LA-UR-95- 4470

CONF-960489--1

Ray Tracing for Point Distribution In Unstructured Grid Generation

Ahmed Khamayseh[1], Frank Ortega[2], and Harold Trease[1]
Communication and Computing Group[1]
Computational Science Methods Group[2]
Los Alamos National Laboratory
Los Alamos, NM 87545

To be submitted for publication in the Proceedings of the Fifth International Conference on Numerical Grid Generation in Computational Fluid Dynamics.

Ray Tracing for Point Distribution in Unstructured Grid Generation

Ahmed Khamayseh, Frank Ortega, and Harold Trease

Los Alamos National Laboratory
Los Alamos, New Mexico, 87545, U.S.A.

Abstract. We present a procedure by which grid points are generated on surfaces or within three-dimensional volumes to produce high quality unstructured grids for complex geometries. The virtue of this method is based on ray-tracing approach for curved polyhedra whose faces may lie on natural quadrics (planes, cylinders, cones, or spheres) or triangular faceted surfaces. We also present an efficient point location algorithm for identifying points relative to various regions with classification of inside/on/outside.

1. Introduction. Algorithms that generate grids for complex geometries with multi-materials are still in a primitive state. Typically, an unstructured grid is generated by regarding the boundary triangulation as a front in which a new layer of tetrahedra is built. As a result, the original front triangles become interior faces of the mesh and a new set of front faces is created. The algorithm continues to build tetrahedra on the new front, growing more tetrahedra until the entire region has been filled. This procedure of constructing unstructured grids is commonly known as the *advancing front* method, see Löhner [3] and Pirzadeh [6]. A particular difficulty of this method occurs in the closing stages of the procedure when the front is folding in on itself and the final vestiges of the empty space are replaced by tetrahedra. It is clearly necessary, in the final stages, to maintain good control over the size of the front faces of the unfilled region that is left.

The current study introduces an alternative approach to the popular *advancing front* approach. The first step involving the generation of a tetrahedral grid is creating the three-dimensional surface models with closed regions. These regions are composed of a Boolean combination of quadrics and/or triangulated surfaces. The second step is to distribute a set of points within these regions as well as on the region surfaces, and this step is the concern of this paper. The third step is the classification, *i.e.*, inside/on/outside, of the points within each region. The fourth step is the Delaunay tetrahedralization of the points, which is required for finite volume or finite element computation. This step is called the grid reconnection step which involves Lawson flipping (a technique commonly used to generate Delaunay triangulations), see Lawson [2]. Lawson flips break the connectivity of the mesh and establish a Delaunay triangulation, which is a common requirement for computation. For a detailed discussion on reconnecting and flipping algorithms, we refer to Painter and Marshall [5] and Trease [7]. Finally, multi-material connections are broken at the material interfaces and new tetrahedra are created to preserve these interfaces.

Ray-tracing technique casts a set of rays through a specified region and determines the points of intersection with the region surface. Points are then distributed along the ray between the intersection points. In this paper we explore different types of point distribution based on ray-tracing technique. In all cases, nodes are distributed on the surfaces and within the volume of the geometry. In order to represent the actual spatial geometry most accurately, we may choose among Cartesian, cylindrical, or spherical coordinate systems for each region. In order to concentrate nodal density where needed to achieve the highest solution accuracy, we may use proportional spacing where the distance between nodes is graded evenly away from the area needing highest resolution.

We also present an algorithm that is both general and efficient for identifying points relative to various regions with classification of inside/on/outside. Of course, the difficulty of this task depends on the nature of the closed region. To identify a query point (inside/on/outside) relative to a region composed of quadrics, the query point must be tested with respect to each of the region's individual surfaces and compared with the Boolean set operations that define the region. Testing a query point with respect to quadrics is fairly straightforward.

The ambiguity arises when classifying a query point with respect to arbitrary surfaces. We will present an efficient algorithm that addresses the problem of classifying a query point with respect to triangulated polyhedra. The essence of this algorithm depends on locating the point on the surface of the polyhedron nearest to the query point. Then an outward normal at the nearest point is prepared for the testing. The practicality of this algorithm is demonstrated by its robustness and efficiency for querying points, not only relative to closed polyhedra but also to nearly closed polyhedra as well.

The paper begins with a detailed discussion of each approach of ratio zoning using ray tracing. It then describes the point location algorithm with the various steps of the algorithm to locate the nearest point on the surface of the polyhedron to the query point. A series of applications are then shown demonstrating the practicality of the present approach of generating unstructured grids for complex configurations.

