JOURNAL OF COMPUTATIONAL PHYSICS 65, 370–385 (1986)

Blending Method for Grid Generation

JOHN STEINHOFF

Department of Engineering Science and Mechanics, The University of Tennessee Space Institute, Tullahoma, Tennessee 37388

Received April 22, 1985

A systematic procedure is presented for synthesizing a complex computational grid out of a number of simpler "elementary" grids. This method is useful when a grid is required for a region which, though complex, consists of a number of simpler sub-regions. Frequently, in such cases, validated grid generation methods already exist for the sub-regions, such as the individual lifting surfaces of an airplane. The procedure presented allows a smooth complex grid to be generated which becomes exactly equal to each elementary grid as the surface corresponding to that elementary grid is approached. In this way, the existing generation methods do not have to be changed and can be used as "black boxes," whether they are algebraic, partial differential equation based, or just given numerically. A number of examples are described in detail.

1. INTRODUCTION

In many cases where a smooth computational grid is required, the boundary of the computational domain can be decomposed into a number of pieces, each of which is fairly simple. We suppose that an adequate grid can be easily generated for each of these pieces, if considered by itself, and describe a method for blending these "elementary" grids into one smooth composite grid which has all of the pieces as its boundary. Examples where this technique can be used include external flow over an entire aircraft, where simple methods exist for generating grids individually over each of the lifting surfaces and the pieces of the body. Other examples include internal flows where a number of ducts or tubes join, and methods exist for generating grids for each element taken separately. An important feature of the concept is that it can be used recursively. Composite subgrids can first be formed from elementary grids, using the method, then, the same method can be used to form larger composite grids out of these individual subgrids. If algebraic methods are used to form each elementary grid, which can often be done since each piece is simple, then the entire grid generation procedure is algebraic, since the blending is non-iterative and involves no partial differential equation solutions. Accordingly, where applicable, it is a fast method suitable for interactive use. Also, if a partial differential equation is to be solved for some physical quantity and an iterative method is used to solve a set of discrete equations on the grid, which is usually the case, then at each iteration the grid can be quickly regenerated and there is no need to store the entire grid

0021-9991/86 $3.00

system. This feature can be especially important for large 3-dimensional problems. This method is very different from other algebraic methods, such as those of Eiseman [1]. Each elementary grid is taken to be previously determined, either by algebraic methods, partial differential equation solution [2], or any other means. These grids can be defined over the entire space, rather than just on surfaces as in "transfinite interpolation" schemes.

An important feature of the method is that it allows the grid designer to use software packages and methods already developed or being developed by others (which can be quite sophisticated and complex) for the elementary grids about each piece of the problem. These can be used as "black boxes," and after each elementary grid is generated the grid designer can blend them together. Also, after a composite, complex grid is generated, if one of the pieces is later modified, only the single new elementary grid need be recomputed and blended into the composite grid.

In this paper two types of problems will be treated. In the first, the elementary pieces of the boundary are physically separated, and in the second they are contiguous. The use of the method will be illustrated with several representative 2-dimensional examples. There is no conceptual difference between 2- and 3-dimensional formulations and results of current work on 3-dimensional grids will be presented in a subsequent paper.

Since the method is local, and each piece only influences the grid in its vicinity, local methods of controlling the grid can be formulated. This could be required, for example, if resolution were inadequate or if grid lines were to cross. Some of these methods will be described. It will be seen that advantages of the method include simplicity and speed, even for complex geometries. Disadvantages include the lack of guarantees against line crossing (although this can be made unlikely) and the requirement that each elementary grid locally have the same topology.

2. The Basic Method

Consider a set of N grids, each spanning the same computational space and approximately the same physical space. For simplicity, we define the computational coordinates to be just the (integer) indices of the grids. Thus, in n dimensions we have an n component vector, $\mathbf{r}_m(\mathbf{I})$ ($\equiv (x_m(\mathbf{I}), y_m(\mathbf{I}), z_m(\mathbf{I}))$ for $n=3$) defined on each grid (labeled m) as a function of the indices $\mathbf{I}$ ($\equiv (i, j, k)$ for $n=3$). It is important to think of the n components of $\mathbf{r}_m$ as ordinary smooth functions defined in the computational ($\mathbf{I}$) space. Defining non-negative weighting functions $P^m(\mathbf{I})$, the physical coordinates of the composite grid are then simply weighted sums of those of the elementary grids:

$$\mathbf{r}_c(\mathbf{I}) = \left[\sum_m P^m(\mathbf{I})\,\mathbf{r}_m(\mathbf{I})\right] \Big/ \left[\sum_m P^m(\mathbf{I})\right].$$

The weighting functions are, in general, functions of all of the indices $\mathbf{I}$, and are a function of how close the point $\mathbf{I}$ is to the elementary surface segments. When $\mathbf{I}$

approaches some surface segment, say m_1, then $P^{m_1}(\mathbf{l})$ must approach 1 and all the other P's must approach 0 since there we must have

$$\mathbf{r}_c(\mathbf{l}) \rightarrow \mathbf{r}_{m_1}(\mathbf{l}).$$

Some of the "art" of using the method resides in the determination of the functions $P^m(\mathbf{l})$. Since values of $\mathbf{r}_m(\mathbf{l})$ which define smooth grids are determined separately about each elementary surface, the $P^m(\mathbf{l})$ do not have to do as much work as in an interpolation method where they typically completely determine one of the coordinates. In the examples to be presented in the next sections, it will be seen that very simple functions are sufficient. The main problems arise when grids must be blended with very different values of $\mathbf{r}$ in certain regions of $\mathbf{l}$ near an elementary surface. Then, care must be taken that a number of derivatives of $P^m(\mathbf{l})$ are 0 as $\mathbf{l}$ approaches the elementary surface (m_1), in addition to the value of $P^{m_1}(\mathbf{l})$ approaching 1. As more derivatives are made to go to 0, the region in $\mathbf{l}$ space, where $\mathbf{r}_c(\mathbf{l})$ approaches $\mathbf{r}_{m_1}$ becomes larger.

