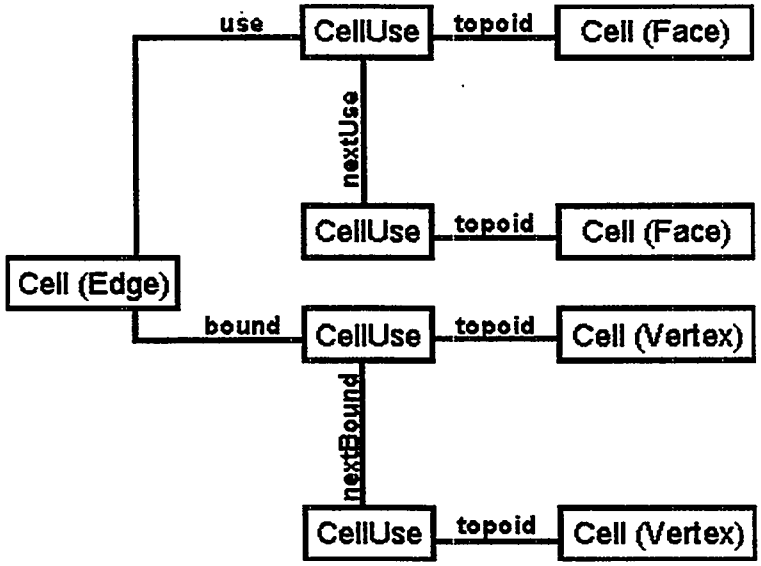


Figure 1.
Upward and downward traversal chart

2.4 Geometric Query

Two types of geometric functions are needed for meshing. The first is computational in nature. This includes the ability to query a geometric entity or evaluate a point on it, such as projecting a point onto the entity, evaluating the normal and tangent information, or finding the geometric and parametric bounds for the entity. **GeomQuery** was the name given to the abstract base class that was developed to abstract these functions. The **Cell** has a "has a" relationship to the **GeomQuery** class. These Geometric Query objects can use internal functions for the above data or link back to the CAD package for these evaluations.

2.5 Geometric Data

The meshing algorithms are written to the **GeomQuery** interface described above. In early implementations, no local curve or surface data was kept. Instead, the CAD packages' functions were used directly to evaluate data. However, it was quickly learned that the CAD package's calculation overhead was prohibitive. The interfaces exposed by the CAD packages are not optimized for cases where many evaluations are repeatedly performed for each geometric entity. The performance for meshing was found to be unacceptable for even the most benign curve and surface types. Therefore, local surface and curve classes have been added and objects from these classes are then cached in memory and optimized specifically for meshing tasks. The virtual base class that was exposed from the boundary representation was called the **GeomData** class. The **Cell** has a "has a" relationship with the **GeomData** class.

3. CAD-Based Feature Suppression

3.1 Overview

Since CAD models are feature rich, an ability to determine which features to include in the FEA idealization without having to go back to the CAD model is beneficial. This type of model defeaturing is found in Butlin [2] and Jones [9]. With this ability to recognize and suppress features, the chances of successfully meshing a CAD model increases. In addition, the resulting meshes usually contain far fewer elements. Another similar approach is presented here.

3.2 Feature Recognition

Listed below are the checks that are made to eliminate features. In order to quantify "smallness", a tolerance, T, is specified:

- C1 -- Cylindrical faces bounded by two circular external edges. The height of the cylinder is smaller than T. (This would represent a boss feature.)
- C2 -- Conic faces that are bounded by two external circular edges. The height of the cone is smaller than T. (This would represent a chamfer feature.)
- C3 -- Toroidal faces that are bounded by three or four circular arcs. The minor radius of the torus is smaller than T. (This would represent fillet features.)
- C4 -- Spherical faces bounded by two, three, or four arcs. The radius of the sphere is smaller than T. (This would represent a fillet feature.)
- C5 -- Faces that are bounded by two circular edges. The distance between the edges is smaller than T. (This represents small holes and conic hole features.)

The elimination of these features from the model is done by simply merging one of the edges into another and removing the face from the topological traversal list of entities for the part. There is also a need to check if the operation can take place at all. For instance, a thin disc might be detected as a small height cylinder. Its removal would collapse the entire model on itself and prevent the analysis model from being performed.

The choice of which edge to keep and which edge to remove can be made by setting up a precedence order for each of the defeaturing operands. Either topological checks and/or geometric checks can be used to determine which edge is to be retained and which is to be discarded.

For each case given above, an algorithm can be created to remove the feature from the model.
For cases C1and C2:

Given: Global defeaturing tolerance, T

Angle tolerance, A
A cylindrical/conic face, F, with a height, H, bounded by two circular edges, E1 and E2.

If H <= T,

Let F1 be the neighboring face attached to E1 and F2 be the neighboring face attached to E2.

Geometric Check:

First, verify that the collapsing of the cylinder would be a valid geometric option. This prevents the collapsing of a face that could degenerate the model as described previously.

1. Evaluate a set of points, P1, along E1 and evaluate normals of points on F1 to create normal set N1.
2. Project points P1 onto E2 to create new set of points P2.
3. Evaluate normals of P2 on F2 to create N2.
4. For each normal pair in N1 and N2, find angle between normals.
5. If angles are within the angle tolerance A, go to Topological Checks. Otherwise, perform no action.