2. Point distribution using ray tracing. In order to create an efficient computational unstructured grid where the grid vertices are determined by pre-determined points, the points must have the following properties. First, some points must lie on the surface of the three-dimensional geometry that composes a material region so that multi-material grid faces are correctly generated. Second, the point distribution should match the surface contour in as many point layers as possible. Third, the user should be able to distribute points so that the point spacing provides the basis for the best computational grid possible. The ray-tracing (or ray-casting) technique provides the best method to accomplish these tasks. By casting an appropriate set of rays through a selected, pre-defined geometric region, the intersection(s) of the ray with the surface of the geometric region can be determined. These intersections provide definite line lengths whereby points can be distributed and spaced according to the user's needs. For example, M points can be evenly distributed along the line segments, or the points can be ratio zoned so that more points are packed near one end of each of the line segments.

There are four ray-casting methods used to distribute points. The first method casts rays from a user-defined reference plane through a set of user-defined points (see Figure 1). The rays are cast normal to the reference plane through the points. For efficiency, the user-defined points should lie on a plane that is almost parallel to the reference plane. This planar method is suitable for creating a Cartesian based grid.

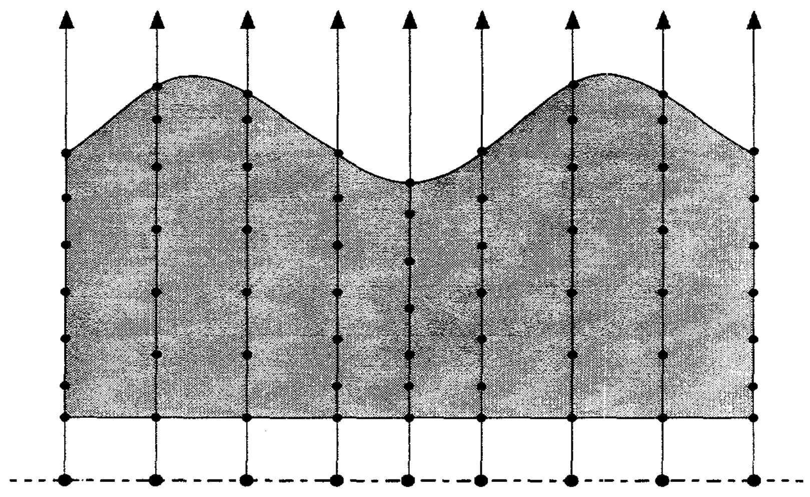

Figure 1. *Node distribution by casting rays from a reference plane.*

The second ray-casting method casts rays from a user-defined reference line through a set of user-defined points (see Figure 2). The rays are cast normal to the reference line through the points. If the user-defined points are generated as a cylinder about the reference line, the resulting point distribution will be cylindrical. The cylindrical method is suitable for cylinder or axis of rotation type of geometries.

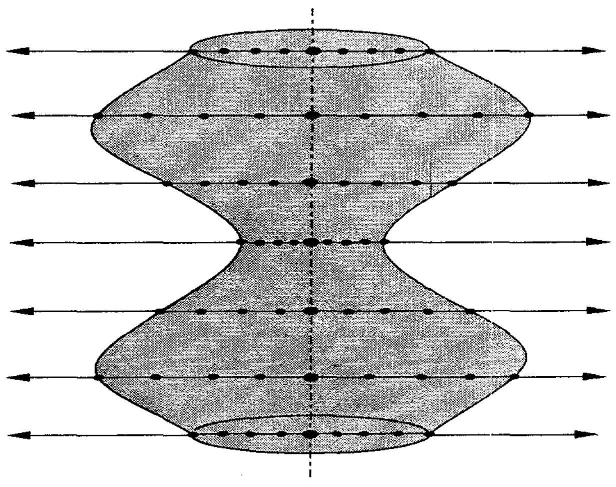

Figure 2. *Node distribution by casting rays from an axis line.*

The third ray-casting method sends rays from a user-defined point source through a set of user-defined points (see Figure 3). If the user generates a spherical set of points about the point source, a spherical-like point distribution will be generated.

