JOURNAL OF COMPUTATIONAL PHYSICS 104, 56-68 (1993)

An Adaptively Refined Cartesian Mesh Solver for the Euler Equations

DARREN DE ZEEUW AND KENNETH G. POWELL

University of Michigan, Department of Aerospace Engineering, Ann Arbor, Michigan 48109-2140

Received April 26, 1991; revised February 21, 1992

A method for adaptive refinement of a Cartesian mesh for the solution of the steady Euler equations is presented. The algorithm creates an initial uniform mesh and cuts the body out of that mesh. The mesh is then refined based on body curvature. Next, the solution is converged to a steady state using a linear reconstruction and Roe's approximate Riemann solver. Solution-adaptive refinement of the mesh is then applied to resolve high-gradient regions of the flow. The numerical results presented show the flexibility of this approach and the accuracy attainable by solution-based refinement.

1. INTRODUCTION

In the past several years, unstructured mesh methods for fluid dynamics have become more and more prevalent, as a way of surmounting some of the difficulties of generating body-fitted meshes about arbitrary bodies [1–4]. Flowfields with multiple bodies are particularly difficult to generate structured meshes for, and multi-block methods [5] or patched-mesh methods [6] must be implemented in these cases if structured meshes are to be used.

Even unstructured mesh generation is not simple for complex configurations, however. Advancing front methods [2] must be carefully implemented to avoid high aspect-ratio or highly skewed cells; Delaunay methods [3, 4] typically require the generation of a cloud of points to triangulate, and special steps must be taken to avoid the breaking of boundary faces.

A very simple mesh generation technique is to "cut" the bodies in the flow out of a Cartesian mesh. Indeed, this technique is commonly used for potential flow calculations [7]. Advantages of using Cartesian meshes for Euler solvers, besides increased ease of mesh generation, include simpler flux formulations and simplifications in the data structure. In addition, Cartesian cells lead to fortuitous cancellation of truncation errors not occurring on less regular meshes. Two difficulties that arise in developing an Euler solver for a Cartesian mesh are:

1. poor resolution of geometric features such as leading and trailing edges, and
2. the introduction of cut cells that are a small fraction of the size of uncut cells.

Euler solvers based on central-differencing on Cartesian meshes have been developed by Clarke *et al.* [8] and Epstein *et al.* [9]. In the work of Clarke *et al.*, resolution of leading and trailing edges was achieved simply by clustering the mesh lines near the points of interest; cut cells were handled by merging them with neighboring uncut cells. In the work of Epstein *et al.*, resolution was achieved by local mesh refinement; cut cells were handled by a non-conservative extrapolation procedure. An upwind-differencing method on an adaptively refined Cartesian mesh has been developed by Berger and LeVeque [10], for unsteady flows. In their work, the large time-step method of Leveque [11] was used to help remove the numerical stiffness due to small cut cells.

In the work presented in this paper, resolution is achieved by use of adaptive refinement; cut cells are handled by a reconstruction method for general unstructured meshes coupled with local time-stepping. The work consists of a mesh generation based on a Cartesian mesh with geometry-based refinement [12], and a flow solver based on the MUSCL concept [13], with a linear reconstruction technique [14] and Roe's approximate Riemann solver [15]. In addition, solution-adaptive refinement of the mesh is used to gain resolution in high-gradient regions of the flow [16, 17]. Each element of the mesh-generation technique and the flow solver is described in the following text, along with the data structure used. Results for a collection of test cases are presented and discussed.

2. GENERATION OF THE MESH AND DATA STRUCTURE

2.1. *Data Structure*

The basic data structure used is a hierarchical cell-based quadtree structure—"parent" cells are refined by division into four "children" cells. This concept is illustrated in

0021-9991/93 $5.00

Fig. 1. Each cell has a pointer to its parent cell (if one exists) and to its four children cells (if they exist). The cells farthest down the hierarchy, that is, the ones with no children, are the cells on which the calculation takes place.

This tree structure contains all the connectivity information necessary to carry out the flow calculations; no other connectivity information is stored. For instance, finding a neighbor cell requires, in the best case, simply querying the parent cell for the location of another of its children. In the worst case, the tree must be traversed all the way to its root. More generally, the expected number of levels of the tree that must be traversed to find neighbor asymptotes to

$$E(n_{\text{lev}}) = \frac{1}{2} + \frac{1}{2}\left(\frac{2}{2} + \frac{1}{2}\left(\frac{3}{2}\cdots\right.\right.$$

$$= \sum \frac{k}{2^k}$$

$$\asymp 2;$$

i.e., the neighboring cell is typically a grandchild of the current cell's grandparent.

