JOURNAL OF COMPUTATIONAL PHYSICS 11, 348–359 (1973)

A Simple Scheme for Generating General Curvilinear Grids

A. A. AMSDEN AND C. W. HIRT

University of California, Los Alamos Scientific Laboratory, Los Alamos, New Mexico 87544

Received March 29, 1972

An intuitively simple approach is presented for the computer generation of two-dimensional curvilinear grids suitable for finite difference solutions of problems in the field of continuum dynamics. An iterative process is employed to transform uniform networks of rectangular zones into more complex configurations. Ease of use and optimal adjustment are stressed, and numerous examples are given.

1. INTRODUCTION

Various techniques have been developed over the years for generating two-dimensional grids that are orthogonal in certain regions or in their entirety. Many of these methods are specifically oriented to the production of grids for the finite difference solutions of particular problems in continuum dynamics, and a number of investigators, such as Chu [1] have turned their attention to creating finite difference approximations for general grids.

Indeed, with the exception of very complicated boundary geometries, meshes in arbitrarily shaped regions are easily and rapidly constructed by the computer. Winslow has described a technique [2, 3] for creating a grid of triangular zones by direct numerical solution of Laplace equations. Other numerical methods have been described by Barfield [4, 5] for generating orthogonal or nearly orthogonal grids. The principal limitation with all of these, however, is the relatively rigid constrictions each places on the final results. For example, in some applications it is often desirable to enhance the grid resolution in selected regions. The technique described here offers a conceptually simple and logical approach to accomplish this, allowing the creation of a wide variety of computing grids or grid sections, through the use of elementary mathematics, letting the high-speed computer calculate the desired coordinates of the mesh vertices through an iterative process. Ease of use and optimal adjustment are emphasized, at the expense of orthogonality if necessary.

2. The Method

Let us begin by considering a network of rectangular zones. To simplify this discussion, we shall define the zones to be of uniform dimensions δx and δy throughout, although the concept is not limited by this restriction. The calculation of the desired grid, conforming to a specified boundary shape, proceeds by moving the vertices of the original grid in a sequence of small steps or iterations toward their final positions. The entire field of vertices is repeatedly swept until all vertices are being moved less than some small predetermined distance ϵ, at which time the field is said to have relaxed and the generation is complete.

In the logic scheme, each vertex is distinguished as being one of two possible types—boundary or interior. The boundary vertices are the crucial ones to move correctly, since the interior vertices are moved to adjust to the boundaries in a smooth fashion. Various prescriptions may be used to move the vertices, and it is often necessary to experiment before an optimum choice is obtained. It is, in fact, this flexibility of choice that we wish to emphasize.

We will study a number of specific examples, of increasing complexity, to demonstrate possible treatments of the boundary and interior vertices.

In the first example, the initial grid consists of square computing zones 20 wide by 10 high (i.e., 21 vertices by 11 vertices), the index i counting vertices horizontally and the index j counting vertices vertically. The zones have dimensions $\delta x = \delta y = 0.1$, and are shown in Fig. 1a. The problem is to deform this rectangular grid into the semicircle of Fig. 1b. Such a grid might be useful for calculating the slosh of a liquid in a round-bottomed tank, using, for example, the Arbitrary Lagrangian–Eulerian (ALE) computing method [6], which can handle curved boundary regions. The transition of the geometry from Fig. 1a to that of Fig. 1b can be accomplished if we move the boundary vertices on the left, bottom, and right radially inward, toward the center of the semicircle; that is, each boundary point at (x, y) is shifted during each iteration pass to

$$\begin{aligned} x_{\text{new}} &= x + \beta(x_0 - x), \\ y_{\text{new}} &= y + \beta(y_0 - y), \end{aligned} \tag{1}$$

where

$$\beta = \beta_0[(x - x_0)^2 + (y - y_0)^2 - r^2]. \tag{2}$$

Here x_0 and y_0 are the coordinates of the circle center (vertex 11, 11), and r is the desired circle radius. In this example, $x_0 = y_0 - r - 1.0$. The coefficient β_0 is chosen to restrict the magnitude of x and y changes to a reasonable value during a given iteration to reduce the possibility of vertices crossing one another. The vertices should move gradually and persistently to their final positions through the course of the iteration. The value of $\beta_0 = 0.01$ was chosen to limit x and y