3. Example 1—Cascade "C" Transformation

This simple example involves a single weighting function. The two surfaces to be fitted by the computational grid are the aifoil surface, where normal velocity is set to zero, and the outer surface, where periodic conditions are imposed at the sides and far field conditions at the ends. A transonic potential flow solution was to be computed on the grid using a multigrid algorithm[1] [3].

First, a vertical shearing is used to approximately straighten the airfoil. After the grid is generated this shearing will be applied in reverse to all the grid points so that the initial airfoil is recovered. The shearing function (of x) is a straight line in front of the leading edge and behind the trailing edge, matching the slope and position of the mean camber line there, and is an interpolating cubic function of x in between. This function is simply subtracted from the initial airfoil coordinates and, after the mappings are complete, added back to each of the grid points to generate the final grid. After the initial shearing, a "C" mesh is generated about the airfoil (Fig. 1). (The open trailing edge is a continuation of the initially rounded trailing edge and is designed to simulate viscous effects.) This mapping involves a square root transformation about a point inside the leading edge region and a shearing. It is a standard mapping for aircraft airfoils and is described in detail in [4]. This is the first grid. It has good properties near the airfoil surface but obviously is not suitable in the outer region for imposing periodic boundary conditions.

The second grid consists of a long Cartesian grid with parallel top and bottom boundaries, capped with a semicircular piece (see Fig. 2). It has the same C mesh

[1] The development of the computer code for the cascade solution was supported by NASA Lewis Research Center Grant NASA NAG 3-398.

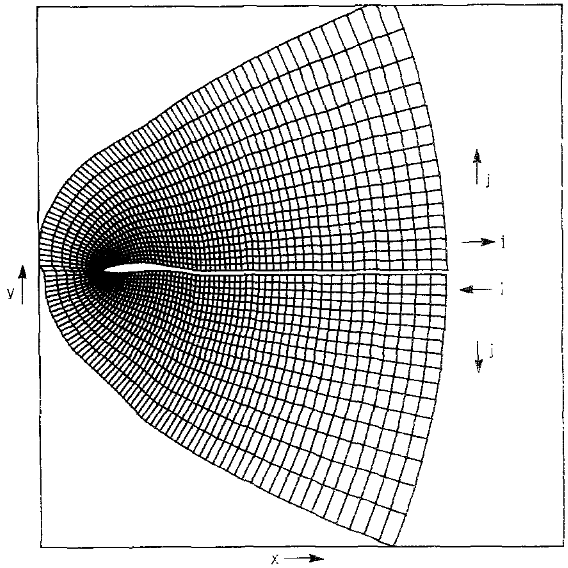

FIG. 1. Inner grid for sheared cascade airfoil.

topology as the grid in Fig. 1, but is ideally suited for imposing periodic boundary conditions on the top and bottom segments and far-field conditions at the ends. The internal grid lines join the upper and lower boundaries orthogonally, as required if the grid is to be smooth when continued periodically (even when the shearing function is added back). The only problem with the grid is that there is no airfoil.

Our objective is to compute a grid that approaches grid 1 along one line ($j = 1$), and grid 2 along the other three ($j = j_{\max}$, $i = 1$ and $i = i_{\max}$) (see Fig. 3). Since there

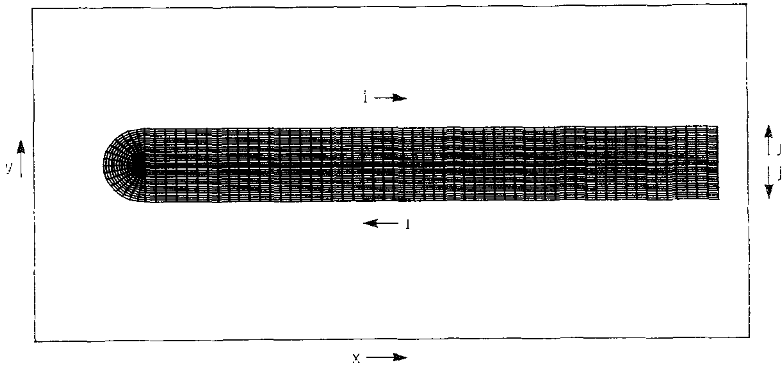

FIG. 2. Outer grid for sheared cascade airfoil.

