UNSTRUCTURED GRID GENERATION USING THE DISTANCE FUNCTION

Barna L. Bihari and Sukumar R. Chakravarthy
Rockwell International Science Center
Thousand Oaks, California

ABSTRACT

A new class of methods for obtaining level sets to generate unstructured grids is presented. The consecutive grid levels are computed using the distance function, which corresponds to solving the Hamilton-Jacobi equations representing the equations of motion of fronts propagating with curvature-dependent speed. The relationship between the distance function and the governing equations will be discussed as well as its application to generating grids. Multiply connected domains and complex geometries are handled naturally, with a straightforward generalization to several space dimensions. The grid points for the unstructured grid are obtained simultaneously with the grid levels. The search involved in checking for overlapping triangles is minimized by triangulating the entire domain one level at a time.

INTRODUCTION

A paper on fronts propagating with curvature-dependent speed by Osher and Sethian (Ref. 6) has motivated us to research the possibility of applying the theory of propagating fronts to grid generation. While this theory has a potential to be used for generating structured grids, powerful and well-tested methods already exist to tackle that problem. However, generating the appropriate unstructured mesh for use with finite element or finite difference methods is still a difficult step given the fact that the success of the method depends largely on a correct discretization of the domain. Several ideas and practical algorithms have been proposed in the past (Ref. 2-5,7); however, this techniuqe is radically different from all of them in several ways.

The idea underlying the grid generation procedure presented comes from the relationship between grid levels and fronts propagating with curvature-dependent speed. The front at the next time level corresponds to the next grid level and satisfies all the requirements generally imposed on a well-generated grid. The equation of motion for a front propagating with curvature-dependent speed is an initial-value Hamilton-Jacobi equation with a right-hand side that depends on curvature effects. The surface is viewed as a level set of the solution to this equation which, as it evolves in time, yields a different level set and thus, the next grid level.

While this method will work with any consistent initialization of the Hamilton-Jacobi variable, by carefully choosing the initial condition, the equation may not have to be solved. As will be shown later, setting the Hamilton-Jacobi variable to a function of the distance from a point in the computational domain to the initial surface, (i.e., geometry), results in a solution to the equation with no diffusion term. That is, the level set corresponding to a particular value is used as a grid level and, by repeatedly using a contour-plotter-like search algorithm, all the grid levels are readily obtained.

To accurately match prescribed outer boundaries of the domain, they are treated the same way as the geometry: they will be represented by another front that also moves, but inward. The strategy for triangulation is based on a "level-by-level" principle which assumes that all curves obtained from the contour plotter are closed curves (loops).

1. EQUATIONS OF MOTION

Some theoretical results will now be presented in anticipation of our later discussion of the actual grid generation scheme. Given a simple, closed, smooth initial curve $\gamma(0)$ in $\mathbb{R}^2$ (Ref. 6), let $\gamma(t)$, $t \in [0,\infty)$ be a one-parameter family of curves representing the grid levels. The $\gamma(t)$ curves are generated by propagating the initial curve normal to itself with speed $F = F(K)$, where K is the curvature. Let $X(s,t) = (x(s,t), y(s,t))$ be the position vector that parametrizes $\gamma(t)$ by s, $0 \le s \le S$, $X(0,t) = X(S,t)$. By convention, the interior is on the left in the direction of increasing s, resulting in a counter-clockwise orientation of γ. The equations of motion can now be written as:

$$x_t = F(K) \frac{y_s}{(x_s^2 + y_s^2)^{1/2}}$$
$$y_t = -F(K) \frac{x_s}{(x_s^2 + y_s^2)^{1/2}} \qquad (1.1)$$

with the initial condition $X(s,0) = \gamma(0)$, $s \in [0,S]$ given. The formula for the curvature is

$$K = \frac{y_{ss}x_s - x_{ss}y_s}{(x_s^2 + y_s^2)^{3/2}} \quad . \qquad (1.2)$$

With $t = f(x,y)$, K becomes

$$K = \frac{f_{xx}f_y^2 - 2f_{xy}f_xf_y + f_{yy}f_x^2}{(f_x^2 + f_y^2)^{3/2}} \quad .$$

As shown in Ref. 6, the function f satisfies

$$F^2(f_x^2 + f_y^2) = 1 \qquad (1.3)$$

if the curve γ stays smooth and nonintersecting.

