Choosing corners of rectangles for mapped meshing

Scott A. Mitchell[1]

Parallel Computing Sciences Dept., Sandia National Laboratories, Albuquerque, NM 87185

Abstract. Consider mapping a regular i x j *quadrilateral mesh of a rectangle onto a surface. The quality of the mapped mesh of the surface depends heavily on which vertices of the surface correspond to* corners *of the rectangle. Our problem is, given an* n*-sided surface, chose as* corners *four vertices such that the surface resembles a rectangle with corners at those vertices. Note that* n *could be quite large, and the length and width of the rectangle,* i *and* j *are not prespecified. In general, there is either a goal number or a prescribed number of mesh edges for each bounding curve of the surface. The goals affect the quality of the mesh, and the prescribed edges may make finding a feasible set of corners difficult.*

The algorithm need only work for surfaces that are roughly rectangular, particularly those without large reflex angles, as otherwise an unstructured meshing algorithm is used instead. We report on the theory and implementation of algorithms for this problem.

We also give an overview of a solution to a related problem called interval assignment: Given a complex of surfaces sharing curves, globally assign the number of mesh edges or intervals for each curve such that it is possible to mesh each surface according to its prescribed quadrilateral meshing algorithm, and assigned and user-prescribed boundary mesh edges and corners. We also note a practical, constructive technique that relies on interval assignment that can generate a quadrilateral mesh of a complex of surfaces such that a compatible hexahedral mesh of the enclosed volume exists.

keywords. quadrilateral, hexahedral, mesh, rectangle-primitives, corner picking, interval assignment, existence

1. Introduction

Currently, practical mesh generation can be a very time consuming process. An experienced mesh generation software user may spend six months or more meshing a large industrial model. The user must supply goals (*soft-sets*) and constraints (*hard-sets*) on the sizes of mesh elements throughout the model, either by specifying a sizing function or actual values on various vertices, curves, surfaces, and volumes. The best software packages provide a toolbox of algorithms that have complex trade-offs in terms of their automation, general applicability to complex geometries, quality of the produced mesh, and speed and memory requirements. Often the user must decide which algorithm to use for many subsets of the model, the order in which subsets are meshed, and additional parameters for each subset. Intelligent automation of these steps greatly speeds the overall mesh generation process. We report on an improved capability that automatically removes some order dependence and automatically specifies some parameters. (The automation of algorithm selection is also an important problem and is being researched.)

In particular, in the CUBIT[1] quadrilateral and hexahedral mesh generation toolkit, a common paradigm is an advancing front. This requires us to first mesh the curves, then the surfaces, and finally the interior of a volume. The user either specifies a *soft-set* goal or a fixed *hard-set* number of mesh edges (called *intervals*) for each curve. However, different quadrilateral surface meshing algorithms require a certain relationship between the number of intervals on each bounding curve. For meshing a surface with a *mapping algorithm*, these constraints depend on some additional parameters, namely which vertices are chosen as the *corners* of the mapping primitive. Typically a curve is

1*samitch@sandia.gov.* This work was supported by the U.S. Department of Energy under contract DE-AC04-76DO00789, by the Applied Mathematical Sciences program, U.S. Department of Energy Research.

DISCLAIMER

contained in two surfaces, and its intervals are constrained in some way by both of them. If surfaces are meshed one by one, then often we paint ourselves into a corner and are left with a collection of surfaces that are unmeshable because their bounding curves are incompatibly meshed. Instead, we automatically and locally chose the corner parameters, then globally find the number of mesh edges (intervals) for each curve such that all surfaces can be meshed with their chosen algorithm and corners. These problems are called *corner picking* and *interval assignment*, respectively. Note that for a given choice of algorithms, corners, and hard-set intervals, interval assignment may be impossible (e.g. see Figure 1). Simultaneously choosing corners and intervals to allow surface meshing is an NP-hard problem[2].

We also note that if we cut surfaces into topological disks, and enforce that every curve is even, then any quadrilateral mesh of the surfaces that is compatible with that curve mesh admits a compatible hexahedral mesh of the enclosed volume.