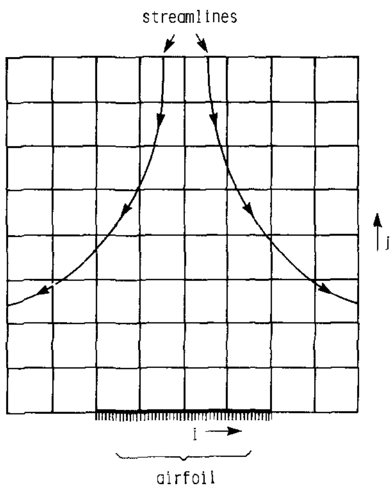

FIG. 3. Computational grid for cascade.

are only two elementary grids, we have here a simple form for $\mathbf{r}_c(\mathbf{l})$ with only a single weighting function $p(\mathbf{l})$:

$$\mathbf{r}_c(\mathbf{l}) = p(\mathbf{l})\,\mathbf{r}_1(\mathbf{l}) + (1 - p(\mathbf{l}))\,\mathbf{r}_2(\mathbf{l}).$$

The constraints on $p(\mathbf{l})$ are:

1. $p(\mathbf{l}) \to 1$ as $j \to 1$, i not close to 1 or $i_{\max}$.
2. $p(\mathbf{l}) \to 0$ as $j \to j_{\max}$, or $i \to 1$ or $i \to i_{\max}$.

The main problem here concerns the points near the leading edge of the airfoil for j near 1. If p is not very close to 1 for $j = 2, 3, \ldots$ then the (distant) points from grid 2 will be significantly included and the final $\mathbf{r}_c$ values will be very different from

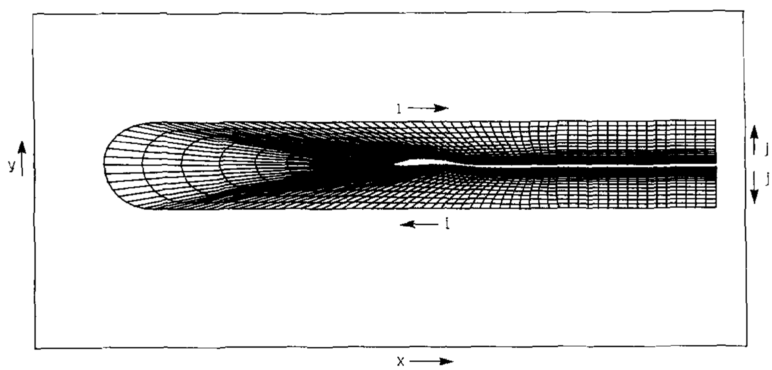

FIG. 4. Blended cascade grid.

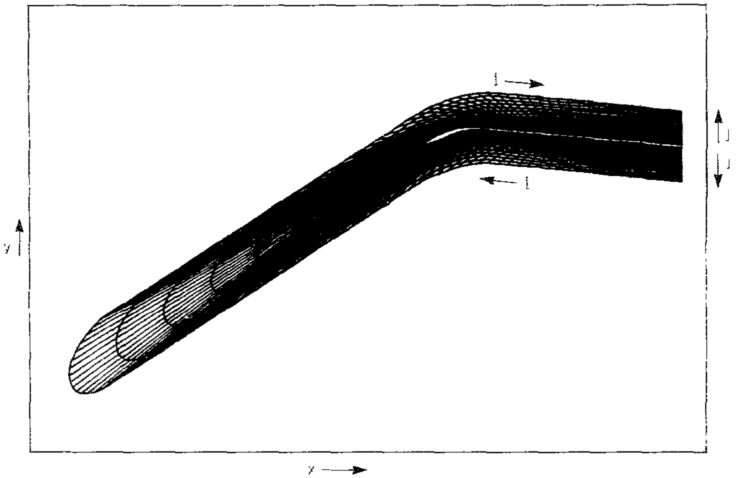

FIG. 5. Final cascade grid without shearing.

those for $j = 1$ (where p is exactly 1). There will thus be a large grid spacing between points with $j = 1$ and $j = 2$, as well as between $j = 2$ and $j = 3$, etc. Accordingly, we choose a function with several vanishing derivatives at $j = 1$:

$$p(\mathbf{I}) = [1 - a(j)]\, b(i)$$

$$a(j) = \alpha^2\, \tfrac{1}{2}[1 - \cos(\pi\alpha)]$$

$$b(i) = \tfrac{1}{2}[1 - \cos(\pi\beta)]$$

$$\alpha(j) = (j - 1)/\Delta$$

$$\beta(i) = \min(\Delta, i - 1, i_{\max} - i)/\Delta$$

where Δ is a length scale, set equal to $(j_{\max} - 1)$.

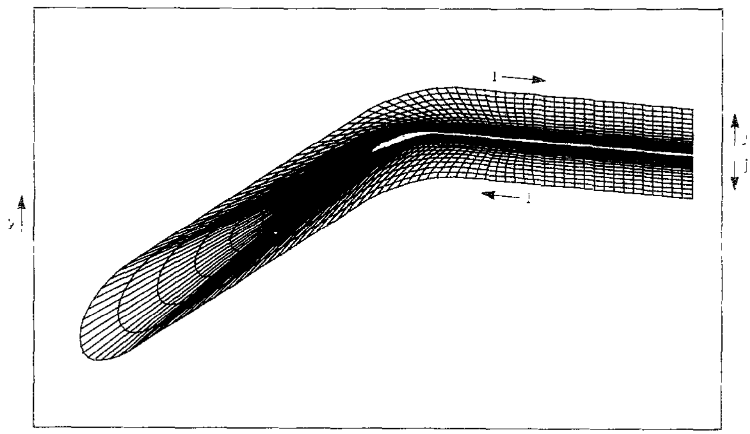

FIG. 6. Coarse cascade grid.