Using these facts, the system (1.1) can now be transformed into a second order Hamilton-Jacobi equation. Let $\phi(x,y,t)$ be a Lipschitz continuous function such that $\phi(x,y,0) > 1$ inside the closed curve γ, $\phi(x,y,0) < 1$ outside γ, and $\phi(x,y,0) = 1$ on γ. We may then write

$$f_x = -\frac{\phi_x}{\phi_t}$$
$$f_y = -\frac{\phi_y}{\phi_t}$$

which, using Eq. (1.3) implies

$$F^2(\phi_x^2 + \phi_y^2) = \phi_t^2 \quad .$$

In general, the curve could propagate inward or outward, but keeping grid generation in mind, choose the direction of propagation to be outward, thus obtaining

$$\phi_t + F(K)H(\nabla\phi) = 0 \tag{1.4}$$

with $H(\nabla\phi) = -(\phi_x^2 + \phi_y^2)^{1/2}$, which is now a Hamilton-Jacobi equation in ϕ where

$$K = -\frac{\phi_{xx}\phi_y^2 - 2\phi_{xy}\phi_x\phi_y + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}} \quad .$$

After solving for ϕ in time, the position of the propagating curve at time t can be obtained by looking for the locations where $\phi(x,y,t) = 1$. By appropriately choosing the speed function F, the smoothness of the successive curves can be controlled as well.

2. INITIAL CONDITIONS

One simple and, as it will turn out, very useful choice for initializing ϕ is

$$\phi(x,y,0) = 1 \pm d((x,y);\ \gamma(0)) \tag{2.1}$$

where $d(X;\ \gamma)$ is the distance from point $X = (x,y)$ to the curve γ, and the "+" sign is chosen for points inside the curve $\gamma(0)$ and the "-" sign is chosen for the points outside the curve. Thus, ϕ will be exactly one for points lying on $\gamma(0)$. We shall term the initialization (2.1) as the ***distance function.***

Proposition: The level curve of the distance function at a level d_o is precisely the level curve of a solution ϕ to Eq. (1.4) with $F(K) = C$ that corresponds to $\phi = 1$.

Proof: $F(K) = C$, a constant speed function, implies that every point on the initial curve is moving normal to itself with the same speed; hence, all of them cover an equal distance δ in time τ. Thus, every point on the new curve $\gamma(\tau)$ will be δ away from the original curve $\gamma(0)$, where

$$\phi(x_0,y_0,0) = 1 \text{ for } \gamma(0) = \{(x_0,y_0)\},$$

and

$$\phi(x_1,y_1,\tau) = 1 \text{ for } \gamma(\tau) = \{(x_1,y_1)\}.$$

$\gamma(\tau)$ would therefore correspond to one level curve of the distance function, i.e.,

$$\phi(x_1,y_1,0) = 1 - \delta \operatorname{sgn} C.$$

This completes the proof.

Applying this result to grid level generation for unstructured mesh where the speed function F is constant, we note that no time integration of Eq. (1.4) is needed. Using the distance function, the grid levels are instantly obtained by simply specifying a set of values and then searching for the locations where the grid function (our Hamilton-Jacobi variable ϕ initialized by Eq. (2.1)) is equal to those values.

3. MESH GENERATION ALGORITHM

Using the above preliminary analysis, we have developed an algorithm that generates the nodes and triangulates between them one level at a time, also providing for arbitrary clustering of the triangles by modifying the distance function described in the previous section.

3.1 Grid Function

Because of the necessity of a grid function, the first step is to set up the computational domain which is a rectangular domain covering the entire region to be triangulated. This region will then be discretized as a cartesian grid, with equal spacing in both the x and y directions. Since this is merely an intermediate step, we keep this overlaid grid as simple and coarse as possible. Typically, the grid spacing will be greater or equal to the minimum side required for the unstructured elements to be generated.

To each point of this computational grid, we assign a grid function initialized as prescribed by Eq. (2.1). The spacing of the computational grid should be fine enough to resolve the sharp corners and the interior openings of the inner and outer boundaries. The boundaries are prescribed by the user in the form of patches, where the program has the built-in capability of point redistribution within each patch. The nodes of the interior boundaries are entered in counter-clockwise order, while the nodes of the exterior boundary are entered in clockwise order. All the curves representing the boundaries are assumed to be closed loops, with the possibility of several inner loops, corresponding to multiply connected domains (Fig. 1). For our example of Fig. 1, a contour plot of the grid function over the region of interest is illustrated in Fig. 2.