Topological Check:

The topological checks are to determine which edge is to be retained in the model and which edge is to be removed. The checks are implemented such that material is added to the solid model.

1. If E1 is on an interior loop of F1, substitute E1 for E2 in F2 topological structure (i.e. E1 becomes the outer loop for F2).
2. If E2 is on an interior loop of F2, substitute E2 for E1 in F1 topological structure (i.e. E2 becomes the outer loop for F1).
3. Eliminate F from topological structure.

3.3 Small Edge Detection and Removal

Cases C3 to C5 remove edges that are below the tolerance, T, as edges are removed the small faces are also compressed out as a by-product.

Another topological operation that can be used to simplify a CAD model is the removal of edges whose lengths are smaller than the tolerance T. As edges are removed from the model, the attached faces are updated. During updating, face topologies can disappear from the CAD model entirely. Since this is not a closed form procedure like the previously discussed operations, a more comprehensive method for determining the precedence of each edge and face is required. Small edges may also be prevented from being compressed out of the model if they do not pass certain criteria. As previously mentioned, compressing out edges in certain cases can invalidate the model. Take, for example, a thin rectangular plate. If the thickness of the plate is less than the defeaturing tolerance, the compression of the model would make the 3D model degenerate. Using a similar geometry check, this condition can be trapped by using normal comparisons as were discussed in the previous section.

For the cases where the compression of a small length edge can compress an entire face from the model, the following procedure can be used to determine if the face should be compressed -- and if so, which edge should be retained:

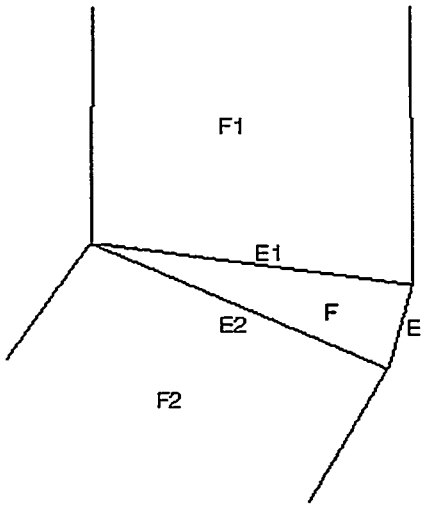


Figure 2.
Suppression of small length edge E and sliver face F.

Given: Global defeaturing tolerance T
A face, F, bounded by three edges E, E1, and E2

If E <= T,

Assume that it is determined that edge E qualifies to be compressed out of the model. Let F1 be the neighboring face attached to E1 and F2 be the neighboring face attached to E2 (see Figure 2).

Geometric Checks:

Verify that the collapse can happen. If so, then determine the edge precedence order. That is, determine which edge (E1 or E2) should be retained and which should be removed.

Validity check:

1. Evaluate a set of points P1 along E1 and evaluate normals of points on F1 to create normal set N1.
2. Project points P1 onto E2 to create new set of points P2.
3. Check the distances between points pairs in P1 and P2. If any distance is greater than tolerance T, do not compress E.

Precedence check:

1. Evaluate normals of P2 on F2 to create normal set N2. At this point, we want to see if one of the faces F1 or F2 is above the other in order to attempt to add material to the model in the defeaturing process.
2. Create a set of local Cartesian coordinate systems C1 at each point in P1 using the normals in N1 as its local z direction.
3. Create a set of local Cartesian coordinate systems C2 at each point in P2 using the normals in N2 as its local z direction.
4. Transform the points in P2 into C1 to create P2 and points in P1 into C2 to create P1.
5. If the local z coordinate for each member in P2 is negative and the local z coordinate for each member of P1 is positive, *retain edge E1.*
6. If the local z coordinate for each member in P1 is negative and the local z coordinate for each member of P2 is positive, *retain edge E2.*
7. Else,
8. If E1's length > E2's length *retain E1.*
9. If E2's length > E1's length *retain E2.*

10. Else, *retain E1*.
At this point, F is removed from the model and if E1 is retained, the edge list for F2 is updated. If E2 is retained the edge list for F1 is updated.

4. Finite Element Data Abstraction

4.1 Overview

As with the CAD boundary representation, finite element data naturally lends itself to object oriented data abstraction. The classical way of representing a finite element is to break the representation into two entities. The first entity is the node that has the global 3D location. The node represents the geometry of the finite element representation. The next entity, which represents the topology of a finite element, is the element. An element contains a list of nodes that make up its shape. A quadratic triangular element will have a list of six nodes: one node for each corner and one node for the midpoint of each side. The design used in this paper is a mix of the classical finite element topological entity in conjunction with the CAD boundary data. Instead of viewing the finite element as a list of nodes, the element is a collection of topological entities that the classical finite elements can be derived from. The topological entities relate the node back to its defining geometry. This was felt to be a more generic and natural representation for the problem. The implementation in this paper uses tri/tet elements; however, the concepts could be extended to quad/hex elements.