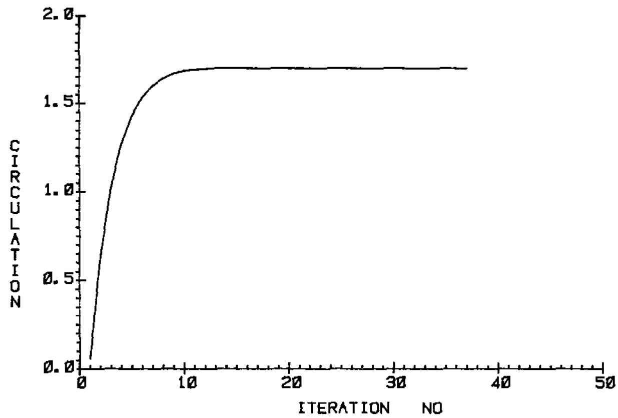

FIG. 7. Circulation development for cascade solution.

The resultant grid is depicted in Fig. 4 and in Fig. 5 with additional stretching in the x direction and the shearing function added. A coarser grid with $\frac{3}{4}$ the number of cells in each direction is presented in Fig. 6 for clarity. The convergence of our finite volume multigrid method for a transonic shock-free case is presented in Fig. 7 for circulation development and Fig. 8 for average residual decay (one fine grid (128×16) iteration per multigrid cycle was used with a total of 5 grids). Besides cascades, this mapping technique would obviously be useful for wind tunnel boundary conditions.

4. Example 2—Wing–Canard

As in the last example, there are "elementary" boundaries which are separated in both computational and physical space. Here, we choose an "H" grid elementary

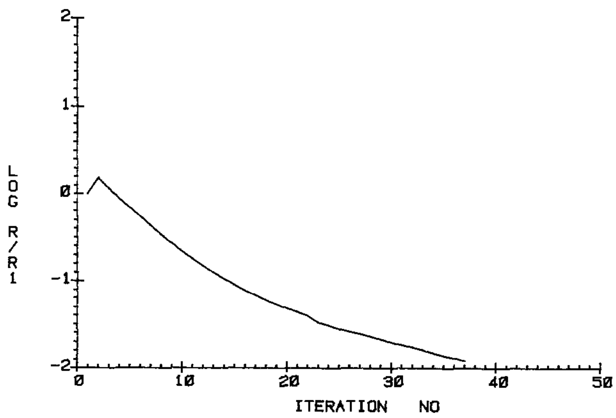

FIG. 8. Residual decay for cascade solution.

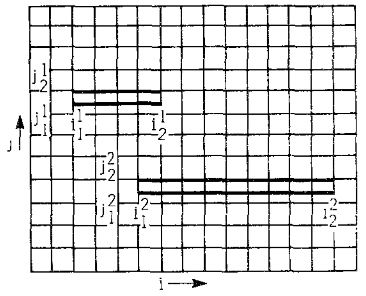

FIG. 9. Computational wing/canard grid.

mapping for both the canard and wing. A detailed study of this mapping was presented in [5] for a single airfoil, where it was shown that a particular transformation can be used to eliminate the singularity which normally arises at the leading edge in this case. A compressible flow problem was solved on this grid and the solution was shown to be accurate once this singularity was removed.

The objective here is to map the wing-canard and outer boundary to a computational grid depicted in Fig. 9, using an elementary H mesh for the canard depicted in Fig. 10 and for the wing in Fig. 11. In this figure, the canard is at zero relative angle of attack. For non-zero relative angle of attack, the entire elementary canard grid is just rotated in physical space before blending.

In this case there are four starting grids: an "outer" Cartesian one associated with the outer boundaries, a wing and a canard grid, and an inner Cartesian grid.

The basic plan in this case is to generate a wing/canard inner grid with fairly uniform grid size (except near the wing and canard), and then to blend this with an

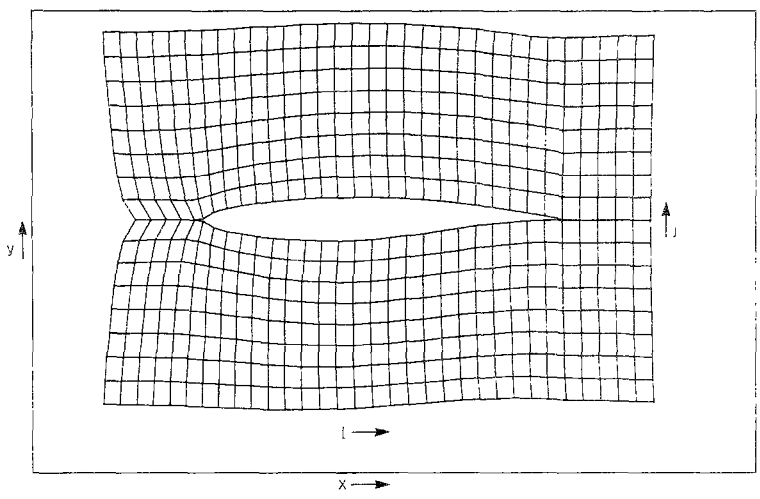

FIG. 10. Elementary grid for canard airfoil.