The decision of which information to store and which to compute each time it is needed, is based on a trade-off between memory usage and computational speed. Computing neighboring cells, for instance, takes 25% of the compute-time if this information is not stored; storing it, however, requires eight integers. Computing the level of a cell in the tree, on the other hand, takes 15% of the compute-time if it is not stored; storing it requires only one integer. Based on these trade-offs, cell-level information is stored, whereas cell-neighbor information is computed each time it is needed. The integer variables stored per cell are thus:

- 5 words—Pointers to one parent and four children cells
- 1 word—Cell type information (whether the cell is inside or outside the flow domain, whether it crosses a boundary, etc.)
- 1 word—Cell level (level of the tree at which the cell resides)

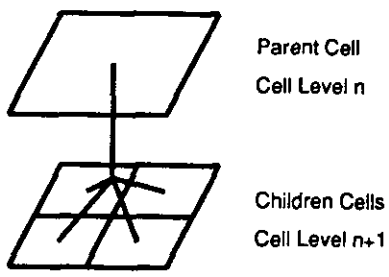

FIG. 1. Parent/children relationship.

- 2 words—Other temporary values—

for a total of nine words. Real variable stored per cell are:

- 4 words—Conserved variables
- 4 words—Temporary conserved variables for time-stepping scheme
- 4 words—Residuals
- 1 word—Time step
- 8 words—Gradients in X and Y directions
- 1 word—Limiter value for frozen limiter
- 3 words—Cell centroid and area

for a total of 25 words. This is not the minimal amount of storage that could be achieved, but is a practical balance of memory and compute-time needs. The amount of memory required for connectivity information (i.e., the integer variables) is quite small; nine words are stored that would not be necessary in a structured-grid code. The compute-time overhead is similarly small; 25% of the compute time is spent on tree-traversal, so that the code is approximately 25% slower than an unstructured-grid code with full connectivity information stored. A FORTRAN version of the code, with a five-stage explicit time-stepping scheme, requires 0.7 ms per iteration per cell on an IBM RS6000. A somewhat more efficient C version of the code requires 0.3 ms per iteration per cell.

The minimal connectivity discussed above has an added advantage; refining or coarsening a cell (i.e., spawning four children cells or removing four children cells) reduces primarily to a trivial change of the cell-based quadtree structure.

2.2. *Generation of an Initial Mesh*

The generation of the initial mesh begins with the creation of the "root" cell. Its size is determined from the size of the flow field and by what the user determines to be the coarsest acceptable mesh for that flow field. Then cells without children, initially just the root cell, are refined until the mesh reaches the coarsest acceptable mesh. This mesh serves as the initial mesh for cases in which no body is cut out of the grid. The procedure for computing the intersections of the body with the mesh depends upon how the body has been defined.

For a body defined by a piecewise analytic function $y(x)$, with the inverse function $x(y)$ also known, a node of the Cartesian mesh can be classified as inside or outside the body immediately, by determining whether the $x =$ constant and $y =$ constant lines through the node intersect the body an even or odd number of times between the node and the outer boundary. For a body defined by a piecewise analytic function $y(x)$, with the inverse function not known, the intersections of the line $x =$ constant with the body are known immediately; a search procedure must be set up to

determine the intersections in the x-direction. For a body defined simply by a set of data points, a search procedure must be carried out in both the x- and y-directions to find the intersections. For bodies defined in this manner, a spline (C^0 at trailing-edges, C^2 elsewhere) is fit through the points defining the body. The spline-fit of the body data points was found to be necessary to obtain smooth flow solutions.

Each cell in the Cartesian mesh is classified as to whether it is inside the body, cut by the body, outside of the body, or in the outer boundary based on the information found at the corners of that cell.