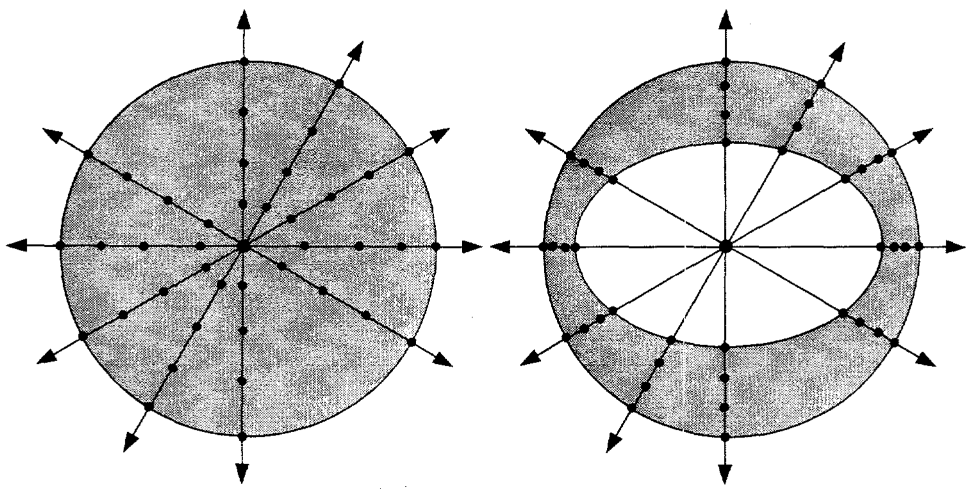

Figure 3. *Node distribution by casting rays from a point source.*

Finally, the fourth ray-casting method is the centerline method (see Figure 4). This method was designed to distribute points within a highly curved closed geometry such as a manifold or oil well bore. In this method a centerline composed of connecting line segments is created with points distributed along the centerline. Rays are then cast from these centerline points normal to the line segment the point lies on. If a cast point lies on the end of two line segments, the average normal of the two line segments is used. Multiple rays are cast from each cast point in a circular arc about the line segment.

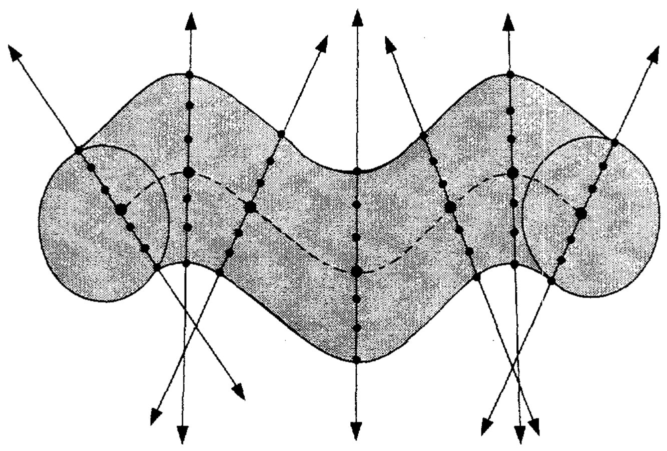

Figure 4. *Node distribution by casting rays from a "centerline".*

The algorithm that generates line segments along the ray is the same for all four ray-casting methods. The algorithm treats each ray as an infinite line. Then, for the selected geometric region, all intersections of the ray with all the surfaces that determine the geometric region are calculated. These intersection points are then tested with the Boolean surface combinations that define the geometric region surface to determine if the point lies on the region surface. These region surface intersections are sorted along the ray from the ray's origin to create a set of line segments. The midpoint of each line segment is tested to see if the midpoint lies inside the geometric region. If the segment midpoint is inside the region, M points are distributed along the line using the user-selected distribution method (*e.g.*, uniform spacing, ratio zoning spacing). The algorithm that determines whether a point is inside, on or outside a specified geometric region is described in the next section.

3. Point classification algorithm. Suppose we are given a polyhedron $\mathcal{P}$ defined by a set of nodes $\{\mathbf{r}_j\}_{i=1}^{n}$ and a set of triangular facets $\{\mathcal{T}_i\}_{i=1}^{N}$. Thus,

$$\begin{aligned}
\mathcal{P} &= \bigcup_{i=1}^{N} \mathcal{T}_i \\
\mathcal{T}_i &= (\mathbf{r}_{i_1}, \mathbf{r}_{i_2}, \mathbf{r}_{i_3}) \quad \text{for} \quad i = 1, ..., N \\
\mathbf{r}_j &= (x_j, y_j, z_j) \quad \text{for} \quad j = 1, ..., n
\end{aligned}$$

We assume $\mathcal{P}$ is not self-intersecting and is oriented in a fashion consistent with the orientations of the individual triangular facets.