3.2 Generation of Interior Nodes

For each grid level, using a contour-plotter-like algorithm, we search for the locations in the computational domain, where the grid function matches a prescribed value. This value will obviously be less than one, and will depend on the nodal density and current grid level. It is important that the contour algorithm follows the contour levels in a continuous fashion, thus yielding closed curves. Simultaneously with following these contour curves, the nodes are generated as well by simply recording the coordinates of equally (or nonequally for varying nodal densities) spaced points along the curves. Once two complete adjacent contour levels are obtained with the corresponding points, no more points are generated, until the triangulation of this set of "ribbons" is complete (Figs. 3 and 4). This inherent topological structure of the nodes is a main advantage of the method, since it greatly reduces the search time necessary to form the best triangles.

SC51777

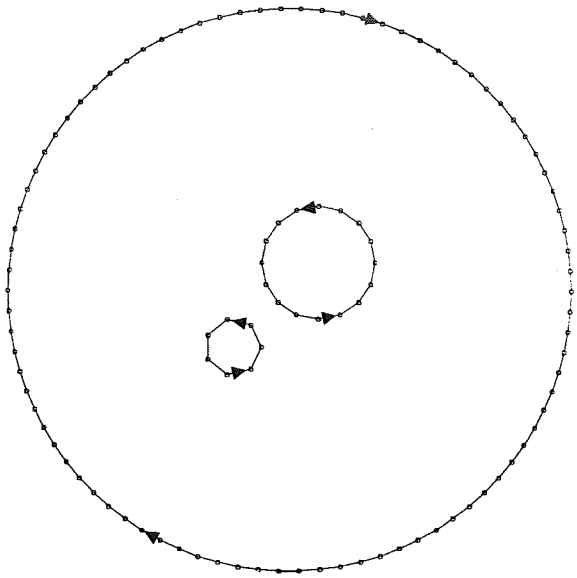

Fig. 1

Boundary of the domain to be triangulated.