With this approach, very small cells can, and often do, appear in the mesh. The only limit placed on the cell size is that a node that is within a very small (machine-zero level) tolerance of a body is considered on that body, which in effect "pulls" the body out to the node. This avoids the problems associated with cells whose areas are on the order of machine accuracy. The remaining cell areas, in a typical case, vary as much as six orders of magnitude from one cell to its immediate neighbor. Due to the reconstruction procedure and the time-stepping procedure, these small cells do not degrade the accuracy or stability of the scheme. These small cells also seem to have little effect on the rate of convergence. Indeed, the supersonic channel flow case shown in the results section has the smallest cut cells, and yet it converges the most rapidly.

2.3. *Geometry-Based Mesh Refinement*

Geometry-based mesh refinement [12] is the next step in creating a suitable mesh. Once a suitable initial Cartesian mesh has been generated and the cut locations determined, a combination of cut-cell refinement and curvature refinement is applied to the mesh.

In cut-cell refinement, each cell cut by a body is refined, along with its three nearest neighbors. Refining neighbors of cut cells ensures a smoother transition to fine cells on the body from the coarser outer flow cells. Refinement of cut cells is applied successively until the mesh size on the body is brought to a user-specified level.

Once the desired cut-cell refinement is completed, curvature refinement is applied to the mesh. The slopes of the body faces on two consecutive cut cells are compared. If the difference in slopes is above a threshold value, both cells, and their nearest neighbors, are refined. The actual check used to flag cells for curvature refinement is given by

$$\left|\left(\frac{\Delta Y}{\Delta X}\right)_{\text{cell 1}} - \left(\frac{\Delta Y}{\Delta X}\right)_{\text{cell 2}}\right| \geq 0.05 \tag{1}$$

with special care taken for faces with ΔX small. An example mesh generated by the above procedures is shown in Fig. 2.

2.4. *Solution-Based Mesh Refinement*

An adaptive mesh may be refined or coarsened based on the characteristics of the flow about the body. Refinement takes place only after a solution is sufficiently converged. At that point, cells are flagged for refinement, based on the difference (undivided) of the total velocity between cells. If the total velocity difference is above a user-specified fraction (typically 5%) of the maximum total velocity difference, then the two cells sharing that face are flagged for refinement. For each cell that is flagged, four children cells are added to the quadtree data structure, one level farther down the hierarchy than their parent cell.

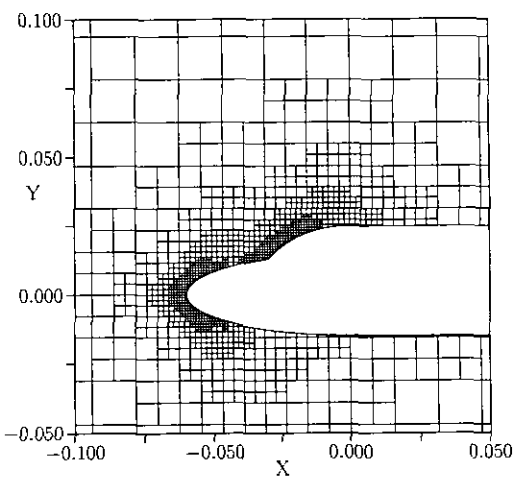

FIG. 2. Example mesh. Double ellipse grid plot.

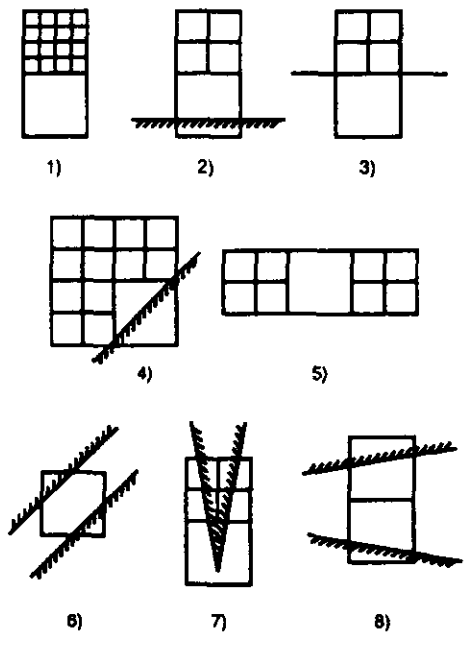

FIG. 3. "Undesirable" mesh features.

2.5. *Mesh "Smoothing"*

The mesh resulting from the above procedure could have certain "undesirable features." Some are undesirable in that allowing them would complicate the data structure; others are undesirable in that computational experience shows that they may degrade the solution somewhat in their vicinity. The eight features that are currently labelled "undesirable" are described below, and depicted in Fig. 3.