Suppose we are now also given a query point q and it is our task to obtain the inside/on/outside classification of q with respect to $\mathcal{P}$. Our approach is to obtain the point p on $\mathcal{P}$ nearest to q in the standard Euclidean metric. We then devise a "normal" $\mathbf{n}_p$ at p such that the following *normal at nearest point test* can be proved to always provide a correct result:

```
if  $p = q$  then $q$ is on $\mathcal{P}$
elseif  $(q - p) \cdot \mathbf{n}_p < 0$  then $q$ is inside $\mathcal{P}$
else  $q$ is outside $\mathcal{P}$
```

Now if $\mathcal{P}$ were a surface of C^1 continuity, it would be easy to see that by choosing $\mathbf{n}_p$ to be the "classical normal" (which is defined on such a smooth surface) then the "normal at nearest point test" would always work. However, here $\mathcal{P}$ is a polyhedron consisting of triangular facets, and so the "classical normal" does exist when p is in the interior of a triangle, but it does *not* exist when p is on the shared edge of two triangles or is the shared vertex of m triangles. Thus a good method for constructing an $\mathbf{n}_p$ in these latter two cases is crucial for the correctness of the "normal at nearest point test".

We now present a method for choosing $\mathbf{n}_p$ in all cases. Define $\mathbf{n}_p$ to be a generalization of the classical normal as follows: Determine if p lies in the interior of a triangle, on the shared edge of two triangles, or on the shared vertex of m triangles. Then

$$\begin{aligned}
&(i) \quad \text{if} \quad p \in \mathcal{T}_i \quad \text{then} \quad \mathbf{n}_p = \mathbf{n}_i \\
&(ii) \quad \text{if} \quad p \in \mathcal{T}_{i_1} \cap \mathcal{T}_{i_2} \quad \text{then} \quad \mathbf{n}_p = \frac{\mathbf{n}_{i_1} + \mathbf{n}_{i_2}}{||\mathbf{n}_{i_1} + \mathbf{n}_{i_2}||} \\
&(iii) \quad \text{if} \quad p \in \cap_{i=1}^{m} \mathcal{T}_i \quad \text{then} \quad \mathbf{n}_p = \frac{\sum \theta_i \mathbf{n}_i}{||\sum \theta_i \mathbf{n}_i||}.
\end{aligned}$$

Here $\mathbf{n}_i$ is the outward normal on $\mathcal{T}_i$ and θ_i is the inclusion angle at the vertex p of $\mathcal{T}_i$ as shown in Figure 5.

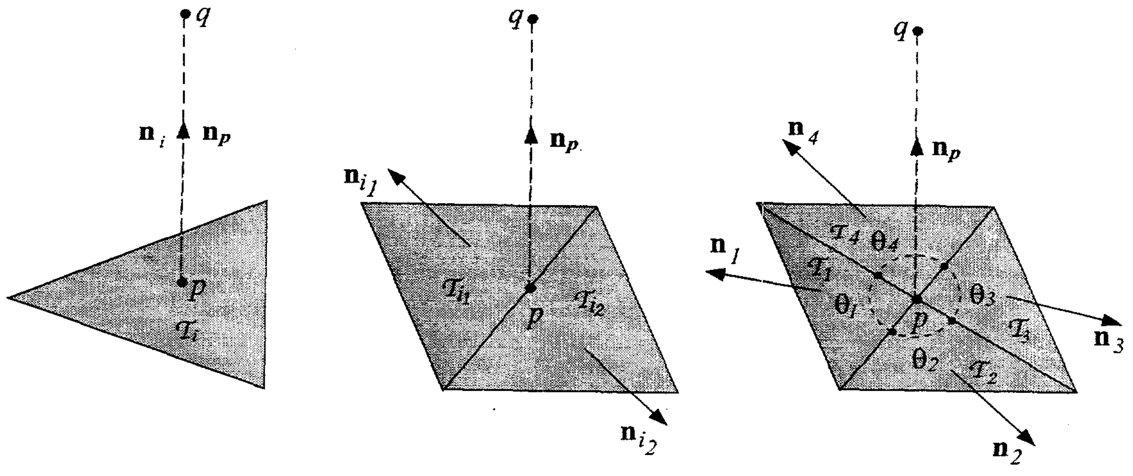

Figure 5. *Computation of synthetic normal in interior, edge, and vertex cases of location of nearest point.*