SC51778

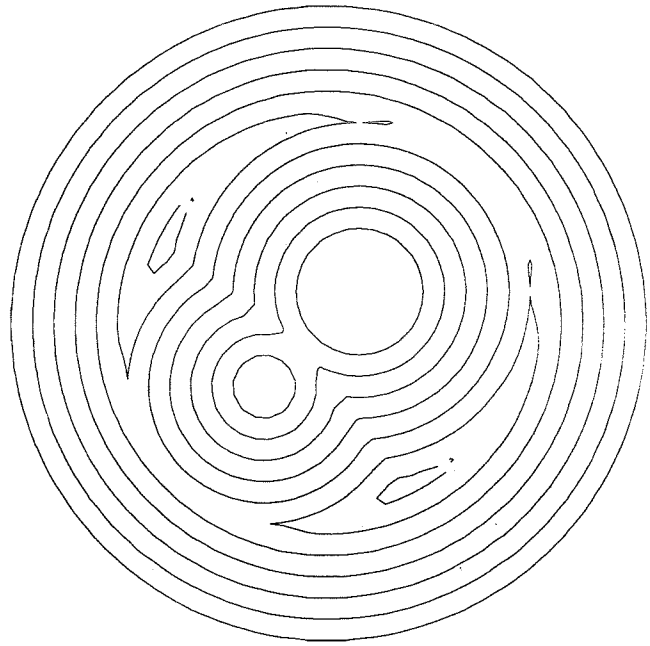

Fig. 2

Contour level curves of the grid function between 0 and 1

SC51779

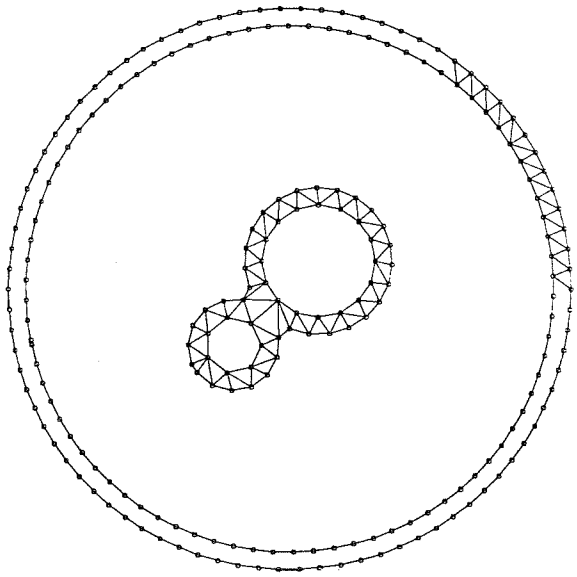

Fig. 3

Triangulation in progress in the first subregion: between the initial curves and the first level set.

SC51780

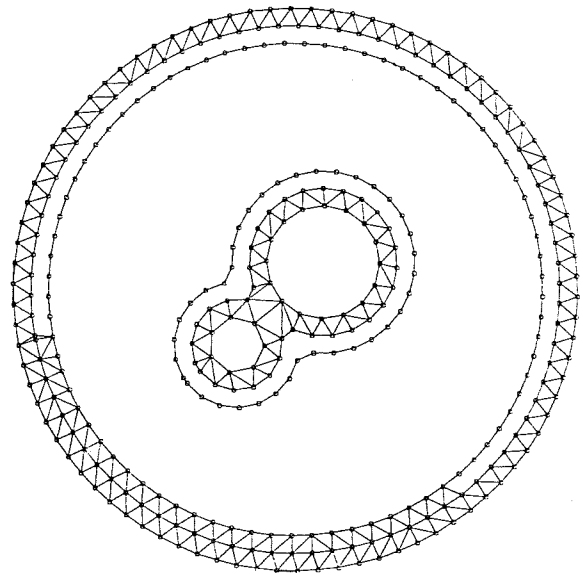

Fig. 4

Triangulation in progress in the second subregion: between the first and second level sets.

3.3 Forming Triangular Elements

Once two adjacent contour levels are known, the triangulation in the region enclosed by them is relatively straightforward. The algorithm that connects the nodes to form triangular elements has the following major steps:

1. Choose an initial "base" AB by connecting two adjacent nodes A and B on one of the contour loops that corresponds to a higher contour level.
2. For each triangle to be formed do:
   - 2a. Using the current base AB as the base of the triangle, choose the third vertex C such that the resulting triangle will, in some sense, be optimal. The criterion used will be elaborated later.
   - 2b. Update the array that contains the sides, with information about the newly created triangle, as well as the array that contains the elements (triangles).
   - 2c. Choose a new base AB from one of the two newly created sides AC or BC of the latest element, and if the other side does not belong to a contour loop, enqueue it for later use. If no more sides are available as eligible bases, stop. Otherwise go back to step 2a.

Criterion for determining vertex C:

The base AB is a directed vector, and the third vertex C will only be chosen from points to its ***right***. All those nodes that belong to the contour loops enclosing our current region of interest are considered. After analyzing the criteria proposed by Cavendish (Ref. 2) and Lo (Ref. 3), we found that those are unnecessarily complicated and time consuming. Given the special setting of our formulation, the following choice of node C guarantees optimal triangulation (Fig. 5):

> choose C so that the norm max(|AC|, |BC|) is minimized over all C that lie to the right of AB.

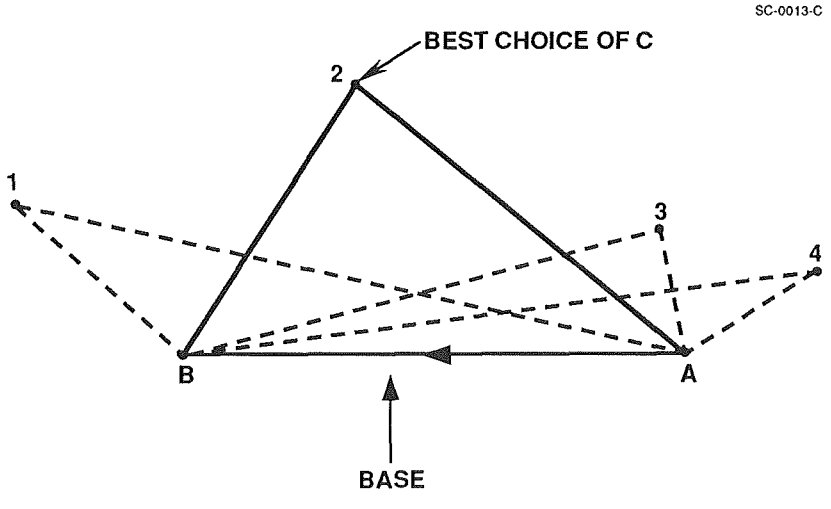

Fig. 5 Criterion for choosing the "best" third vertex C for triangle ABC.