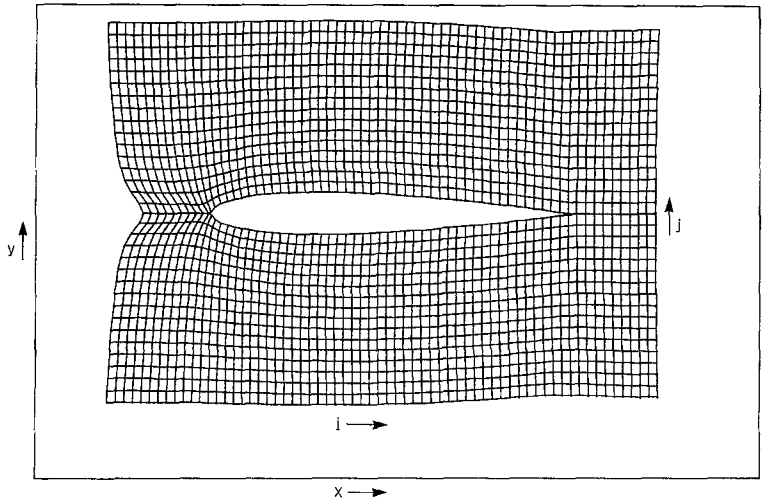

FIG. 11. Elementary grid for main airfoil.

"outer" grid with much larger spacing to develop far-field stretching. The elementary canard grid is labeled $m=1$, and the elementary wing grid, $m=2$. These are first blended with a fine "inner" Cartesian grid ($m=3$) to get intermediate composite grids (labeled 13, 23). These two are then blended to get an inner wing/canard grid, labeled 123. Finally, to provide far field stretching, this grid is blended with an elementary Cartesian grid (label 4) which has much larger grid spacing.

First, an "inner" canard (wing) grid is computed by blending the canard (wing) and inner Cartesian grid. The first blendings (13, 23) are done with a weighting function

$$p^m(\mathbf{l}) = \tfrac{1}{2}[1-\cos(\pi\alpha^m)]\,\tfrac{1}{2}[1-\cos(\pi\beta^m)]$$

where $m=1$ for canard and 2 for wing, and

$$\begin{aligned}
&\alpha^m(i)=0, && i \leqslant i_0^m\\
&\alpha^m(i)=(i-i_0^m)/(i_1^m-i_0^m), && i_0^m < i < i_1^m\\
&\alpha^m(i)=1, && i_1^m \leqslant i \leqslant i_2^m\\
&\alpha^m(i)=(i_3^m-i)/(i_3^m-i_2^m), && i_2^m < i < i_3^m\\
&\alpha^m(i)=0, && i \geqslant i_3^m
\end{aligned}$$

The function $\beta^m(j)$ is defined in the same way, with $i_k^m \leftrightarrow j_k^m$, $k=0, 1, 2, 3$. Then, for the inner (composite) canard and wing grids ($\mathbf{r}_{13}(\mathbf{l})$, $\mathbf{r}_{23}(\mathbf{l})$),

$$\mathbf{r}_{m3}\mathbf{l} = p^m(\mathbf{l})\,\mathbf{r}_m(\mathbf{l}) + [1-p^m(\mathbf{l})]\,\mathbf{r}_3(\mathbf{l})$$

where $\mathbf{r}_1(\mathbf{l})$, $\mathbf{r}_2(\mathbf{l})$, and $\mathbf{r}_3(\mathbf{l})$ are the canard, wing, and inner Cartesian grids, respectively. In the grids $\mathbf{r}_{13}(\mathbf{l})$ and $\mathbf{r}_{23}(\mathbf{l})$ the canard or wing lies in the region

$$i_1^m \leqslant i \leqslant i_2^m; \qquad j_1^m \leqslant j \leqslant j_2^m.$$

In our case $j_2^m = j_1^m + 1$ and the line $j = j_1^m$ forms the lower surface and $j = j_2^m$ the upper surface for $i_2^m \geqslant i \geqslant i_1^m$. The two lines coincide in physical space for $i < i_1^m$ and $i > i_2^m$. Also, the original element canard or wing grid lies in the region

$$i_0^m \leqslant i \leqslant i_3^m; \qquad j_0^m \leqslant j \leqslant j_3^m.$$

The generation of the (13) and (23) grids is just a small algebraic step in the overall grid generation procedure: the elementary inner grid (3) is just a Cartesian grid and a simple formula is used for the coordinate values. These grids are not separately stored—the coordinate values are used as they are computed in the next grid blending step. In the rest of this section it will be assumed that $i_1^1 < i_2^1$; $i_1^1 < i_1^2$; $i_1^2 < i_2^2$, but i_2^1 not necessarily $< i_1^2$ (the canard and wing may overlap in i); similarly that $j_1^1 < j_2^1$; $j_1^1 > j_1^2$; $j_1^2 < j_2^2$ but j_1^1 not necessarily $> j_2^2$.

The composite inner grid, $r_{123}(\mathbf{l})$, is defined to approach $\mathbf{r}_{13}(\mathbf{l})$ as $j \to j_2^1$ $(j \leqslant j_2^1)$; and to approach $\mathbf{r}_{23}(\mathbf{l})$ as $j \to j_2^2$ $(j \geqslant j_2^2)$. For $j \geqslant j_2^1$ (upper part of grid) we have

$$\mathbf{r}_{123}(\mathbf{l}) = \mathbf{r}_{13}(\mathbf{l}) \tag{1}$$

while for $j \leqslant j_1^2$ (lower part of grid),

$$\mathbf{r}_{123}(\mathbf{l}) = \mathbf{r}_{23}(\mathbf{l}). \tag{2}$$

We first define the distance functions

$$z_1(\mathbf{l}) = \max(0, j_1^1 - j)$$
$$z_2(\mathbf{l}) = \max(0, j - j_2^2).$$

The function $z_1(\mathbf{l})$ is 0 where conditions (1) applies, and $z_2(\mathbf{l})$ is 0 where condition (2) applies. We then define a single distance function (z) that is 0, where (1) applies and 1 where (2) applies:

$$z = z_1/(z_1 + z_2).$$

Then, we finally have

$$\mathbf{r}_{123}(\mathbf{l}) = p(z)\,\mathbf{r}_{23}(\mathbf{l}) + [1 - p(z)]\,\mathbf{r}_{13}(\mathbf{l}),$$

where

$$p(z) = \tfrac{1}{2}[1 - \cos(\pi z)].$$

The grid $\mathbf{r}_{123}$ is shown in Fig. 12.