Corner picking is a highly geometric problem, but is non-Euclidean in that the length of a curve matters less than its user-specified number of intervals, and whether these intervals are soft- or hard-set. The interval assignment problem comes from geometric data, but is essentially a combinatorial problem, usually solved by linear programming[3] or more recently by network flow algorithms[2].

2. Corner Picking

"Map meshing a surface" produces some (hopefully isomeric) map from an abstract structured quadrilateral mesh of an n-gon *primitive* to a surface. Typical n-gons include triangles, rectangles and pentagons. Mapping is valued as the fastest and least memory intensive surface meshing algorithm, but it is only applicable to special geometries: The quality of the surface's mesh depends on how close the map is to a constant function. The quality of the map depends heavily on the interior angles of the vertices of the surface which are identified with corners of the abstract n-gon, the relative number of intervals on sides, and whether any reflex interior-angles exist. Here for brevity we consider only the rectangle-primitive, which is a regular i x j mesh (see Figure 7 left for a 5x5 example). Our technique has been applied to triangle primitives and could be applied to other n-gons as well.

Often the surface is composed of four curves, and the corners of the rectangle primitive are uniquely determined by the geometry. However, we can rectangle-primitive map a surface that resembles a rectangle in a fuzzy sense, and is bounded by more than four curves. Some of the rectangle sides will be composed of a composite of several curves.

The interval assignment constraints are that opposite sides have exactly equal intervals. The intervals on hard-set curves are fixed, but the intervals on soft-set curves can be adjusted up or down to satisfy these constraints. In the case of some curves having hard-set intervals, the corner choice may also determine the feasibility of mapping the surface, as in Figure 1. There is even a global problem that is NP-complete[2] that we don't address here: Some corner choices may lead to the global interval assignment problem being infeasible, as in Figure 1 right.

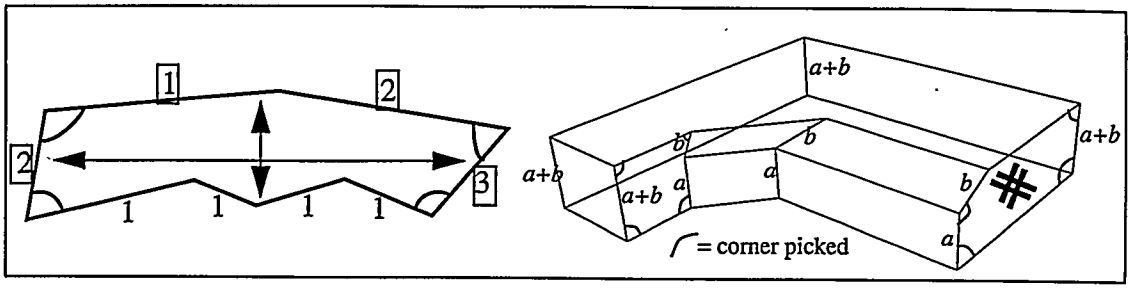

Figure 1. Left, the hard-set (boxed) intervals and corner choice leads to an infeasible interval assignment problem. First, the opposite right and left sides are hard-set with unequal values. More subtly, the top and bottom sides are also infeasible: Since each curve must have at least one interval the bottom side must have at least four intervals but the top side is hard set to three. Right, local corner picking makes global interval assignment impossible on this real-world geometry. We get a system of equations that reduces to $a+b = a$, whose only solution has $b=0$, but a curve must have at least one interval!

A previous method used in CUBIT is to pick as corners the vertices with smallest interior angle. Obviously this can lead to a poor mesh. Arkin et. al[4] presents a polygon-matching algorithm that defines a metric for the shape-distance between two polygons in terms of an integral involving the two polygons' *turning-functions*. We can't immediately use this algorithm because one of our polygons is a rectangle of indeterminate height and width, and because interval assignment imposes different constraints and goals on which curves are on sides opposite each other.

However, our method of choice is a heuristic loosely based on the turning-function of Arkin et. al[4]. The heuristic has a provable running time of $O(n^2)$, where n is the number of curves. It usually gives a good solution, but has trouble finding any solution in the case that many curves are hard-set and there are few feasible choices of corners. If the heuristic fails, we fall back on a set of algorithms that provably find a feasible set of corners, if one exists, in time $O(n^2 \log n)$. A post-processing heuristic shifts corners incrementally, singly and in pairs, keeping feasible at all times, searching for a local optima.