Once the "best" C is selected, we must ensure that it indeed yields a triangle that will not overlap any other existing triangles. In practice, this is done by checking if any of the existing sides would partly or entirely lie within triangle ABC. If this happens, this choice of C is

marked and thrown away, and the same criterion is used to select a new C from the remaining available nodes. Once again, the fact that only a relatively small number of nodes are considered in the checking routine greatly reduces the computational time required in the search.

Note that this triangulation scheme is indeed a "greedy algorithm" in that it looks only one step ahead and only tries to create the best ***next*** triangle without weighing the impact of this choice on later choices. However, because connecting the nodes is very much a local process, there are typically only two or three good choices at each step, hence they cannot result in radically different triangulations. In fact, because of the regular spacing of the nodes lying on contour loops, only about 1% of the best C's get eliminated due to overlapping.

Once the current region of interest is triangulated, the program obtains the next set of nodes (step 3.3 above) which, in turn, will define the next region of interest. Steps 3.2 and 3.3 follow each other until the whole domain is covered. During the above triangulation process, information about neighboring elements and connected nodes is being stored as well, which could be useful input to some flow solvers. The entire triangulated domain is shown in Fig. 6.

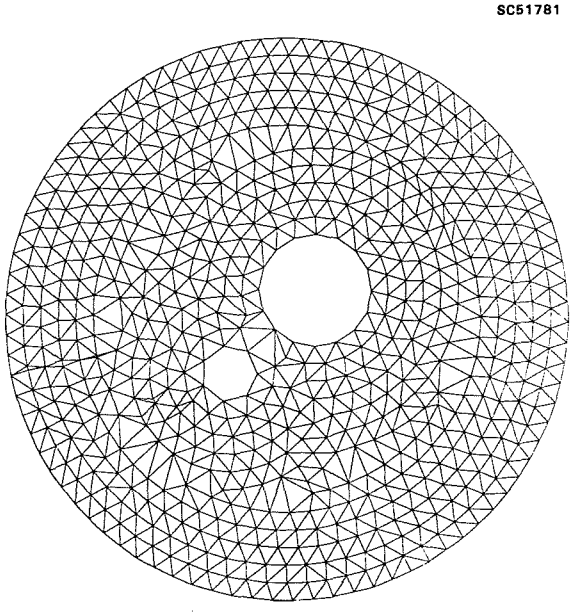

Fig. 6 The entire triangulated region before smoothing.

3.4 Smoothing

Once the whole domain is triangulated, to further regularize the elements, we apply a very simple and effective smoothing algorithm (suggested by Cavendish in Ref. 2). This process consists of replacing the coordinates of each node by the average of the coordinates of those surrounding nodes that it is connected to by a side. That is, each node is replaced by the centroid of the surrounding polygon. Since this algorithm follows the order in which the nodes were generated, at each step, the most updated coordinates are used. To accelerate convergence, we suggest that at each smoothing cycle the order is reversed, hence, propagating the smoothing in the opposite direction. Generally, two smoothing cycles result in triangles sufficiently close to a satisfactory set of nodes. The final, smoothed triangulated domain after two smoothing steps is shown in Fig. 7.

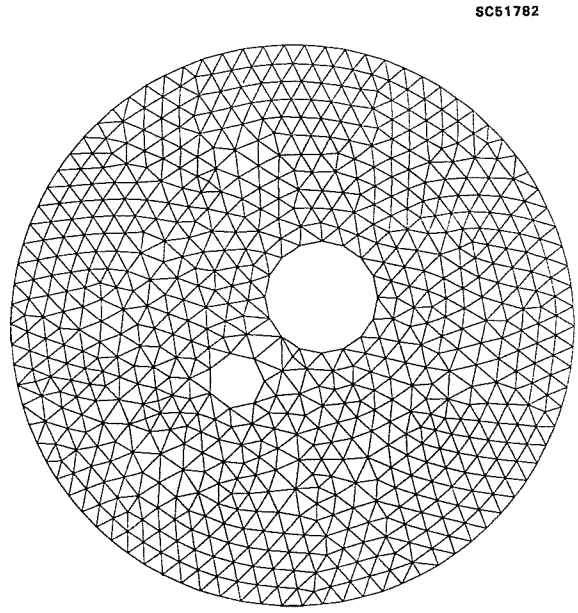

Fig. 7 Triangulated region after two cycles of smoothing.