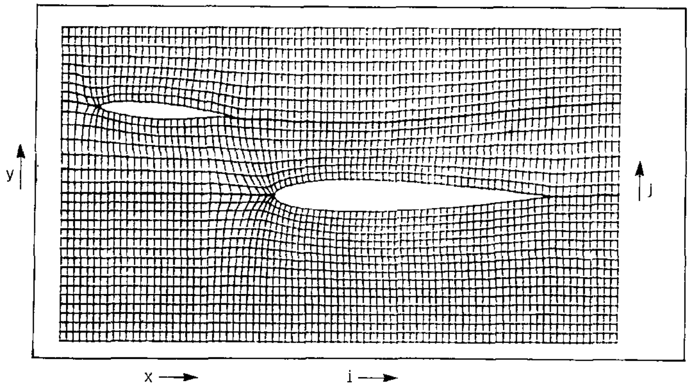

FIG. 12. Blended wing/canard grid (unstretched).

The purpose of the final blending is to stretch $\mathbf{r}_{123}$ in the far field. We want the final grid, $\mathbf{r}_c(\mathbf{l})$, to equal $\mathbf{r}_{123}(\mathbf{l})$ inside the region

$$i_1^1 \leqslant i \leqslant i_2^2; \qquad j_1^2 \leqslant j \leqslant j_2^1.$$

Defining grid 4 to lie in the region $i_1^4 \leqslant i \leqslant i_2^4$; $j_1^4 \leqslant j \leqslant j_2^4$;

$$z_1 = [(\max(0, i - i_2^2, i_1^1 - i))^2 + (\max(0, j - j_2^1, j_1^2 - j))^2]^{1/2}$$

$$z_2 = [(\min(i - i_1^4, i_2^4 - i))^2 + (\min(j - j_1^4, j_2^4 - j))^2)]^{1/2},$$

$$z = z_1/(z_1 + z_2),$$

we have the final grid,

$$\mathbf{r}_c(\mathbf{l}) = p(z)\, r_4(\mathbf{l}) + [1 - p(z)]\, r_{123}(\mathbf{l}),$$

where $p(z)$ is defined as above. This is shown in Fig. 13 and the inner part expanded

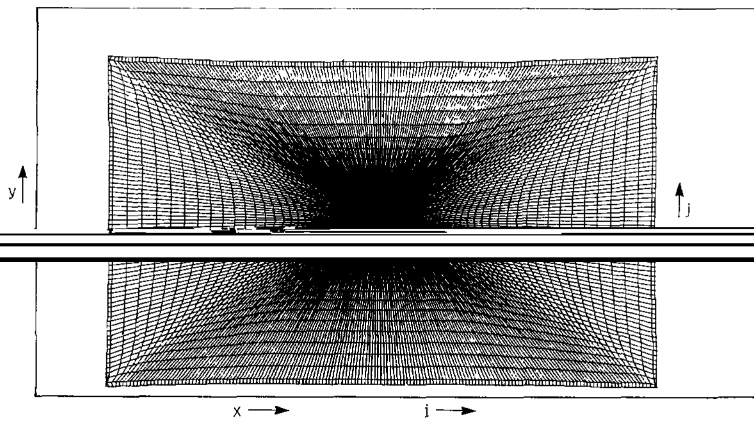

FIG. 13. Final wing/canard grid.

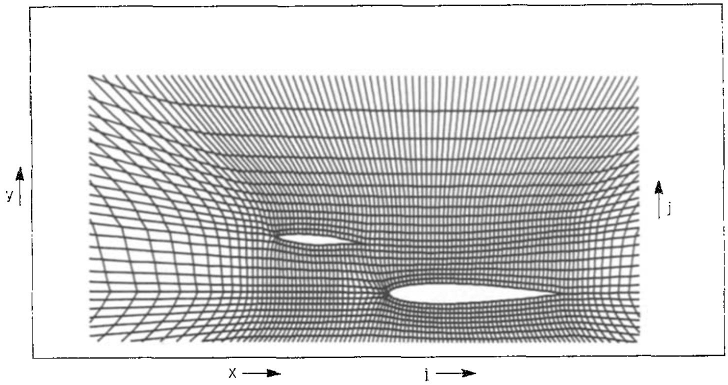

FIG. 14. Inner portion of final wing/canard grid.

in Fig. 14. It can be seen that the stretching is more efficient than with the conventional product form where the grid lines are continued to the outer boundary with the same spacing, as shown in Fig. 15.

It should be noted that there are a large number of ways of assembling the elementary grids into the final grid. We chose here a simple step-by-step method which is not necessarily the most efficient but perhaps is more instructive. Also, even though the intermediate grids were presented separately, they need not be generated separately. Even with the blending used here, all of the blending steps could be done together for each grid point (i, j) before computing the next point, so that only one pass through the grid need be made, and no intermediate grids need be generated. Some of these intermediate grids are only shown for clarity.