Note that in all three cases this definition for $\mathbf{n}_p$ is equal to the following surface integral:

$$\mathbf{n}_p = \frac{\int_{B_\epsilon(p)\cap\mathcal{P}} \mathbf{n}\, dS}{\|\int_{B_\epsilon(p)\cap\mathcal{P}} \mathbf{n}\, dS\|},$$

where $B_\epsilon(p)$ is a solid ball of sufficiently small radius at p, and $\mathbf{n}$ is the outward normal on the surface $B_\epsilon(p)\cap\mathcal{P}$. This "synthetic normal" of course agrees with the classical normal wherever the latter is defined on $\mathcal{P}$. A rigorous mathematical proof of the correctness of the algorithm which involves the *synthetic normal* can be found in Khamayseh *et al.* [1].

In our algorithm for querying points relative to a given polyhedron, the nearest point to each query point needs to be located on the surface of the polyhedron. This step is the most computationally expensive and time consuming one. In order to speed up the algorithm and avoid searching every triangle for the nearest point, we propose an algorithm that will cut down on the *search time*, that is, the number of operations required to locate the nearest point on the polyhedron.

We now employ an octree subdivision of the polyhedron that will reduce the complexity of the search algorithm from $O(MN)$ to $O(M \log N)$ where M is the number of query points and N is the number of triangular facets. The construction of an octree data structure for arbitrary three-dimensional objects is discussed in Meagher [4]. The octree affords a substantial decrease in execution time because the spatial information contained in the octree allows rapid culling of large numbers of triangles that cannot possibly contain the nearest point to the query point.

We start out by generating a *global* bounding box $\mathbf{B}_\mathcal{P}$ which is the smallest box (with sides parallel to the three Cartesian coordinate planes) that will completely enclose the polyhedron. We construct an octree hierarchical data structure by successively subdividing the bounding box $\mathbf{B}_\mathcal{P}$ in each dimension to form eight sub-boxes or octants. The subdivision is continued until some stopping criterion is satisfied. Each node in the tree corresponds to some region of space bounded by the box associated with that node. Since the tree is constructed in the Cartesian space, the faces of the box are subsets of the planes of constant-x, constant-y, and constant-z. Therefore, only six coordinate extrema are stored. If the node is not a terminal or leaf node, then it has eight children or subboxes, the union of which completely fills the parent box associated with the node. In the current implementation, the polyhedron data associated with an octant is stored in the form of an array of pointers to surface triangles associated with that octant.

We test whether q lies within the bounding box of $\mathcal{P}$. If $q \notin \mathcal{B}_{\mathcal{P}}$, then q is outside the polyhedron. Otherwise, we locate the leaf octant $\mathcal{O}$ in which q resides. The parent octant of $\mathcal{O}$ must contain a nonempty leaf octant $\mathcal{O}'$, or else it would never have been subdivided. We compute the shortest distance from q to all the vertices of all the facets residing in such a nonempty leaf octant $\mathcal{O}'$:

$$r \;=\; \min_{\mathcal{T}_i \in \mathcal{O}'} \left\{ ||q - \mathbf{r}_{i_1}||, ||q - \mathbf{r}_{i_2}||, ||q - \mathbf{r}_{i_3}|| \right\}.$$

Then we create the query point bounding box

$$\mathbf{B}_q \;=\; [x_q - r, x_q + r] \times [y_q - r, y_q + r] \times [z_q - r, z_q + r].$$

It is now sufficient to search the subset of triangles residing in leaf octants which overlap with $\mathbf{B}_q$. The search speed increases exponentially with the depth of the octree, since children of octants that are not intersected by the box $\mathbf{B}_q$ may be eliminated.

Now for each triangle residing in an overlapping leaf octant, we need to locate the nearest point p_i to q. p_i will be either on a vertex, edge, or in the interior of the triangle. We start first by computing the distance between q and each of the three triangle vertices, recording the nearest vertex—a possible candidate for p_i.