2.1 Heuristic Corner Picking

The algorithm makes two $O(n^2)$ passes. The first pass determines opposite corners 1 and 3, the second pass determines corners 2 and 4. In the first pass, all pairs of vertices (a,b) are considered, and the pair with the best (smallest) *quality function* value is chosen. The quality function is a weighted sum of the corner_angle function, the turn_angle function, and the interval_ratio function. The corner_angle function depends on the interior angles at the corners (ideally $\pi/2$), the turn_angle function on the amount the vector tangent to the bounding curves turns strictly between successive corners (ideally $\pi/2$, to leave room for another corner), and the interval_ratio function depends on the ratio of intervals between sides (ideally 1). The parameters for corners $(a,b+1)$ can be determined in $O(1)$ time given their values for corners (a,b). These functions are super-linear, the goal being to not vary too greatly from a rectangle in any one sense.

See Figure 2 for the effect of the turn_angle function. In addition, for certain pathological cases a possible improvement is to consider the maximum deviation of the tangent vector, rather than just comparing vectors adjacent to the corners. In addition to the interval ratio, a pair of vertices is excluded if the hard-set intervals preclude any possible mapping with those vertices as opposite corners. The second pass chooses corner 2 between 1 and 3, and

corner 4 between 3 and 1. It is based on a similar quality function that considers the interior angles at corners 2 and 4 (ideally $\pi/2$), the ratio of intervals between opposite sides (ideally 1, for both pairs of opposite sides), and the amount of turning between successive corners (ideally 0).

If there are very few curves, say less than nine, then we forego the above heuristic and just exhaustively try all (< 210) combinations of four corners. We use similar functions to those described above, picking the combination with the best quality function. This quality function takes into account the fact that if a side must increase by some number of intervals to match the opposite side, if many curves are hard-set then the change must be distributed among the remaining soft-set curves. For illustration, the examples in section 2.3 and most other examples don't use the exhaustive search.

After intervals are assigned and curves are meshed, a similar heuristic is used to decide where to place the corners for the actual meshing. There is additional freedom, as any mesh vertex can be a corner, not just a curve vertex. Usually the chosen corners are the same, but not always. For example, the circle in Figure 2 has only one bounding curve and no corners to chose before curve meshing, and the union of circles in Figure 2 has only bad reflex corners before curve meshing. Picking corners for surfaces with meshed curves takes time $O(m)$, where m is the number of mesh edges, since in each pass only the $m/2$ pairs of mesh vertices that exactly divide the surface need be considered. (One could try all possible quadruples of corners in time $O(m^2)$.)

```
turn_angle_function ( turn_angle, target_turn_angle )
    return | turn_angle - target_turn_angle |^1.7 / 2

corner_angle_function ( corner_angle, target_corner_angle )
    ratio = ( corner_angle - target_corner_angle ) / target_corner_angle
    smoothly blow up ratio as corner_angle goes from 3π/4 to π
    return 2 * ratio^1.7

interval_ratio_function ( side_intervals, opposite_side_intervals )
    ratio = side_intervals / opposite_side_intervals
    if (ratio < 1) then ratio = 1/ratio
    return ratio - 1
```

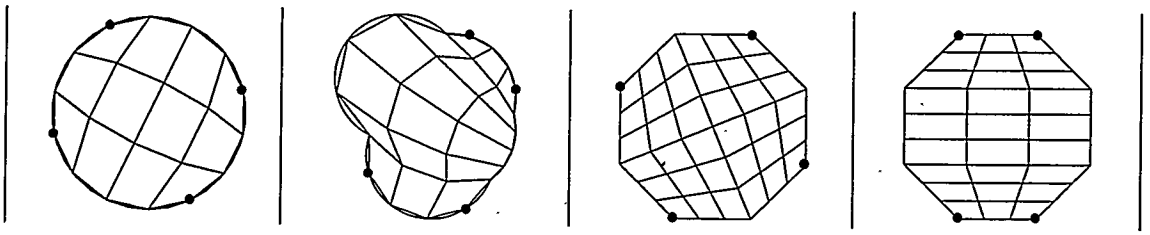

Figure 2. The left circle has no corners, and second from the left has only bad corners, until after intervals are assigned. Right, without the turning function, the corner choices for both octagons have the same quality function value, as opposite-side intervals are equal and corner angles are all $3\pi/4$. But with the turning function the more regular mesh is preferred.