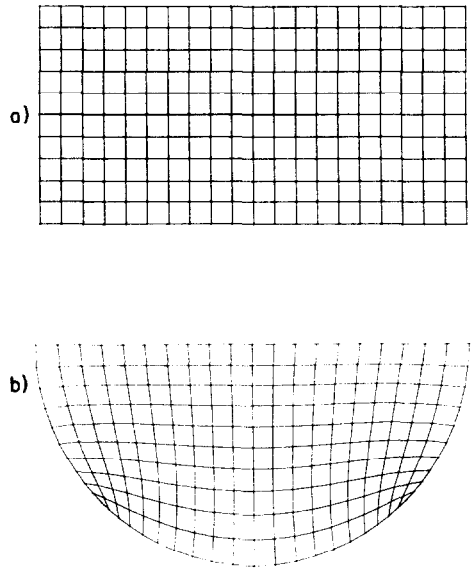

FIG. 1. (a) Initial 20×10 computing grid; (b) semicircle formed by iterative process.

changes to no more than approximately $\delta x/10$. All boundary vertices not on the top edge ($j = 11$) are moved by the above prescription, whereas the vertices at $j = 11$ are never moved. This leaves only the interior vertices to be considered, and in this example they are moved by the simple prescription of placing each at the average position of its eight closest neighbors,

$$\begin{aligned}(x_i^j)_{\text{new}} &= \tfrac{1}{8}(x_{i+1}^j + x_{i+1}^{j+1} + x_i^{j+1} + x_{i-1}^{j+1} + x_{i-1}^j + x_{i-1}^{j-1} + x_i^{j-1} + x_{i+1}^{j-1}),\\ (y_i^j)_{\text{new}} &= \tfrac{1}{8}(y_{i+1}^j + y_{i+1}^{j+1} + y_i^{j+1} + y_{i-1}^{j+1} + y_{i-1}^j + y_{i-1}^{j-1} + y_i^{j-1} + y_{i+1}^{j-1}),\end{aligned} \tag{3}$$

where, for simplicity, the most updated values of x and y available are used on the right side of these equations. Thus, we now have a prescription defined for every vertex, and it is an easy matter to construct a logic path for the computer to follow. The iterative sweeping of all vertices is performed until no boundary or interior vertex has an x or y change greater than ϵ (we have chosen $\epsilon = 10^{-4}$). The transition of the grid of Fig. 1a to that of Fig. 1b required 291 iterations, a task involving only a few seconds of computer time on the CDC 6600, and very little programming.

It was found on this example that the number of iterations required for relaxation was at a minimum for β_0 values in the range 0.10–0.92. The number dropped rapidly as β_0 was increased from 0.01 to 0.08, where only 75 iterations were required. For β_0 values up to 0.94, the number of iterations required was almost

constant, with a minimum of 67 for β_0 values of 0.90–0.92. Above $\beta_0 = 0.92$, however, the number of iterations again rose quite rapidly, with 227 required for $\beta_0 = 0.98$, and with $\beta_0 = 1.0$, the solution never relaxed. This experimentation with various values of β_0 was tried only with this first example, but it is evident that it is worthwhile to find an optimum β_0 when similar grids are to be generated repeatedly.

It should be noted that it is generally not necessary to relax the boundary vertices to their final positions insofar as grid generation itself is concerned (see, for example, Ref. [2]). A similar calculation was made in which the boundary vertices were simply placed in their final semicircular configuration, and the interior points were forced to relax to this boundary using Eqs. (3). In the early stages the grid points were strongly crossed, but they eventually uncrossed and the convergence was as fast as the optimum-β_0 run above, requiring only 68 iterations using the same $\epsilon = 10^{-4}$ convergence criterion.

If desired, the possibility of crossover can be eliminated by always expanding the grid outward to form the desired curvilinear grid, then rescaling the final set of coordinates to the actual zone size desired.

The reason we wish to emphasize the technique of relaxation of the boundary points is that in many cases it is preferable to generate the curvilinear grid within and during a continuum dynamics calculation. Thus, one iteration pass of the grid relaxation is performed during each time step of the calculation, and effectively provides a rezoning technique. (In a grid qualitatively like that of Fig. 1, this may be useful for calculating spherically diverging shocks.) To allow even slight crossing of mesh lines would be detrimental or catastrophic to the results if it occurs during a calculation. Moreover, the expansion-and-rescaling technique to eliminate crossover is not appropriate if a calculation is being performed simultaneously.