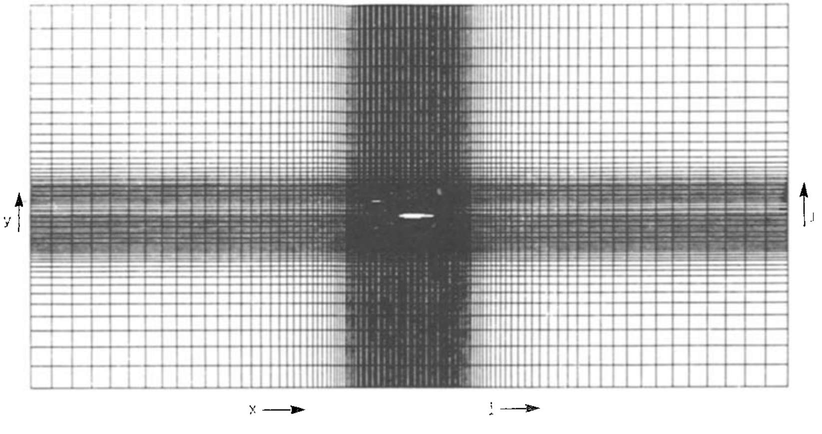

FIG. 15. Wing/canard grid with conventional stretching.

5. Example 3—Contiguous Segments

Here, we treat a set of smooth line segments as boundaries, so that the computational region is bounded by generalized polygons in physical space. If each elementary surface is a straight line, we choose each elementary grid to be a Cartesian grid; if it is curved, we choose another, simple grid that is curved. These are oriented so that a segment of one of the coordinate lines coincides with the given boundary segment. An example of the (block) type of grid that we treat in computational space is shown in Fig. 16. Each segment of the inner polygon as well as the outer boundary rectangle corresponds to a smooth line in physical space. Also, either the values of i or the values of j at the end points of each segment are equal, so that the segments are either horizontal or vertical in computational space.

The spacing of each elementary grid is determined by the spacing parallel to the boundary segment, and normal to it. The parallel spacing, Δs, is just the physical length of the segment divided by the number of cells along it. For a straight segment;

$$\Delta s = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2}/|n_2 - n_1|$$

where the subscript (1) refers to one end of the segment and (2) to the other, and n is either i or j. (This assumes that there is uniform grid spacing along the segment, which is not necessary for our method but is taken for simplicity.) The normal spacing is input externally for each segment. Also, the i and j values of the segments as well as the boundaries are the same in each elementary grid. That is, each elementary grid has the same i, j limits but different values of x and y at each point.

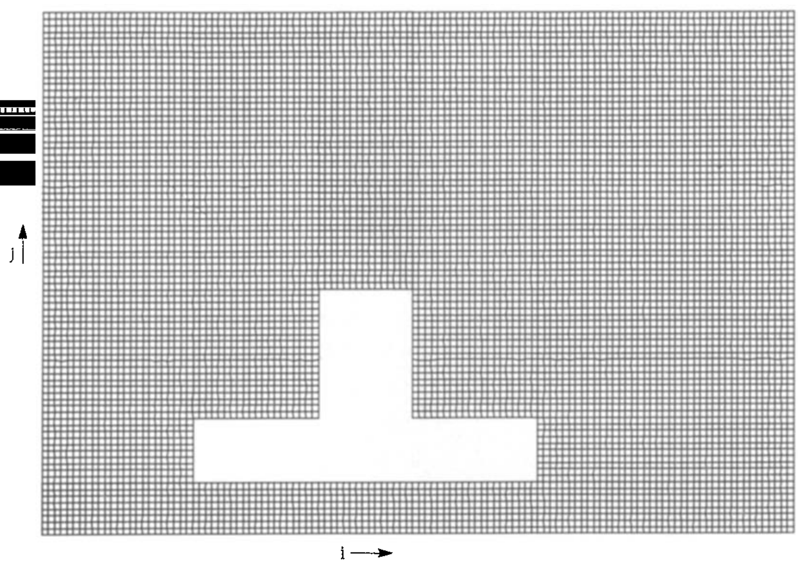

Fig. 16. Computational grid for automobile grid.

These values of x and y lie along a different elementary segment (in physical space) for each grid, for the appropriate values of i and j. For a curved segment we can, for example, start with a straight segment, generate the grid as above, and add a shearing (to form the curved segment) to the entire Cartesian sub-grid as well as to the boundary segment. Other methods can also be used to generate the subgrids.

As we approach some segment (k) in $\mathbf{I} = (i, j)$ space, the composite grid, $\mathbf{r}_c(\mathbf{I})$ must approach that particular elementary grid, $\mathbf{r}_k(\mathbf{I})$. Thus, we have

$$\mathbf{r}_c(\mathbf{I}) = \left[\sum P^m(\mathbf{I})\, \mathbf{r}_m(\mathbf{I})\right] \Big/ \sum P^m(\mathbf{I}).$$

We choose a distance function from point $\mathbf{I}$ to each segment similar to that in the last example:

$$\tilde{z}^m = [(\max(0, i - i_2^m, i_1^m - i))^2 + (\max(0, j - j_2^m, j_1^m - j))^2]^{1/2},$$

where we take

$$i_1^m \leqslant i_2^m; \qquad j_1^m \leqslant j_2^m.$$

Each $\tilde{z}^m$ vanishes on segment m. We then generalize the formulae of the last section to N segments instead of two. We define a "global" distance function for each segment that is 1 when $\mathbf{I}$ approaches the segment ($\tilde{z}^{m_1} \to 0$) and 0 when $\mathbf{I}$ approaches any other segment ($\tilde{z}^{m_2} \to 0$, $m_2 \neq m_1$):

$$z^m = \frac{1/\tilde{z}^m}{\sum_k 1/\tilde{z}^k}.$$

Then, we simply have

$$P^m(\mathbf{I}) = \tfrac{1}{2}[1 + \cos(\pi z^m)].$$

The composite grid resulting from applying these formulae to a particular set of segments is shown in Fig. 17, and an expanded view of the inner several grid lines