4.2 Geometric Representation

The class that was created to represent all geometric information in the finite element structure is Mesh Boundary Point (**MBPoint**). The **MBPoint** contains more than just the 3D location in space of the node. It is also a container for all of the geometric information that is known about that point. Another class called **Geompt** was created to store information that is specific to one CAD entity. The **MBPoint** has a "has a" relationship with a link list of **Geompt** objects. The **Geompt** contains the *topoid* of the CAD entity, the node's parametric location for that entity, and other attributes. Some of these attributes are the flags that tell if the node actually lies on the entity within a tolerance, whether the entity's parametric representation is current, and whether this entity is the prime entity representation for this **MBPoint.** For instance, if a node lies on a vertex of a block, it would have one **Geompt** of the vertex (which would be marked as it's prime entity), six **Geompt** objects from the three edges and three faces that are attached to the vertex.

The member functions of the **MBPoint** class associate the operands of the CAD's **Brep GeomQuery** class to the finite element node. The operations of evaluating, getting normal and tangent information, and projecting on a specific entity are all member functions of the **MBPoint** class. Other functions such a moving a point from one location to another are encapsulated in the **MBPoint** class. This function takes care of managing the **Geompt** list and, if needed, updating the parametric locations.

4.3 Topological Representation

With the geometric information removed from the finite elements, the problem of representing this in a class hierarchy becomes much easier. If we ask the question, "What are the topological entities that represent the finite element class?", a very natural representation falls out. First, the nodes that represent the vertices of the finite element are abstracted into a topological class, which we called **MBE0** (Mesh Boundary Elements 0-dimension). The edges of the finite element abstract into a class **MBE1**, the faces into **MBE2**, and the volume elements into **MBE3**. The **MBE0** has a "has a" relationship with the **MBPoint** class. Most geometric operations are accessed through the interface of the **MBE0** into the **MBPoint** class. Since our finite element specification called for quadratic elements, the **MBE1** class also has a "has a" relationship with the **MBPoint** class. The coordinate retrieved from this **MBPoint** represents the mid side node in the quadratic finite element.

The **MBE1** class also contains pointers to the two **MBE0**s that bound it. The **MBE2** class contains pointers to the **MBE0**s and **MBE1**s that bound it and the **MBE3** class contains pointers to the **MBE0** and **MBE2** objects that bound it.

There are also upward and downward traversal methods for each class. These traversals are not limited to one level up or down, but have been implemented, for instance, to allow the traversal of all **MBE3**s attached to a specific **MBE0** object.

5. FE Model-Based Defeaturing

5.1 Overview

As previously discussed, defeaturing of the CAD model can significantly reduce the number of elements, increase the robustness of meshing, and decrease the time necessary to mesh the model. However, the techniques discussed in Section 3 do not cover all CAD topological and geometric problems. Recognition and suppression of features that could cause meshing problems becomes difficult at the **BRep** level beyond those discussed in Section 3. However, it became apparent that to achieve a high level of meshing success, more forms of defeaturing were needed.

After doing analysis on CAD models that failed to mesh, several causes were identified for meshing failures. Most of the meshing failures occurred while meshing the faces of the CAD models. The faces that failed had one of several attributes that seem to cause the meshing problems. These included extremely small length edges as compared to the entire face, tangencies between two edges at a shared vertex, and edges that were coincident (within a tolerance). Though some of these can be identified at the **BRep** level, there are no easy topological changes that can be made to modify the model.

5.2 MBE Level Defeaturing

Instead of working at the **BRep** level, MBE level defeaturing procedures work with the data generated during the meshing process. This process first meshes the edges of the model. This is followed by meshing the faces and then for volume meshes, the meshing of the interior. After the first stage is complete and **MBE0** and **MBE1** objects have been created on the edges and vertices, the following algorithm was implemented to scan these objects and collapse near coincident or tangent edges into one another. The basic algorithm looks for nodes that are close to edges that do not contain the node. This will happen when the model has a slight rise from one face to another, as shown in Figure 11. It can also happen when a model has a face with edges that are nearly coincident or tangent, as shown in Figure 13.

Given: Global defeaturing tolerance, T
Length ratio tolerance, R
Angle tolerance, A
Average length of edges, L
Face normal tolerance, FN
Merging tolerance, MT

1. Let Node1 belong to the set of **MBE0** objects. Let *MBN1* be a set of **MBE1s**, such that E belongs to *MBN1* if the distance(Node1,E) < L. Let *SMBN1* be a subset of *MBN1*, where E belongs to *SMBN1* if E contains Node1 - Figure 3 presents such a case where Node1 has four edges that are close to it. Two of the edges are in *SMNB1* and the other two are not.
2. Find an edge, Edge1, in *MBN1* such that Edge1 does not contain Node1 and Edge1 shares a common face with Node1. If there are none, get another Node1 from the set of **MBE0** objects and start with Step 1.

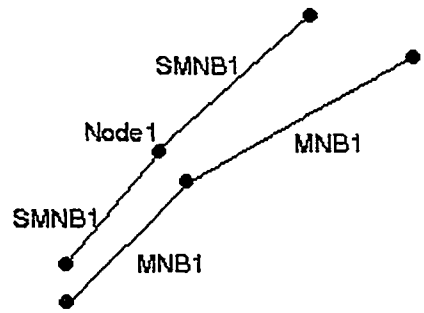


Figure 3.
Node1 is in the proximity of edges in the set MNB1.