In combining the grid generation as a rezoning technique within the continuum dynamics calculation, it was seen that some overrelaxation of the interior vertices was required to keep them moving inward ahead of the exceptionally fast motion of the edge vertices. Even with a β_0 of only 0.01, the grid will virtually reach its final configuration in only 25 iterations, and optimum speed of grid generation may cease to be a desired goal.

If desired, the area of the final semicircle can be made equal to the original grid area by an appropriate choice of δx and δy in the original rectangle. For example, to obtain a final semicircle radius of 1, with the circle center again at vertex (11,11), the initial δx and δy of the rectangle are seen to be 0.0886, and the initial coordinates of vertex (1,1), thus, lie at (0.1138, 0.1138) rather than at (0.0, 0.0). In the iteration the grid vertices will automatically move inward or outward as required. The vertices along the top boundary, other than the corners, should be evenly spaced in x between neighbors on the left and right. The resulting specified-area grid has the same qualitative appearance as that of Fig. 1b, and required 286

iterations to generate with $\beta_0 = 0.01$. The reduced number of iterations from the earlier example, using the same β_0, may be attributed to the fact that vertices do not have to be moved as far.

The choice of interior vertex positioning as governed by Eqs. (3) is roughly equivalent to making the grid locally orthogonal. More accurate formulas are described by Winslow [2, 3]. However, it is the purpose of this paper to indicate how other choices for grid generation can be realized as well.

In the examples discussed thus far, the approximate orthogonal spacing of interior vertices has been sufficient, but the approach will handle any arbitrary bias or weighting toward a particular direction that one desires, giving more flexibility in this regard than the usual orthogonal approach. For example, unequal weighting of the four principal neighbors is often useful, as illustrated in the examples of Fig. 2. Figure 2a shows the initial notched grid of equal rectangular cells. The problem is to move the boundary vertices lying in the notch into a semicircular configuration. Remaining boundary vertices not in the notch are moved to the average position of their two boundary neighbors on either side, while the four outside corners are held forever fixed. Figure 2b shows the resulting grid when all eight neighbors are weighted equally for each interior vertex, according to Eqs. (3). The interior vertices can be drawn closer to the semicircle with a weighting (W) for each neighbor, proportional to its distance from the semicircle. A possible form is

$$W_N = 1 + [\alpha/(1 + 20d^2)],$$

where

$$d = [(x_N - x_0)^2 + (y_N - y_0)^2 - r^2]/r^2,$$

α is a constant, and the subscript N denotes the vertex label of a particular neighbor. The set of resulting weights are then used to define new interior vertex positions to replace Eqs. (3),

$$x_i^j = \sum_1^8 W_N x_N \Big/ \sum_1^8 W_N$$

$$y_i^j = \sum_1^8 W_N y_N \Big/ \sum_1^8 W_N .$$

Figure 2c shows the grid when $\alpha = 1$, while the extreme deformation of Fig. 2d is obtained with $\alpha = 3$. The solutions required 144, 168, and 225 iterations for Figs. 2b, 2c, and 2d, respectively, with the convergence criterion $\beta_0 = 0.01$.

A grid similar to that in Fig. 2c was successfully used in several ALE-method calculations of flows about plane cylinders and spheres. The weighting treatment was necessary to increase grid resolution near the curved boundary for the accurate calculation of boundary layer formation in this region.