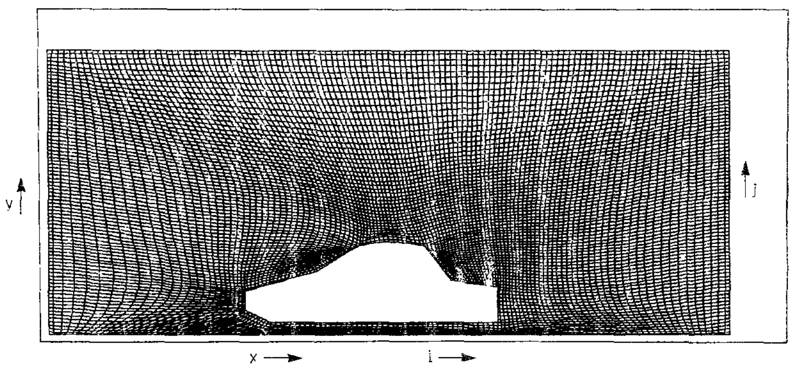

FIG. 17. Automobile grid.

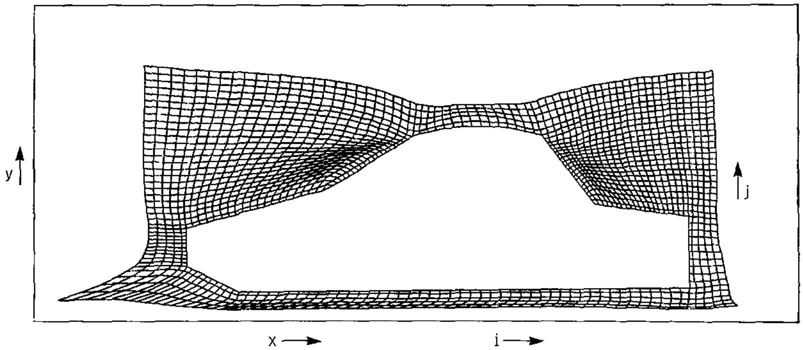

FIG. 18. Inner portion of automobile grid.

in Fig. 18. In this example all segments are straight except one, which is a circular arc. Although the spacing along each segment is constant (equal to the segment length in physical space divided by the length in computational space) the normal spacing is not: The cell height at the segment and on the grid line containing the segment is half that of the cells away from the segment, for added accuracy at the boundary. The code which generated these grids is less than 200 lines long, even though it can treat a number of separate polygons (the outer boundary is treated as just another polygon).

For generalization of this mapping, other boundary conforming elementary grids can be used instead of the simple ones shown here. Also, grid bunching near and normal to the segments can easily be implemented. In this case, a non-uniform spacing along each segment should be used that approximately matches the variable grid cell height normal to neighbouring segments.

6. CONCLUSION

A method of grid generation has been described that can be used to blend a number of elementary grids together into a smooth composite grid. If these elementary grids have desirable properties near a set of grid boundaries, such as orthogonality, then the composite grid can also be made to have them. This can be especially useful when designing a grid for a complex object such as an airplane, where methods already exist for generating good grids about each of the components. The method is computationally fast and, depending on the elementary grids, can be coded to recompute algebraically the entire grid for each iteration of some other solution scheme, which requires the grid. In these cases the full grid need not be stored in the computer, which can be an advantage in large 3-dimensional computations.

An additional feature is the recursive property, that allows more complex grids to be generated from simpler ones. This also allows "patches" to be blended into regions where the original composite grid has undesirable properties, such as excessive skewness or "folding over." Also, as described, simple unified methods of treating contiguous surface exist, as well as simple methods of refining the grid near these surfaces.

Although we have described some examples, the true usefulness of this method will only become apparent after it has been utilized in a large number of more complex cases and modifications are found to cure the many problems that are likely to arise.

Acknowledgments

The author would like to thank Mr. K. Ramachandran for generating the computer plots presented and Professor K. C. Reddy for making helpful comments concerning the manuscript.

References

1. P. R. Eisenman, "Grid Generation for Fluid Mechanics Computations," *Annual Reviews of Fluid Mechanics*, Vol. 17 (Annual Reviews, Palo Alto, 1985), p. 487.
2. J. F. Thompson, "Elliptic Grid Generation," *Numerical Grid Generation*, edited by J. F. Thompson (North-Holland, New York, 1982), p. 79.
3. A. Jameson, and D. A. Caughey, *in Proceedings of the Third AIAA Conference on Computational Fluid Dynamics, Albuquerque, NM*, 1977, p. 35; A. Jameson, *in Proceedings of Fourth AIAA Conference on Computational Fluid Dynamics, Williamsburg, VA*, 1979, p. 122.
4. A. Jameson, *Commun. Pure Appl. Math.* **27**, 283 (1974).
5. R. B. Pelz, and J. S. Steinhoff, *in Proceedings of the ASME Winter Meeting on Computers in Flow Predictions and Fluid Dynamics Experiments, Washington, D. C.*, 1981, p. 27.