Second, we loop over the triangle edges and evaluate

$$t \;=\; \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||^2},$$

where $\mathbf{a} = \mathbf{r}_{i_2} - \mathbf{r}_{i_1}$ and $\mathbf{a} = q - \mathbf{r}_{i_1}$. If $t \in (0, 1)$ then the nearest point possibly belongs to the edge and is computed as

$$p_i \;=\; t\mathbf{r}_{i_2} + (1 - t)\mathbf{r}_{i_1}.$$

This test is performed over the three edges $l_1 = (\mathbf{r}_{i_1}, \mathbf{r}_{i_2})$, $l_2 = (\mathbf{r}_{i_2}, \mathbf{r}_{i_3})$, and $l_3 = (\mathbf{r}_{i_2}, \mathbf{r}_{i_3})$.

Third, we need to check if the p_i lies in the interior of the triangle. (We need to do so if $t \in (0, 1)$ is satisfied on two or three edges.) From q we move in the direction $\mathbf{n}_i$ by a distance d to get the projection point q_i in the plane of the triangle $\mathcal{T}_i$. The projection point q_i is computed using the formula

$$q_i \;=\; q - d\mathbf{n}_i$$

where $d = (q - \mathbf{r}_{i_1}) \cdot \mathbf{n}_i$. Now we determine if q_i lies inside or outside of the triangle. This is accomplished by subdividing the triangle into three triangles $\mathcal{T}_1 = \{\mathbf{r}_{i_1}, \mathbf{r}_{i_2}, q_i\}$, $\mathcal{T}_2 = \{\mathbf{r}_{i_2}, \mathbf{r}_{i_3}, q_i\}$, and $\mathcal{T}_3 = \{\mathbf{r}_{i_3}, \mathbf{r}_{i_1}, q_i\}$. For the projection point q_i to lie inside the triangle, each individual area of the three sub-triangles must be positive, *i.e.*, $area(\mathcal{T}_1) > 0$, $area(\mathcal{T}_2) > 0$, and $area(\mathcal{T}_3) > 0$.

We conclude that p_i, the nearest point on triangle $\mathcal{T}_i$ to the query point q, is the nearest of the candidate points obtained when considering the vertices, edges, and the interior of the triangle. Finally, the nearest point p on $\mathcal{P}$ to q is the nearest p_i over all the triangles $\mathcal{T}_i$ in the overlapping leaf octants.

4. Applications. The algorithms outlined in this paper are currently used in the grid generation and modeling code X3D at Los Alamos National Laboratory. X3D is a tetrahedral grid generator used to solve time-dependent, multi-material grid generation problems. We are currently applying this grid generation system to several classes of problems involving complex geometries, multiple materials, and time-dependent physics. The physics operators that we are solving on our tetrahedral grids include computational fluid dynamics (CFD), diffusion, and Monte Carlo transport. The current areas of application involve radioactive waste disposal, oil-reservoir simulation, semiconductor design, and automotive CFD applications. Many of the geometric modeling concepts, algorithms, and implementations found in the X3D system are based on the methods discussed in Trease [7].

Two examples are presented to demonstrate the robustness and usefulness of our techniques for point distribution and point location in the generation of grids for complex geometries. Figure 6 shows an exploded view of distinct materials with grids in a MOSFET semiconductor device with curvilinear interfaces.