2.2 Provable Corner Picking

Occasionally when there are many hard-set curves, the above heuristic fails to find a set of corners that admits a feasible solution to the interval assignment problem. In that case we can provably find a feasible a feasible set of corners, if any exist, in time $O(n^2 \log n)$, where n is the number of curves bounding the surface. There are four cases, depending on the number of soft-set curves. We then apply an incremental shifting heuristic to get a good quality set of corners. Figure 4 and Figure 5 illustrate some meshing results.

Case three or more soft-set curves. We can find a feasible solution (one always exists) in $O(n)$ time. We put a soft-set curve by itself on a side, say opposite a side that's all hard-set, and the remaining two or more soft-set curves on sides opposite each other.

Case two soft-set curves. If there is a solution with both soft-set curves on the same side or on opposite sides, then there would also be a solution if one of the soft-set curves were hard-set to one interval. So we consider this a degenerate form of the one soft-set curve case below, and here only consider solutions with soft-set curves on adjacent sides.

Finding an adjacent-side solution if one exists takes time $O(n^2)$. If there is a solution with soft-set curves *S0* and *S1* on adjacent sides (sharing corner 2) then sliding corners 1 and 3 towards *S0* and *S1* is also a solution. Thus we reduce to the following form: *S0* is immediately after corner 1, and *S1* is immediately before corner 3, and the sum of the hard-set intervals from corner 3 to 1 is greater than the hard-set sum from 1 to 3. See Figure 3.

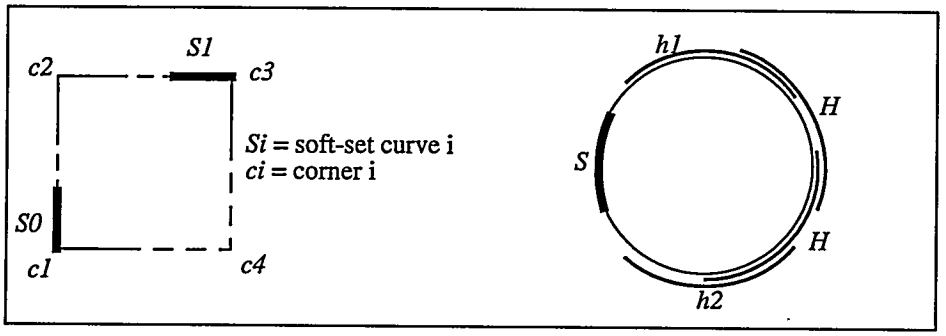

Figure 3. Left, the canonical form of a feasible solution with two soft-set curves *S0* and *S1* on adjacent sides. Right, the canonical form of a feasible solution with one soft-set curve. Right, the canonical form of a feasible solution with one soft set curve *S*. The *H*s are subsequences of hard-set edges, with interval count *A*. For a given *A*, if any pair of *H* form a feasible choice of corners, then the pair *h1* and *h2* closest to *S* does.

Finding corners 1 and 3 satisfying the hard-set sum constraint takes linear time, and checking all possible pairs of corners *c2* and *c4* for feasibility takes only $O(n^2)$ time if we incrementally update parameters.

Case one soft-set curve. In this case any feasible corner solution has two opposite hard-set sides with exactly the same interval sum *A*. The remaining side *s1* containing the soft-set curve *S* has a smaller hard-set interval sum than its opposite side *s2*.