4. CLUSTERING OF NODES NEAR BOUNDARY CURVES

To achieve varying nodal densities within the computational domain, we use the grid function described earlier. Modifying the formula used in Eq. (2.1), the grid function will not be a function linearly decreasing with the distance from the boundary curves, instead, an average of decreasing exponential functions of distances from each boundary curve. The boundaries are entered as patches, and each patch has a clustering factor c_i associated with it. The grid function at each point will then be divided by a weighted average of all the c_i's. That is:

$$\phi_{jk} = 1 \pm d((x,y);\ \gamma(0))/a_{jk} \tag{4.1}$$

where

$$a_{jk} = \frac{\sum_{i=1}^{n} c_i w^i_{jk}}{\sum_{i=1}^{n} w^i_{jk}} \tag{4.2}$$

and

$$w^i_{jk} = e^{-d((x_{jk},y_{jk});\gamma_i(0))\cdot b} \tag{4.3}$$

Here n is the total number of patches, ϕ_{jk} is the grid function at grid point (j,k), $\gamma_i(0)$ is the notation used for the ith patch representing the geometry or outer boundary, and b is a scaling factor dependent on the dimensions of the whole domain. It is clear from Eqs. (4.2) and (4.3) that the scaling used in Eq. (4.1) is very heavily dependent on geometry patches close to the point (j,k), while the influence of every other patch is relatively small and it diminishes

exponentially as the distance to that patch increases. Averages must be taken to ensure a smooth grid function so the contour algorithm will work. Figure 8 illustrates our initial example with the ratio of the largest and smallest cells being 10 (that is, $\max_{1\le i\le n} (c_i) / \min_{1\le i\le n} (c_i) = 10$) before smoothing, while the effect of smoothing (after two cycles) is shown in Fig. 9.

SC51783

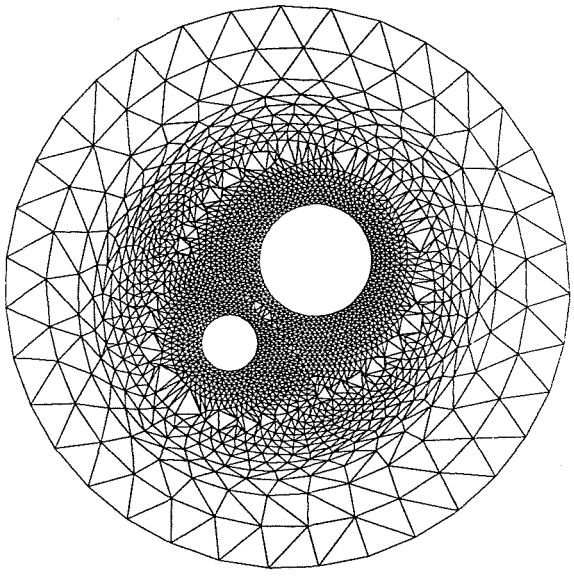

Fig. 8

Clustered triangular mesh around the two inner circles before smoothing.

SC51784

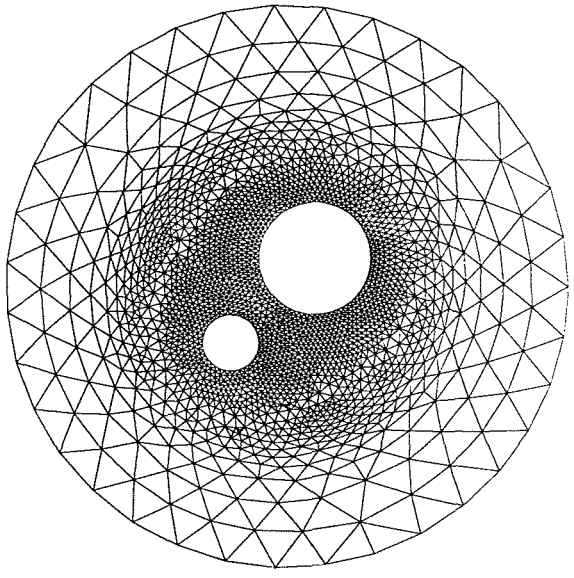

Fig. 9

Clustered mesh of Fig. 8 after two cycles of smoothing.