1. Cell level differences greater than 1 between two neighboring cells
2. Cell level differences greater than 0 normal to body-cut cells
3. Cell level differences greater than 0 through outer flow boundaries
4. Cell level differences greater than 0 between three-sided cells and their neighbors
5. "Holes" in the mesh
6. More than two cuts on a cell
7. Cell level differences greater than 0 on trailing edge of body
8. Bodies too close, only two cells apart.

When an undesirable feature is found, the mesh is "smoothed" to eliminate it, by refining appropriate cells until the feature no longer exists. This is a recursive procedure, that converges to a mesh with no undesirable features.

3. FLOW SOLVER

The flow solver described here consists of three primary components: a linear reconstruction method for obtaining accurate, limited values of the flow variables at face midpoints; an approximate Riemann solver for computing the flux through cell-faces; and a multi-stage time-stepping scheme for advancing the solution to a steady state. The individual components of these procedures, along with the procedures used at boundaries, are described below.

3.1. *Reconstruction Procedure*

In order to evaluate the flux through a face, flow quantities are required at both sides of the face. To achieve higher-order accuracy, solution-gradient information must be used. A linear reconstruction method [1] is used to determine a second-order approximation to the state at the face midpoint, based on the cells in the neighborhood of the face. It relies on a suitable path integral about the cell of interest, with the gradient of a quantity W_k in a cell being determined by

$$\nabla W_k = \frac{1}{A_\Omega} \int_{\partial\Omega} W_k \hat{\mathbf{n}}\, dl, \tag{2}$$

where A_Ω is the area enclosed by the path of integration, $\partial\Omega$. Here, W_k represents the quantity being reconstructed; in this work, the primitive variables $\mathbf{W} = (\rho, u, v, p)^T$ are reconstructed.

The path for the integration is constructed by connecting the centroids of neighboring cells. Away from cut cells or cell-level differences, the eight immediate neighbors of a cell are used. Near a body, as few as four cells are used to construct the path. The path is determined by finding the nearest neighbor in each of eight directions (north, northeast, east, southeast, south, southwest, west, and northwest). If two neighbors are found in one direction (as can occur when the neighbor is at a different level of the tree), both are used. If no valid neighbor is found, the cell itself is used as the neighbor in that direction.

These "altered" paths lead to less accurate approximations to the gradient than those calculated in regions away from cut cells or cell-level differences. The reconstruction scheme described below, however, retains the property of exact reconstruction of a linear function, even in regions where altered paths are used, as long as a minimum of three non-colinear cell-centers are used to form the path. Some examples of the paths are shown in Fig. 4, with the X denoting the cell for which the gradient is being calculated.

Once the cells in the path are determined, the path integral is carried out numerically. In general, the area inside the path is calculated by summing the areas of the triangles formed by connecting the centroids in the path to the centroid of the cell for which the gradient is being calculated.

Once the gradient of W_k is known in each cell, the value of W_k can be found anywhere in the cell from

$$W_k(x, y) = W_k^c + \nabla W_k \cdot \mathbf{dr}, \tag{3}$$

where W_k^c is the value of W_k at the cell centroid, and $\mathbf{dr}$ is defined as

$$dr_x = x - x^c, \qquad dr_y = y - y^c. \tag{4a}$$

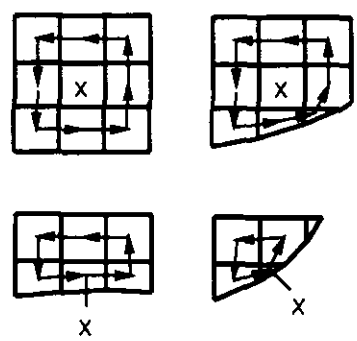

FIG. 4. Normal and special paths.