Finding a solution in time $O(n^2 \log n)$ time is somewhat tricky. Among all candidate sides H with interval sum A, there is a side $h1$ closest to S in the clockwise direction, and a side $h2$ closest in the counter-clockwise direction. See Figure 3 right. If there is any pair of candidate sides that do not overlap, then $h1$ and $h2$ do not overlap. In addition, among all candidate sides H, choosing $h1$ and $h2$ as opposite sides gives $s1$ with the smallest hard-set interval sum and $s2$ with the largest hard-set interval sum. Hence, for a given A, finding $h1$ and $h2$ is sufficient to determine if there is a solution. In practice we find $h1$ for various A and as we encounter other H with the same A, we stop when we have a feasible solution.

In particular, our algorithm is as follows: We consider all pairs of corners not straddling S as candidate sides H. We first consider H with smaller first corner. We keep a list of these H sorted by hard-set interval sum A. If a second H is found with a given A, then using stored data we can in constant time check if the two candidates form a feasible solution. If they do, we're done. If they don't, then we retain the first candidate (its $h1$ for A) and continue.

Case no soft-set curves. There is a $O(n \log n)$ algorithm in this case. Any two pairs of corners that exactly divide the interval sum in half are a feasible solution. All such pairs can be found by advancing opposite corners in lock-step manner. However, we have not implemented this since the heuristic (section 2.1) would have found a feasible solution if one existed, albeit in $O(n^2)$ time.

2.3 Corner Picking Examples

The first set of examples, Figure 4 through Figure 5, illustrates heuristic corner picking, and each of the cases of the provable corner picking.

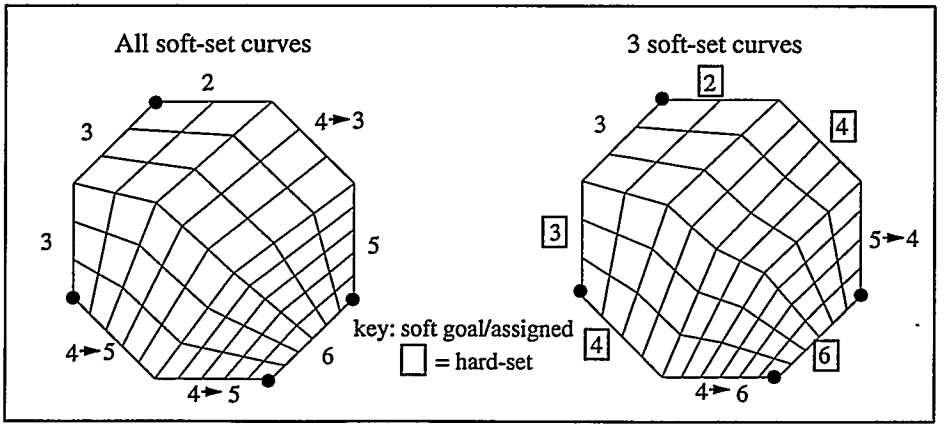

Figure 4. Left, heuristic corner picking works well on examples without hard-set curves. Right, the heuristic and the exhaustive-search were turned off so that the provable algorithm for the "only 3 soft" case plus the incremental shifting heuristic was run.

The second set of examples, Figure 6 and Figure 7, shows the results of corner picking and interval assignment on a sampling of surfaces.

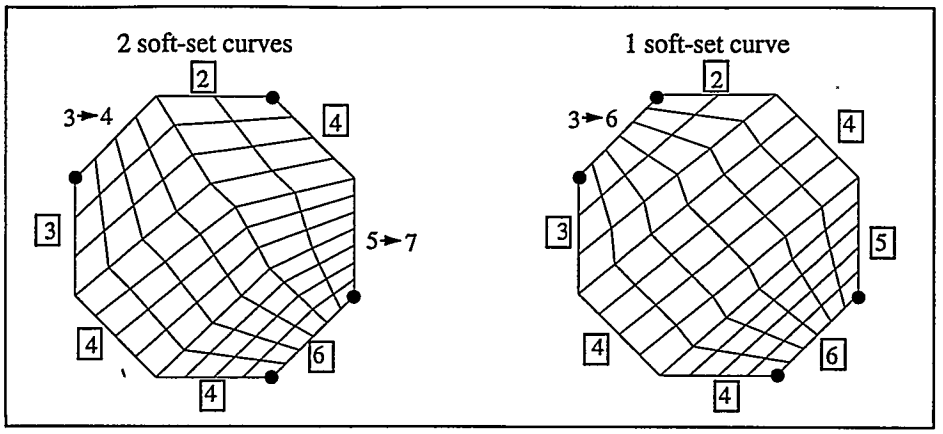

Figure 5. Provable corner picking plus heuristic improvement. Left 2-soft-set curves, right 1-soft-set curve case.