5. THE GENERAL PROGRAM

A flow chart containing the main building blocks of the program is now presented to illustrate the logic of the algorithm.

1. Input the geometry and outer boundary data, and clustering parameters for the patches.
2. Redistribute points along geometry and outer boundary loops according to required clustering.
3. Initialize grid function over computational domain
4. Until the whole domain is covered, do:

    4a. Find contour level for next grid function value; simultaneously generate nodes on the obtained loops.
    4b. Connect nodes in subregion enclosed by two sets of contour loops.

5. Do twice:

    5a. Smooth the entire region.

6. FURTHER EXAMPLES

The example used thus far is relatively simple; to illustrate how this method tackles very complex, multi-connected geometries, several examples of different characteristics follow.

Figure 10 shows the word "GRID" with equally spaced mesh around it. The same geometry is used in Fig. 11, where we have a clustered mesh instead, with the ratio of the largest and smallest elements being 10. Note the exponentially decaying influence of the clustering factor associated with each geometry segment as the distance to them increases throughout the domain.

SC51785

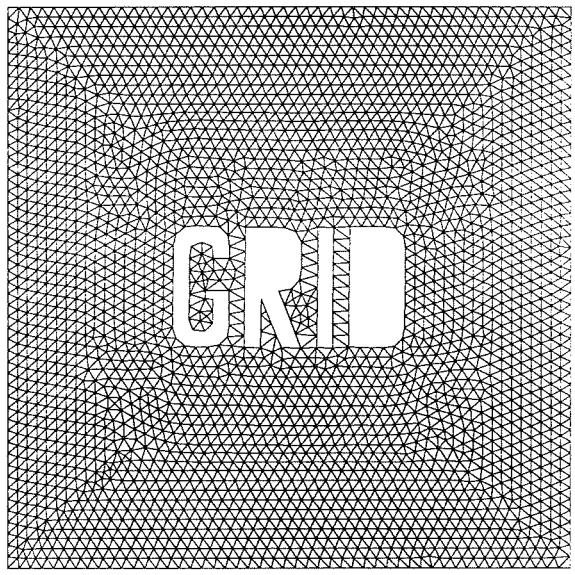

Fig. 10

Smoothed, equally spaced mesh around the word "GRID".

SC51786

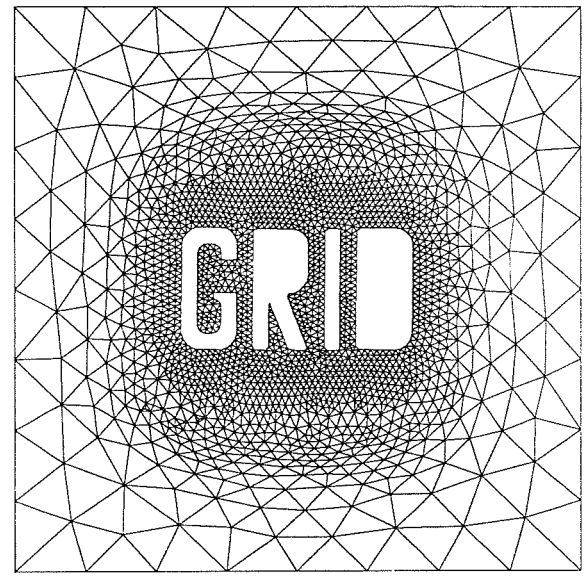

Fig. 11

Smoothed, clustered mesh around the word "GRID".

Figure 12 shows a grid for the "bomb-bay" problem, where heavy clustering is required in the cavity area and near the wall, while a quite course grid is sufficient as the far field is approached. Figure 13 shows a blowup of the cavity area.