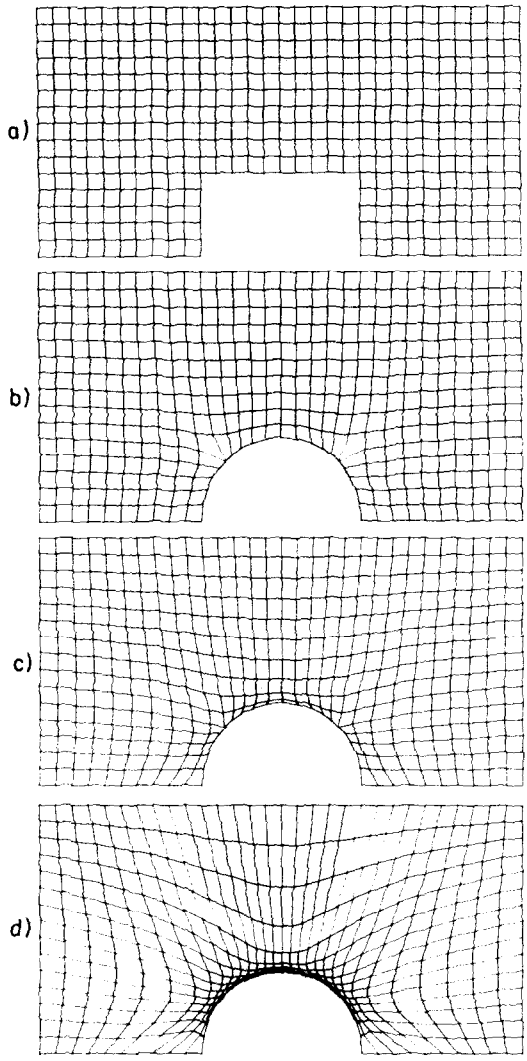

FIG. 2. (a) Initial grid; (b), (c), (d) are resultant semicircles with various amount of weighting of interior vertices.

Another possible mesh for such calculations has the horizontal grid lines run up over the cylinder, rather than intersecting it, thus more closely following streamlines of the flow. A grid of this type can be formed by starting with the full rectangular grid as in Fig. 3a and "pushing" a portion of the boundary inward to form the semicircle, where some number (K) of vertices along the bottom are

chosen to define the semicircle. In this example, the angle θ at the circle center (x_0, y_0) subtended by neighboring pairs of vertices along the semicircle is constant and is given by $\theta = \pi/(K - 1)$. Both θ and the coordinates of the boundary vertices to be pushed in are easily specified; the k-th point around the semicircle will have final coordinates

$$\begin{aligned} x_k &= x_0 - r\cos[(k-1)\,\theta], \\ y_k &= y_0 + r\sin[(k-1)\,\theta], \end{aligned} \tag{4}$$

where r is again the semicircle radius. A vertex point (i, j) to be pushed in to the k-th circle point can be moved by the simple relaxation equations

$$\begin{aligned} x_i^j &= x_i^j - \beta_0(x_i^j - x_k), \\ y_i^j &= y_i^j - \beta_0(y_i^j - y_k). \end{aligned} \tag{5}$$

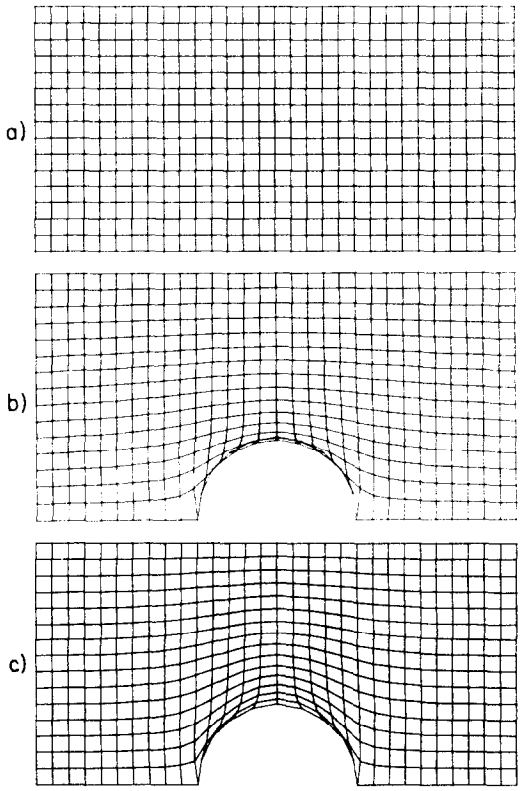

FIG. 3. (a) Initial grid (b) grid; pushed in to form semicircle; (c) crossover eliminated by unequal weighting.

When interior vertices and other boundary vertices are moved to average positions among adjacent eight or two neighbors, respectively, the resulting grid is shown in Fig. 3b. A total of 342 iterations were necessary to obtain this configuration, with $\beta_0 = 0.02$. However, moving interior vertices to average positions of four neighbors instead of eight gives virtually identical results, and is faster. One of the problems of the pushing-in technique is evident in Fig. 3b—the semicircle is properly formed, but the $j = 2$ line has crossed over and lies below the semicircle formed from the $j = 1$ line. This results from the logic chosen for moving the interior vertices and not from the fineness of the iteration. (A run with β_0 reduced to 0.004 gave identical results after 1322 iterations.) To prevent crossover the treatment of the interior vertices must be modified. The more accurate equipotential method of Winslow [2, 3] is one method that would prevent this crossover. Alternatively, it may be noted that crossover really is manifested in the y direction, and evidently an equal weighting of neighbors for a new y coordinate does not