Figure 6. Corner picking and interval assignment on some long surfaces.

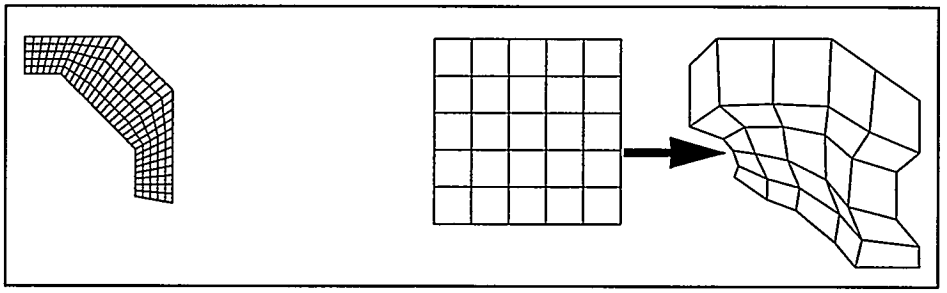

Figure 7. Some more corner picking and interval assignment examples.

3. Interval Assignment

We briefly give an overview of the interval assignment algorithm for a general advancing-front hexahedral meshing algorithm. First corners are picked for the primitive and semi-structured surface meshing algorithms. Then the interval constraints for all surfaces are assembled into a mixed-integer linear program (MILP). The various surface meshing algorithms create two types of constraints. First, more structured algorithms create constraints of the "intervals on opposite sides are equal", or "sum of intervals on two sides are greater than the third" variety. Second, a quadrilateral mesh of a surface must have an even number of mesh edges on its boundary, so the unstructured algorithms create "sum of intervals on all sides are even" constraints. The sum-even constraints force the problem to be mixed-integer. The intervals themselves must also be integer, but this can often be obtained "for free" by careful MILP formulation.

$$\sum_{s \in curves} intervals_s = 2k \qquad k \in integers$$

The set-up is as follows: The soft-set curve intervals are the variables of the mixed-integer linear program (MILP). We add two extra *delta* variables for each curve, that compute the positive and negative difference of the assigned-interval to the goal-interval. We weight the deltas inversely proportional to the interval goal (to compute relative change). We add another variable M that computes the maximum of the weighted deltas.

The solution process has two steps. In the first step we relax the integer variables: We solve the LP with objective minimizing M. We find a curve that forced M to be as large as it is, and fix that curve's intervals to be the nearest integer value and remove it from the LP. We repeat until all curves are fixed or M is zero. At the end of the first step we have an integer solution that satisfies all of the constraints except the sum-even constraints.

In the second step we set tight bounds on the sum-even "k" and curve variables, then solve the MILP minimizing the weighted sum of intervals (for speed). The Branch & Bound procedure may take too long (for a given set of bounds it may take exponential time). If this is the case, we try less tight bounds. We have four sets of bounds. There is guaranteed to be a solution, but no guarantee on the running time.

Our techniques give interval assignments that have very high fidelity to the user-desired goals: Most previous work relies on minimizing the sum of differences between assigned and goal intervals, whereas we essentially minimize the lexicographic vector of differences. We have noted cases where we succeed but some commercial packages fail to assign intervals compatible with meshing the surfaces. However, our techniques are slower because we iteratively solve the relaxed LP, taking time $O(n^3)$ rather than $O(n^2)$.

Our techniques are practical for models with up to a few thousand curves (or more if we're lucky and certain goals and meshing algorithms are chosen). To progress beyond that we conjecture that the problem should be divided up into subproblems, as is typically done with large LPs. Figure 8 shows some real-world models with about 500 curves each.