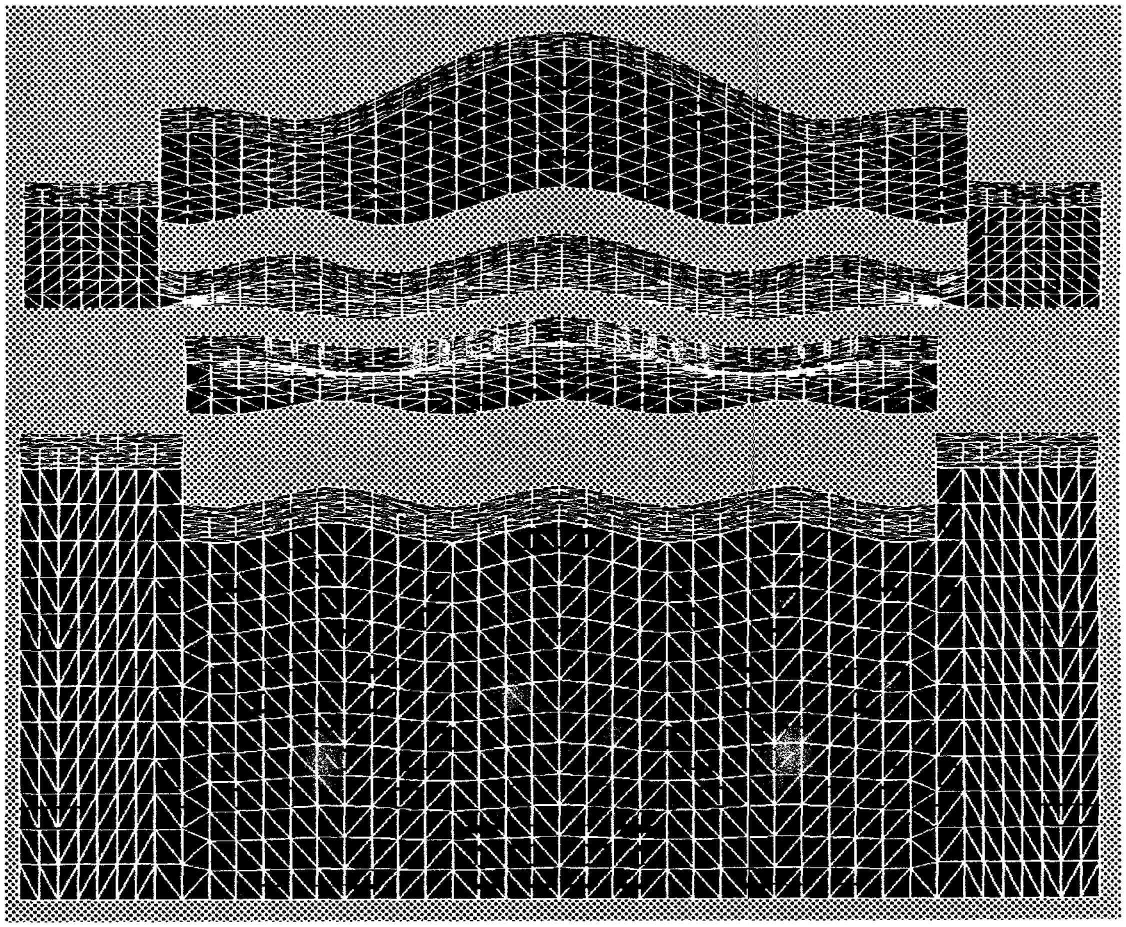

Figure 6. *Exploded gridded regions of a MOSFET semiconductor device with curvilinear interfaces.*

The polyhedral regions corresponding to each individual material are constructed using Boolean set operators on a combination of quadric and triangulated surfaces. The grid points in each sub-region were distributed by casting rays from a reference plane to produce a Cartesian type of point distribution.

Figure 7 shows an exploded view for a multi-material surface of revolution, challenging because of the presence of the needlelike protrusions of the "cap" and the corresponding indentation in the complementary part. In this example the grid points in each sub-region were distributed by casting rays from an axis line to produce a cylindrical type of point distribution. The graphics for these two examples indicate the lack of any incorrect point classifications due to the algorithm, because any such errors would result in visible roughness on the interfaces between the exploded materials.