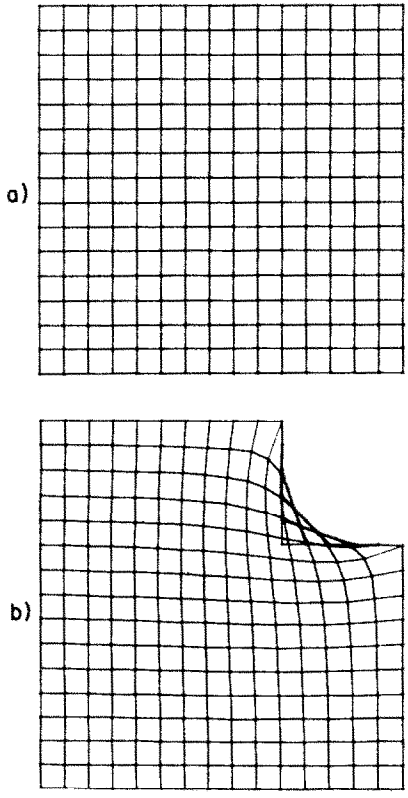

FIG. 4. (a) Initial grid; (b) corner pushed in, exhibiting crossover of interior vertices.

give a proper balance. The validity of this argument is shown in Fig. 3c where crossover has been eliminated by making 80% of y-coordinate weighting dependent on the two principal vertical neighbors, and only 20% on the two lateral neighbors,

$$y_i^j = 0.4(y_i^{j+1} + y_i^{j-1}) + 0.1(y_{i+1}^j + y_{i-1}^j).$$

The interior x weighting remained equal for the four neighbors. The grid of Fig. 3c also required 342 iterations to create.

Thus, it is seen that a judicious choice of interior vertex weights is crucial in strongly concave regions if crossover is to be eliminated in the final configuration. Usually, unequal weighting is helpful, and the strongest "springs" should pull normal to the concave boundary, as approximated in the above 80% and 20% equation.

Despite its usefulness, the pushing-in technique cannot be used in all instances. An extreme example is shown in Fig. 4. The upper right corner of the grid in Fig. 4a was pushed in to form a notch (Fig. 4b). As in the previous examples, final boundary point coordinates were defined *a priori*, with the vertical boundary points on the right side laid over flat, and the points along the top turned down vertically. The crossover is not incurable, but the zoning around the notch would be highly irregular regardless of the treatment of the interior points in that region, and consequently of little value as a finite difference grid. Generally, such a notched shape would be better formed from an initially notched rectangular region.

The general iterative scheme is ideally suited to the formation of a grid for computing bifurcated flows. One possible technique makes use of a grid with a row of zones of zero height along the line of bifurcation. The initial grid shown in Fig. 5a is actually 11 zones high rather than the 10 zones it appears to be. Here, the y coordinates of the $j = 7$ grid line are initially identical to the y coordinates of the $j = 6$ grid line, and the mesh is split along this central line into the configuration shown in Fig. 5b.

The desired final coordinates of only the four corner vertices on the right side, (19, 1), (19, 6), (19, 7), and (19, 12), need to be known and specified beforehand. These four vertices are then iterated to their final positions, the two lower ones moving downward, and the two upper ones moving upward. Other boundary vertices that will lie at corners of the final grid (1, 1), (7, 1), (9, 6), (9, 7), (1, 12), and (7, 12), and also vertices on the $j = 6$ and $j = 7$ grid lines out to $i = 9$, are never moved. Remaining boundary vertices are then moved to the average positions of their appropriate two neighbors, and interior vertices to the average of their eight neighbors. Vertical grid lines in the central zones ($j = 6$) stretched in the bifurcation have been deleted from the plot. In a finite difference calculation using this grid, the $j = 6$ vertices on the left eight zones must be identified with