3. First check for a node that is close to other edges. This indicates a rise or coincident edge condition. Let Node2 be the projection of Node1 onto Edge1, and let D=distance(Node1,Node2) – see Figure 4. –

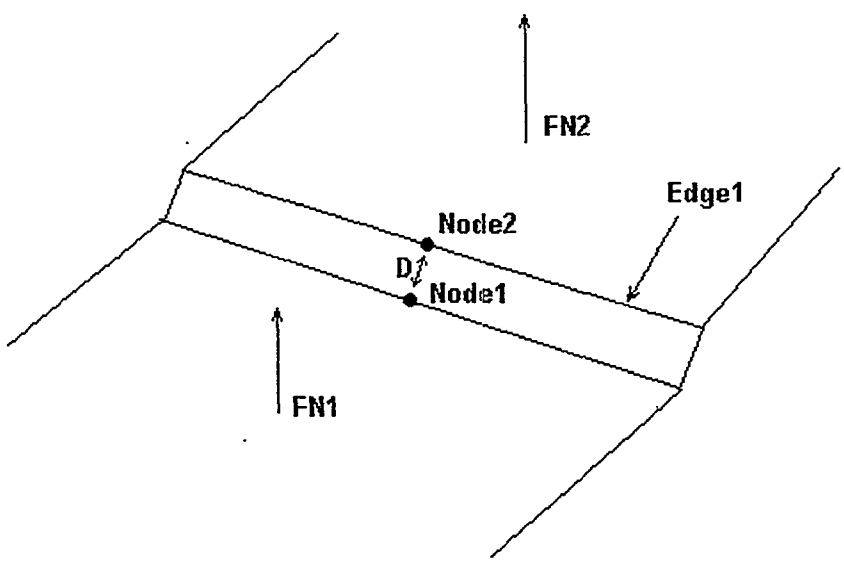


Figure 4.
Node2 is the projection of Node1 on Edge1. FN1 and FN2 are the face normal sets.

4. Let SL = maximum (length of edges belonging to *SMBN1*).
   Let AL = maximum (SL, Length of Edge1).
   If D < T, or the ratio of D /AL > R, then continue with step 7.
5. Check for a tangency condition. If EN has a **MBE0** that is shared by Edge1, let A1=angle(Edge1,EN). See Figure 5.
6. If the angle A1 < A, then continue with step 7. Else, go to step 2 - see Figure 5.
7. Now Node1 is a candidate to be merged out of the model along with it's connected edges. Create another candidate at Node2. First check that Node2 is not close to one of the endpoints of Edge1. Let EP1 and EP2 be the endpoints for Edge1. If distance(EP1,Node2) < MT, then set Node2=EP1. Else if distance (EP2,Node2) < MT, then set Node2=EP2. Else split Edge1 at Node2 and create a new **MBE0** at this point.
8. Edge1 was chosen such that it and Node1 have a face in common. Look at the faces that are not in common. If these normals are in the same direction, then the node can be merged out. If one has a tangent edge, a nearly

coincident edge, or a slight rise from one face to another, this is exactly the case. However if one has a small cut feature in the model, this is not the case.

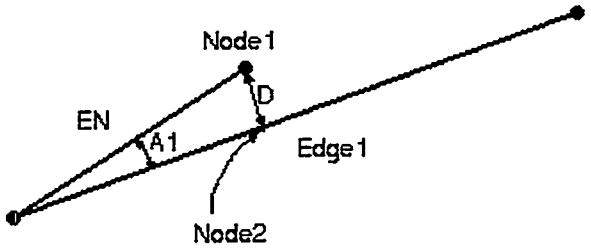


Figure 5.
Edge EN contains node Node1 and shares a node with Edge1.
A1 is the angle between edges EN and Edge1

9. Let FN1 be the face normals from Node1 and FN2 be the face normals from Node2. If angle(FN1,FN2) > FN, another candidate node is retrieved and the procedure starts again at step 1. (This results in a split of Edge1, which will result in better element quality, but no merging is possible.) See Figure 6.

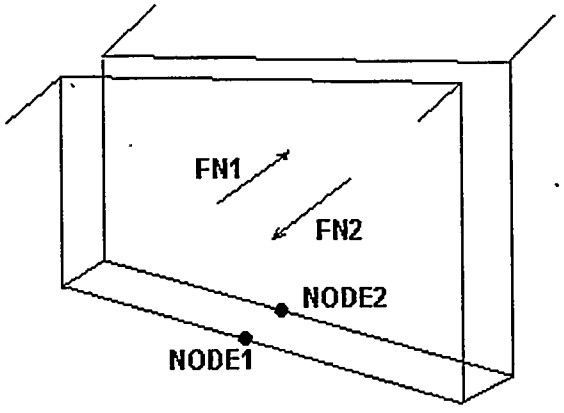


Figure 6.
This situation shows an example where Node1 and Node2 cannot be merged.