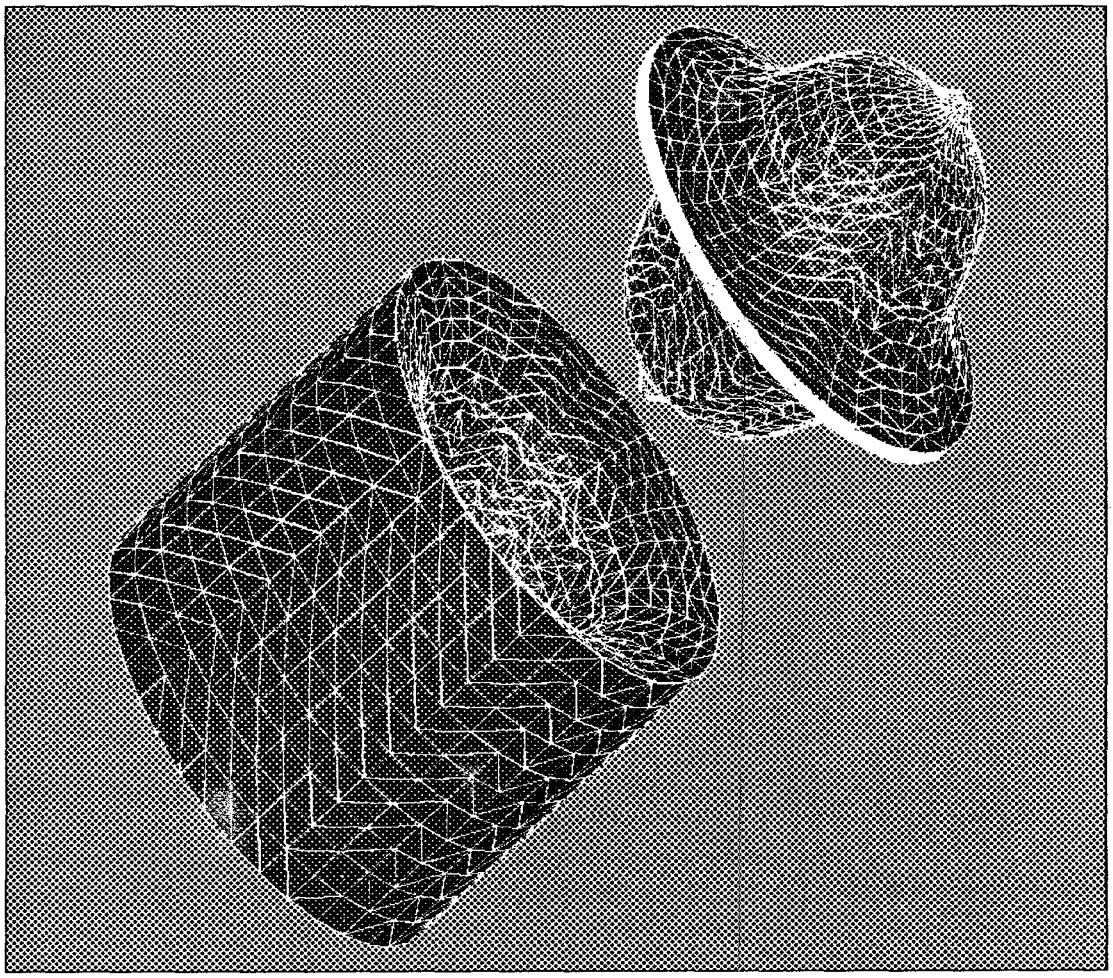
Figure 7. *Exploded gridded regions of a challenging geometry containing sharp protrusions.*

5. Conclusion. This paper describes an alternative method to the *advancing front* method in which the grid points are concentrated where they are needed to increase the accuracy of the solution. The four options of point distribution are based on casting rays through the material regions and finding the intersections of the rays with surfaces of the regions whereby points are distributed. These options were designed to create a distribution of grid points that best fits the geometric configuration. This point distribution is necessary in order to generate high quality unstructured grids while still preserving multi-material interface integrity.

We have also presented the "normal at nearest point" algorithm for solving the point location problem for polyhedra. This technique has been chosen for solving all point location problems in Los Alamos National Laboratory's X3D grid generation code. Using X3D, we have successfully tested the algorithm on complex grids involving nontrivial material interfaces and have detected no point location errors even under finite machine precision.

REFERENCES

[1] A. Khamayseh, F. Ortega, and A. Kuprat, *A Robust Point Location Algorithm for General Polyhedra*, Computer Aided Geometric Design, (submitted).

[2] C. L. Lawson, *Software for C^1 Surface Interpolation*, in Mathematical Software III, edited by John R. Rice, Academic Press, 1977.

[3] Löhner, *Generation of Three-Dimensional Unstructured Grids by the Advancing Front Method*, AIAA 26th Aerospace Science Meeting, Reno, AIAA Paper 89–0365, 1989.

[4] D. Meagher, *Geometric modeling using Octree Encoding*, Computer Graphics and Image Processing, vol. 19, pp. 129–147, 1982.

[5] J.W. Painter and J.C. Marshall, *3-D Reconnection and Fluxing Algorithms*, Proceedings of the second Free Lagrange Conference, Lecture Notes in Physics, Springer-Verlag, New York, vol. 395, pp. 139-148, 1990.

[6] S. Pirzadeh, *Unstructured Viscous Mesh Generation by the Advancing Layers Method*, AIAA 11th Applied Aerodynamics Conference, Monterey, AIAA Paper 93–3453, 1993.

[7] Trease, H.E., *Three-Dimensional Free Lagrangian Hydrodynamics*, Proceedings of the first Free Lagrange Conference, Lecture Notes in Physics, Springer-Verlag, New York, vol. 238, pp. 145-157, 1985.