3.1 Ensuring the existence of a compatible hexahedral mesh

We consider the problem of generating a hexahedral mesh of a volume that exactly matches a quadrilateral mesh of the surfaces bounding the volume. Mitchell[5] notes that for such a compatible hexahedral mesh to exist for a handlebody, there must be an even number of quadrilaterals, and certain edge cycles (the ones contractible to a point in the volume) must be even. Being able to insert cutting disks along representative cycles is sufficient to show the existence of a hex mesh. See also Thurston[6]. Eppstein[7] gives a more constructive proof of the existence of a hexahedral mesh, in the case that the quadrilateral mesh is even and two-colorable. (Eppstein's sufficient condition is stronger in that two-colorable implies that **every** edge-cycle is even, yet weaker in that it is not restricted to handlebodies.)

Here we give a practical way to construct a quadrilateral mesh satisfying Eppstein's sufficient conditions, ensuring that a compatible hexahedral mesh exists. In essence, we reduce the existence problem to a surface cutting problem

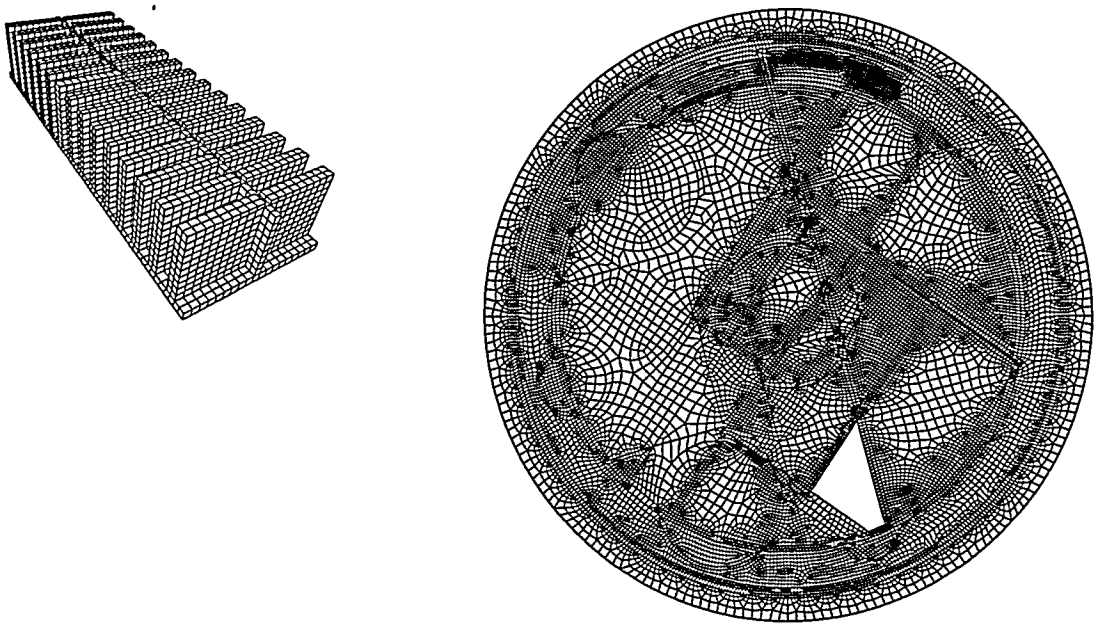

Figure 8. Left, a heat-sink meshed with a generalization of mapping. Right, a mostly unstructured mesh of a complex of 2d surfaces.

and a trivial interval assignment problem. (At the time of this draft we have not yet implemented the surface cutting for all types of surfaces.)

First, each surface bounding the volume must be "cut" by inserting two-sided curves, to reduce its topology to that of a disk. See Figure 9 left. This is somewhat analogous to inserting the cutting disks of Mitchell[5], but in a lower dimension and it does not restrict us to handlebodies. Also, finding these cutting curves is a much easier problem, perhaps solved by sweeping, particularly if we restrict to the typical case that the volume is defined by constructive solid geometry, and each surface is a subset of a plane, a sphere, or a cone. In the general case, finding cutting curves could be accomplished by forming a quadrilateral mesh of the surface, then retaining as cutting curves some chains of mesh edges connecting disjoint components of the surface boundary (one-sided surfaces don't bound a volume, so they need not be considered).