10. Node1 and Node2 can now be merged. It must now be determined which of the **MBE0** objects is to be retained in the model and which one is to be deleted. This done be assigning an order of precedence to each **MBE0**, the **MBE0** with the highest order of precedence is retained and the other **MBE0** removed. The order of precedence is determined by first checking the **BRep**'s topological entities that attach to each **MBE0**. If one **MBE0** is attached to an entity of a lower dimensionality, then it will be retained. If both **MBE0**s **BRep**'s lowest dimension entity are the same go to step 11.
11. The next operation in determining the order of precedence is to create a set of planes at Node1, PL1, from the Node1's location and the face normals, FN1. Then Node2's location is then compared against these planes, PL1, to determine if Node2 is above or below each of the planes. The procedure is then repeated for Node2. If either **MBE0** lies below each of the planes, this it is chosen to be merged out of the model. This is consistent with the goal to add material when defeaturing the model. If neither **MBE0** fulfills these criteria, then go to step 12. See Figure 7.

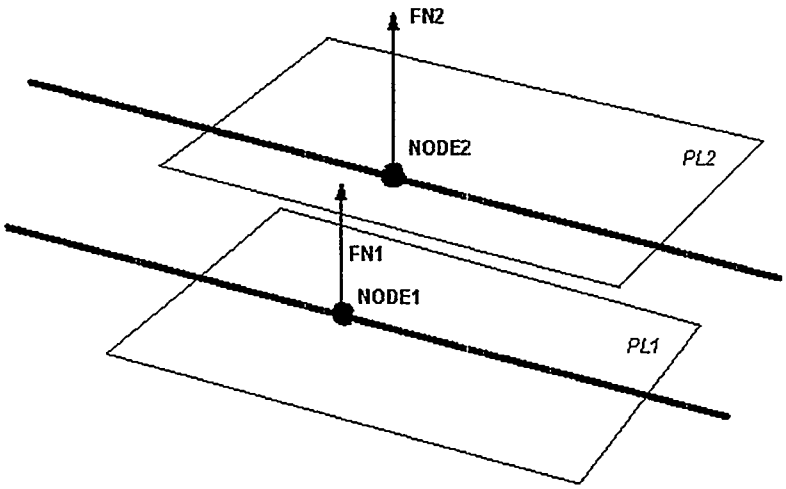


Figure 7
Face normal sets FN1 and FN2 when coupled with nodal positions of Node1 and Node2 will define a set of planes PL1 and PL2.

12. The next test in the order or precedence is to see which **MBE0** would move the least distance if merged. Node1's location is projected onto all of Node2's faces and the maximum distance that Node1's location is from any of Node2's faces is found. Then the procedure is done for Node2. The node whose maximum distance is smallest is retained and the other one is removed.
13. Merge the node with the lowest order of precedence into the other node. Using the situation portrayed in Figure 4, it can be shown that Node2 would have a higher order of precedence than Node1 (using the procedure in step 11 and shown in Figure 7). After merging Node1 into Node2, material would be "added" to the model. See Figure 8.

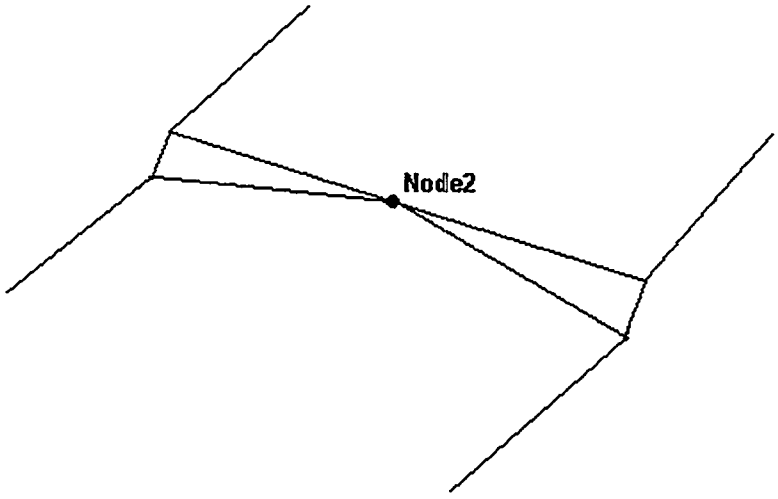


Figure 8
After merging Node1 into Node2 the resulting boundary would appear like this figure. Further defeaturing could take place because of the tangencies and/or the small edges surrounding Node2.

6. Results and Examples

6.1 Example of topological defeaturing of the CAD model.

A parametric model was constructed of a disk, shown below in Figure 9. The four holes in the disk, the fillet, and the recess where defined with parameters. The parameters were changed such that these features became small compared to the size of the model and highlighted. This model was then analyzed by fixing the inner cylinder and applying a moment to the outer cylinder. Two analyses were performed, the first without defeaturing and second with the highlighted features suppressed. The meshing time and the number of elements for each model are presented in Table 1. The meshes for each analysis are shown in Figure 10. It can be seen that the suppression of the small features dramatically reduces the number of elements and time to mesh. This model is contrived to illustrate this point and is not meant to claim that the use of engineering judgement should not be used on whether feature suppression is important to the analysis.