7. CONCLUSIONS

A new, two-dimensional unstructured grid generator has been developed using the distance function to obtain the interior nodes for a prescribed, possibly multiconnected domain, where the nodal density can vary throughout the domain. The algorithm eliminates the need to break up the domain into several subdomains and triangulate each of those subdomains separately. Dense clustering of nodes is made possible with the density of the nodes varying smoothly.

The natural question arises as to whether this method could have an extension to three dimensions. The preliminary analysis given at the beginning of this paper convinces us that the theory of curves moving with curvature-dependent speed can be easily generalized to surfaces

SC51787

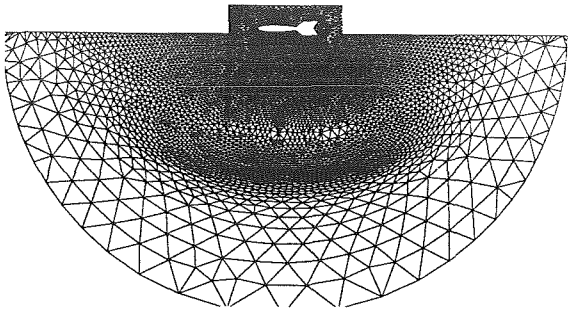

Fig. 12

Clustered mesh in the cavity area of the bomb-bay problem.

SC51788

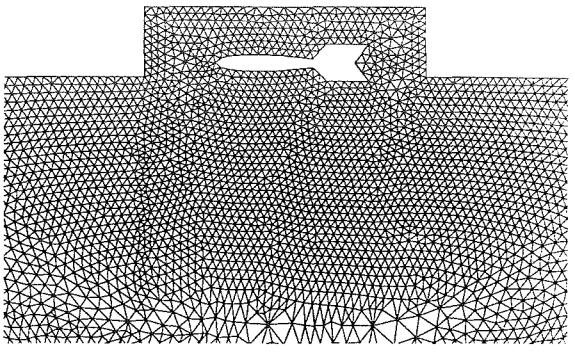

Fig. 13

Close-up of the bomb-bay mesh of Fig. 12.

in an n-dimensional space. Similarly, the proposition proved earlier also holds for any n-dimensional space; hence, the distance function can be used in the same way in three dimensions. Even obtaining the isosurfaces (which correspond to contour curves in two dimensions), representing the different grid levels, is relatively straightforward. However, to pick up the nodes on these isosurfaces according to a prescribed distribution function is the same problem as surface-gridding, which can be a complicated and time-consuming problem by itself. Once the nodes are known, forming the tetrahedral elements would be a process similar to that of triangulation described above, with search done on a level-by-level basis again. The same concepts would apply in three dimensions, with faces replacing sides, and intersection of planar sections replacing intersection of line segments. The criterion for choosing the best tetrahedron could be applied in its present form as well; we simply choose the fourth point of a tetrahedron to minimize the maximum of the three edges to be created. The three dimensional background grid, however, which has to resolve the smallest element, may have to be quite dense, thus creating a new, and perhaps unsurpassable bottleneck. Further results for extending the method to three dimensions will be reported in future work.

ACKNOWLEDGEMENTS

We would like to thank Stanley Osher, Kuo-Yen Szema and Chung-Lung Chen for their constructive comments and help in writing this paper.

REFERENCES

1. Barles, G., "Remarks on a Flame Propagation Model," Institut de Recherche en Informatique et Automatique (INRIA), Sophia Antipolis, France, Report No. 464, 1985.
2. Cavendish, J.C., "Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method," International Journal for Numerical Methods in Engineering, Vol. 8, 679-696 (1974).
3. Lo S.H., "A New Mesh Generation Scheme for Arbitrary Planar Domains," International Journal for Numerical Methods in Engineering, Vol. 21, 1403-1426 (1985).
4. Löhner, R., "Some Useful Data Structures for the Generation of Unstructured Grids," Communications in Applied Numerical Methods, Vol. 4, No. 1, 123-135 (1988).

5. Löhner, R. and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method," International Journal for Numerical Methods in Fluids, Vol. 8, No. 10, 1135-1149 (1988).
6. Osher, S. and Sethian, J.A., "Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations," Journal of Computational Physics, Vol. 79, No. 1, November 1988.
7. Yerry, M.A. and Shephard, M.S., "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique," International Journal for Numerical Methods in Engineering, Vol. 20, 1965-1990 (1984).