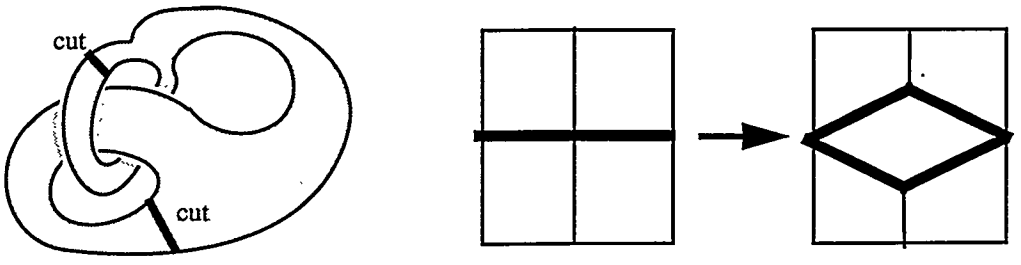

Figure 9. Left, cutting a surface into a topological disk by introducing two-sided cutting curves. Right, adding a quad.

We can ensure that every curve (including the cutting curves) has an even number of intervals or mesh edges, and simultaneously satisfy the meshing algorithm requirements, by adding constraints to the interval assignment MILP (see section 3). We mesh the curves, then the surfaces with a quadrilateral meshing algorithm. (Every surface admits a quad mesh, since the number of bounding mesh edges is even.) If there is an odd number of quadrilaterals, then we add one e.g. by opening a pair of mesh edges sharing a mesh node into a quadrilateral; see Figure 9 right. We claim the surface mesh is two-colorable (see Theorem 1), so by Eppstein[7] a hex mesh exists.

Theorem 1 If all bounding curves have an even number of intervals, and every bounding surface is topologically a disk, then any even quadrilateral mesh of the bounding surfaces admits a compatible hexahedral mesh.

Proof. We claim the mesh of each surface is two-colorable red and green, with vertices of the surface red, hence the surfaces taken together are two-colorable. First, the mesh of the curves is two-colorable with all vertices red, since any path between two vertices is even (since curves only meet at vertices and curves have an even number of mesh edges). For each surface, there is a two-coloring that respects this coloring of the curves: First, each surface has the topology of a disk, and hence any quadrilateral mesh of each surface has a two-coloring[7]. Second, vertices are separated by an even length chain of edges and hence must all be the same color, so chose red.

Acknowledgements

I wish to thank Sandia's Discrete Algorithms Group (DAG) and especially Jonathan E. Atkins for contributions to section 2.2.

References

[1] T. D. Blacker, W. J. Bohnhoff, T. L. Edwards, J. R. Hipp, R. R. Lober, S. A. Mitchell, G. D. Sjaardema, T. J. Tautges, T. J. Wilson, W. R. Oakes, S. Benzley, J. C. Clements, L. Lopez-Buriek, S. Parker, M. Whitely, D. White, and E. Trimble. CUBIT mesh generation environment volume 1: users manual. SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico, May 1994.

[2] Rolf H. Möhring, Matthias Müller-Hannemann, and Karsten Weihe. Mesh refinement via bidirected flows: Modeling, complexity, and computational results. Technische Universität Berlin, Department of Mathematics, Report No. 520 / 1996. URL: http://www.math.TU-Berlin.DE/~mhannema/projecteng.html

[3] T. K. H. Tam and C. G. Armstrong. Finite element mesh control by integer programming, International Journal for Numerical Methods in Engineering, vol 36, 2581-2605, 1993.

[4] Esther M. Arkin, L. Paul Chew, Daniel P. Huttenlocher, Klara Kedem, and Joseph S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes, Cornell University, Department of Computer Science, TR 89-1007, May 1989.

[5] Scott A. Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume, Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS '96), Lecture Notes in Computer Science 1046, pages 465-476, Springer, 1996. Also http://sass577.endo.sandia.gov/9225/Personnel/samitch/exist-abstract.html

[6] William 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.

[7] David Eppstein. Linear complexity hexahedral mesh generation. 12th ACM Symp. Comp. Geom., Philadelphia (1996) 58-67. Also http://www.ics.uci.edu/~eppstein/