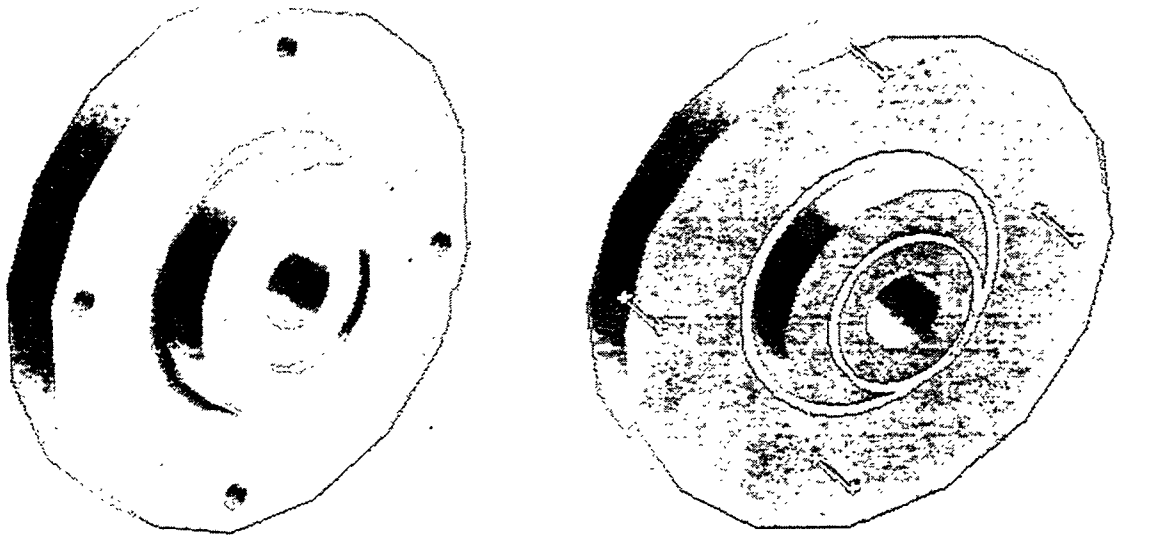


Figure 9.
Parametric disk model. On the left is the original model. The figure on the right shows the model after parametric changes have produced small features to be removed from the topological model.

	Number Of Elements	CPU Time (Sec.)
Model Without Defeaturing	14,743	60
Model With Defeaturing	1,038	11

Table 1.

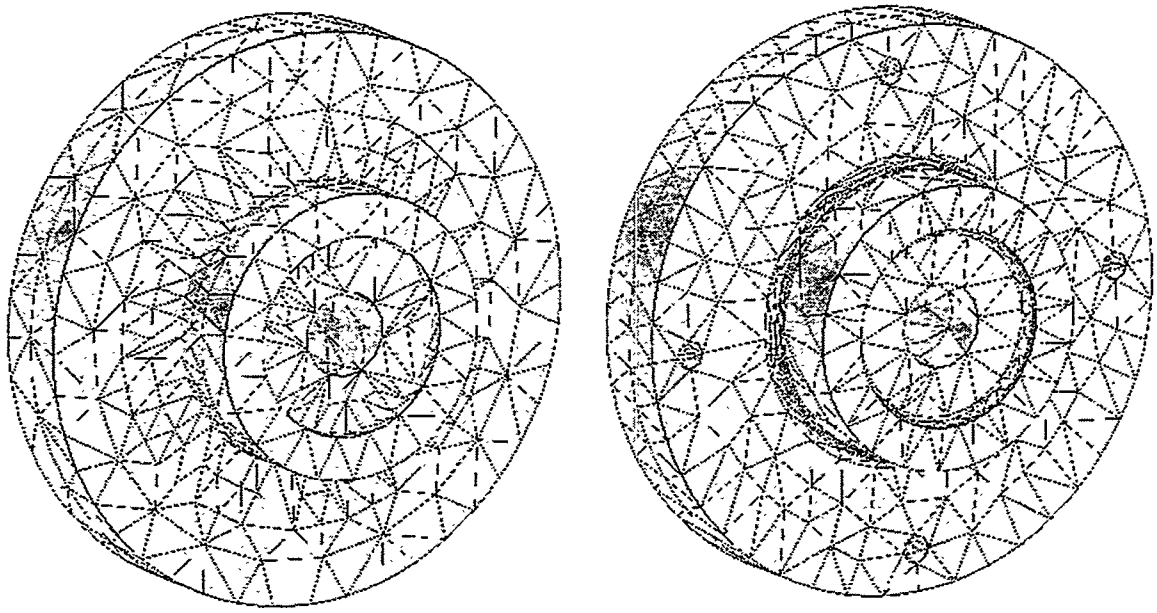


Figure 10.
Finite element model with (model on the left) and without (model on the right) defeaturing.

6.2 Finite element defeaturing

Two examples are given showing faces that have degenerate exterior loops. In the first model, the attachment of the upper block is off by a small distance and creates a degenerate edge along the top. (See Figure 11, notice what appears to be a single line across the top of the block, but is, in reality, a coincident edge). Figure 12 shows the finite element mesh along that top section after MBE1 defeaturing has been applied.