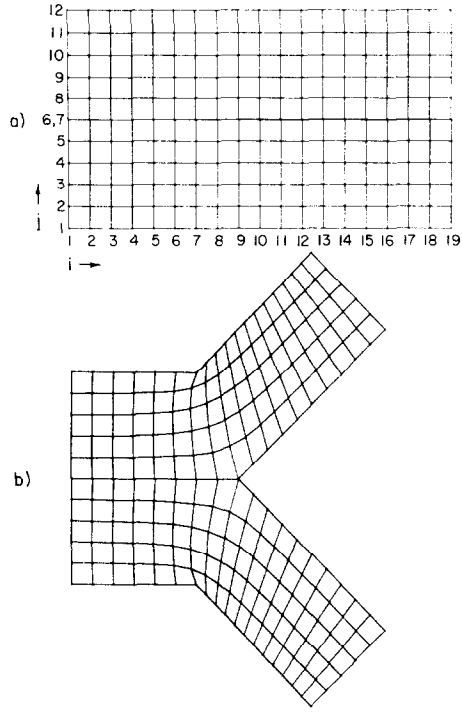

FIG. 5. (a) Initial 18 × 11 grid, where $j = 6$ row has zero height; (b) bifurcated flow grid formed by iterative process.

the $j = 7$ vertices above. The grid in Fig. 5b required 360 iterations to create, with $\beta_0 = 0.02$, $\epsilon = 10^{-4}$, and $\delta x = \delta y = 0.2$.

The generation of the grid in Fig. 6b from the grid of Fig. 6a followed the same general concepts as the previous example. Here, however, we specified the final coordinates of all 16 boundary vertices forming each of the four curvilinear boundaries. These were calculated by the computer from specified radii (1.4 and 2.4) and circle centers (1.0, 0.0) and (1.0, 4.8). Remaining vertices were averaged as in the previous example. This 90° circular-turnout grid required 386 iterations in its generation.

These are but a few examples of rectangular grids that have been distorted into a variety of different shapes, requiring only elementary algebra and trigonometry on the part of the user. At times some experimentation must be used to arrive at optimum grids if orthogonality is not a suitable criterion. Rather than being a disadvantage, it is this feature that makes the present technique particularly useful, as for example, in the case of Fig. 2 where finer zoning was desired near the curved

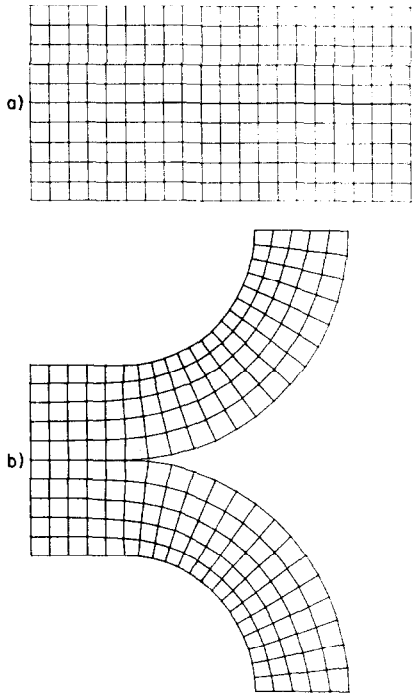

FIG. 6. (a) Initial 20 × 11 grid, where $j = 6$ row has zero height; (b) bifurcated flow grid formed by iterative process.

boundary. The possibilities of these simple, intuitive, iterative techniques for the generation of complex grids are unlimited, since the boundary and interior vertices may be moved by any reasonable prescriptions that produce the desired grid structure.

ACKNOWLEDGMENT

The authors acknowledge the assistance of F. H. Harlow and H. M. Ruppel for their helpful discussions and advice on the manuscript. This work was performed under the auspices of the United States Atomic Energy Commission.

REFERENCES

1. W. H. CHU, Development of a general finite difference approximation for a general domain. Part 1: Machine transformation, *J. Comp. Phys.* **8** (1971), 392.
2. A. M. WINSLOW, " 'Equipotential' Zoning of Two-Dimensional Meshes," Lawrence Radiation Laboratory, Report UCRL-7312, 1963.

3. A. M. WINSLOW, Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh, *J. Comp. Phys.* **2** (1967), 149.
4. W. D. BARFIELD, Numerical method for generating orthogonal curvilinear meshes, *J. Comp. Phys.* **5** (1970), 23.
5. W. D. BARFIELD, An optimal mesh generator for Lagrangian hydrodynamic calculations in two space dimensions, *J. Comp. Phys.* **6** (1970), 417.
6. C. W. HIRT AND A. A. AMSDEN, An arbitrary Lagrangian-Eulerian computing method for all flow speeds, *J. Comp. Phys.*, to be published.