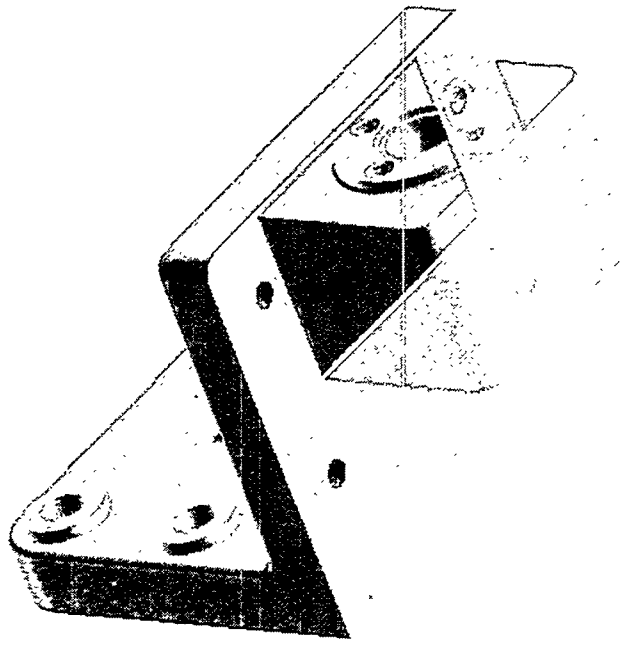


Figure11.
Highlighting of a face with a degenerate edge.

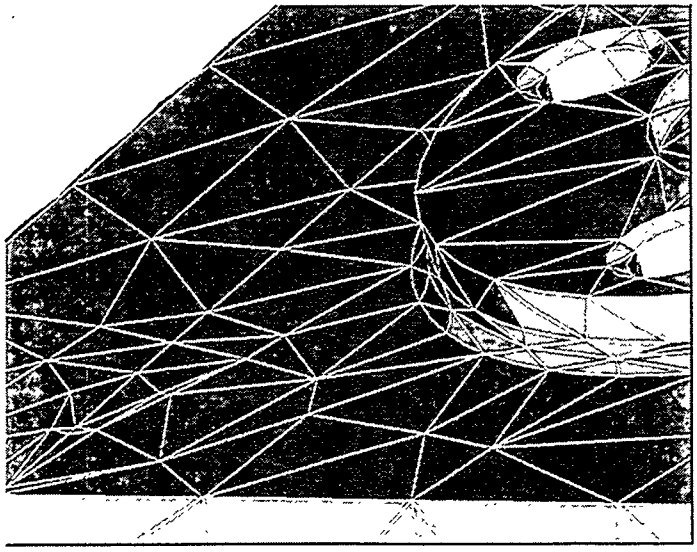


Figure 12.
Zoom of finite element mesh showing the area of the coincident edges.

Another model of a hack saw shows a highlighted face that has a degenerate face loop (see Figure13) and a close-up of the elements where the MBE1 defeaturing has taken place (see Figure 14).

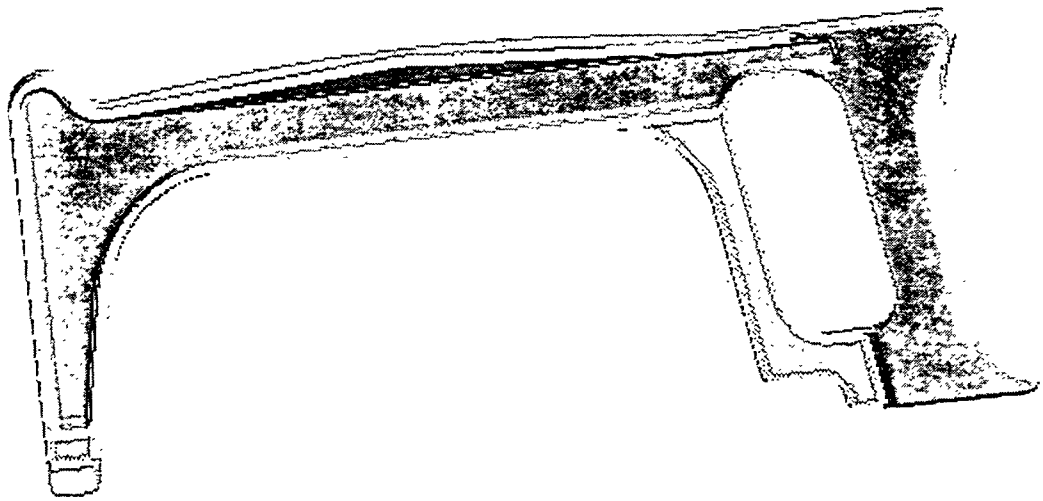


Figure 13.
Hack saw model showing degenerate face loops.

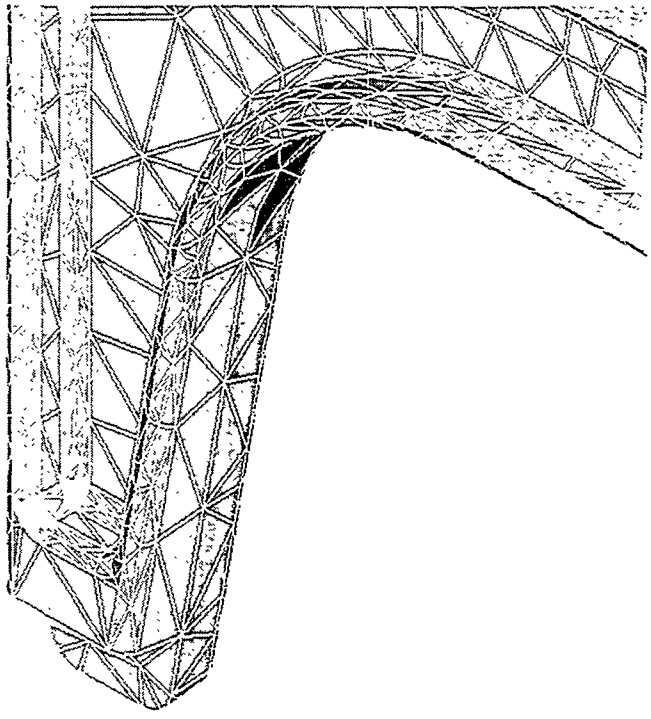


Figure 14.
Zoom of finite element mesh.

7. Conclusion

This paper presents a framework for automatically meshing real world 3D CAD models with tetrahedra. The emphasis is on dealing with the problems inherent in meshing CAD models. Processes are described which can recognize undesirable features in CAD models that can make meshing fail, give poorly shaped elements, or give too many elements. In addition, two general types of defeaturing are presented – geometry-based defeaturing and FE model-based defeaturing. These forms of defeaturing are shown to reduce model size and increase mesher robustness.

This paper also illustrates the advantages of using a well thought out data abstraction design to represent not only the CAD geometry and topology but the finite element model as well. A good foundation in object-oriented design allows the programmer of the defeaturing algorithm to focus on the algorithm and not on the data management of the underlying structures.

While the defeaturing presented in this paper is an automatic procedure, it is a tool to be used by an informed engineer. Good engineering judgment still needs to be exercised. As with any analysis, the effects of the feature suppression should be independently verified.

Acknowledgements

The authors would also like to thank ANSYS, Inc. for its support of the research, development and publication of the work and Chris Brunetti for his help in creating the examples used in this paper.

References

[1] Armstrong, C. G., D. J. Robinson, R. M. McKeag, T. S. Li, S. J. Bridgett, R. J. Donaghy and C. A. McGleenan, "Medials for Meshing and More", *Proceedings of 4th International Meshing Roundtable*, Sandia National Laboratories, p.277-288, October 1995.

[2] Butlin, G. and C. Stops, "CAD Data Repair", *Proceedings of 5th International Meshing Roundtable*, Sandia National Laboratories, pp.7-12, October 1996.

[3] Dey, S., M. S. Shephard, and M. K. Georges, "Elimination of the Adverse Effects of Small Model Features by the Local Modification of Automatically Generated Meshes," *Engineering with Computers*, Vol. 13, p. 134-152, 1997.

[4] Donaghy, R. J., W. McCune, S. J. Bridgett, C. G. Armstrong, D. J. Robinson and R.M. McKeag, "Dimensional Reduction of Analysis Models", *Proceedings of 5th International Meshing Roundtable*, Sandia National Laboratories, pp.307-320, October 1996.

[5] *Engineering Automation Report*, p. 10, August 1997.

[6] Firmin, T., and J. Walsh, "The Challenge of Transferring Solid Model Data to ANSYS," *Proceedings of the Eighth International ANSYS Conference*, Pittsburgh, Pennsylvania, 1998.

[7] Halpern, M., "Industrial Requirements and Practices in Finite Element Meshing: A Survey of Trends," *Proceedings of 6th International Meshing Roundtable*, Sandia National Laboratories, pp.399-411, October 1997.

[8] Importing Solid Models, ANSYS Modeling and Meshing Guide, Release 5.4, 000862, 2nd Edition, SAS IP, Inc., Chapter 6, September 1997.

[9] Jones, M.R., M. A. Price and G. Butlin, "Geometry Management Support for Auto-Meshing", *Proceedings of 4th International Meshing Roundtable*, Sandia National Laboratories, pp.153-164, October 1995.

[10] Price, M., C. Stops, and G. Butlin, "A Medial Object Toolkit for Meshing and Other Applications", *Proceedings of 4th International Meshing Roundtable*, Sandia National Laboratories, p.219-229, October 1995.

Panel Discussion

Notes:

Notes:

Notes:

Index of Authors & Co-Authors

Index by Affiliation

Index by Affiliation, con't

Distribution:

MS9018 Central Technical Files, 8940-2
MS0899 Technical Library, 4916 (5 copies)
MS0619 Review & Approval Desk, 12690 (2 copies for DOE/OSTI)

MS0441 9226 (250 copies)
To be distributed at the
7th International Meshing Roundtable '98
Dearborn, Michigan
October 26-28, 1998

For additional copies:
Sandia National Laboratories
PO Box 5800, MS0441
Albuquerque, NM 87185-0441