For example, the values at the face midpoints of an uncut cell are simply

$$W_k^{\text{top}} = W_k^c + \frac{\Delta y_{\text{cell}}}{2} \nabla_y W_k \tag{5a}$$

$$W_k^{\text{right}} = W_k^c + \frac{\Delta x_{\text{cell}}}{2} \nabla_x W_k \tag{5b}$$

$$W_k^{\text{bottom}} = W_k^c - \frac{\Delta y_{\text{cell}}}{2} \nabla_y W_k \tag{5c}$$

$$W_k^{\text{left}} = W_k^c - \frac{\Delta x_{\text{cell}}}{2} \nabla_x W_k. \tag{5d}$$

3.2. *Limiting*

If the full gradient were used in reconstructing the values at face midpoints, the computed values could fall outside the bounds of the data used in the path integral. To avoid this, the computed gradients are *limited*; that is, the primitive variables $\mathbf{W} = (\rho, u, v, p)^{\mathrm{T}}$ are reconstructed via

$$\mathbf{W}(x, y) = \mathbf{W}^c + \phi \nabla \mathbf{W} \cdot \mathbf{dr}, \tag{6}$$

where ϕ is a limiter, with a value between zero and one. In regions where $\phi = 1$, a linear reconstruction is being used, and the truncation error is $\mathcal{O}(h^2)$; in regions where $\phi = 0$, a piecewise constant reconstruction is being used and the truncation error is $\mathcal{O}(h)$ [1]. The limiter ϕ is defined as

$$\phi = \min \begin{cases} 1 \\ \min_k \left(\dfrac{|W_k^c - \max_{\text{path}}(W_k)|}{|W_k^c - \max_{\text{cell}}(W_k)|} \right), \\ \min_k \left(\dfrac{|W_k^c - \min_{\text{path}}(W_k)|}{|W_k^c - \min_{\text{cell}}(W_k)|} \right) \end{cases} \tag{7}$$

which is a diffusive limiter of the minmod variety [18]. The minimum and maximum over the path are found by examining the values of W_k used in the path integration; the minimum and maximum over the cell are found by using the gradient to reconstruct W_k at the corners of the cell. Thus, the limiter acts to ensure that the values of W_k at the nodes of the cell for which the gradient is being calculated are bounded by the values of W_k that are used in calculating the gradient. Using a single limiter for the gradient of the vector $\mathbf{W}$ was found to give superior convergence. In standard MUSCL-type schemes [13], a separate limiter is typically used for each variable and for each mesh direction, resulting in eight limiters for each cell.

Unfortunately, limiting can seriously hamper the convergence to a steady state, with the non-linearity of the scheme resulting in limit cycles. To combat this problem, the limiter values are "frozen" after a certain point in the convergence, using previously stored values of the limiter rather than recomputing them at each time step. Freezing the limiters allows the residuals to converge to machine zero.

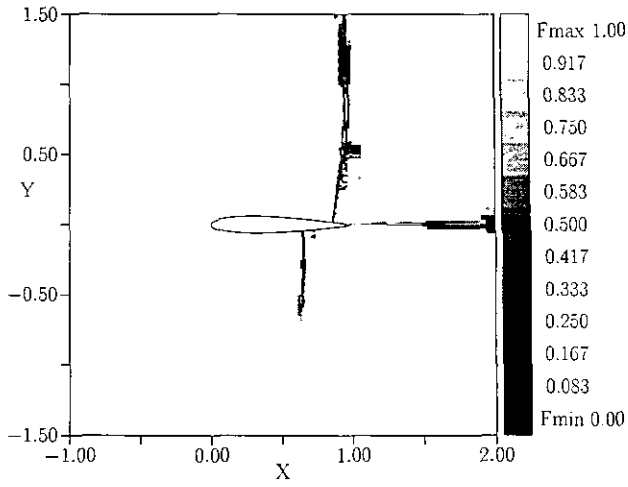

FIG. 5. Limiter values—transonic airfoil. NACA 0012; values of the limiter ϕ.

Typical values of the limiter are shown in Fig. 5, in which the $\phi(x, y)$ is plotted for a transonic airfoil case. As can be seen, a linear reconstruction ($\phi = 1$) is used nearly everywhere. In the immediate vicinity of the shocks and the wake, the limiter reduces the order of accuracy of the scheme. The percentage of cells in which the limiter is less than one is extremely small, however. The limiter values on the body are one, allowing the full accuracy of the scheme there. The order of the scheme, in a case in which the limiter is everywhere equal to one (an airfoil at subcritical speed), is shown in Fig. 6. This figure is a log–log plot of the drag (which should be zero) as a function of the size of an uncut cell of the mesh. The slope of this curve, for all but the very

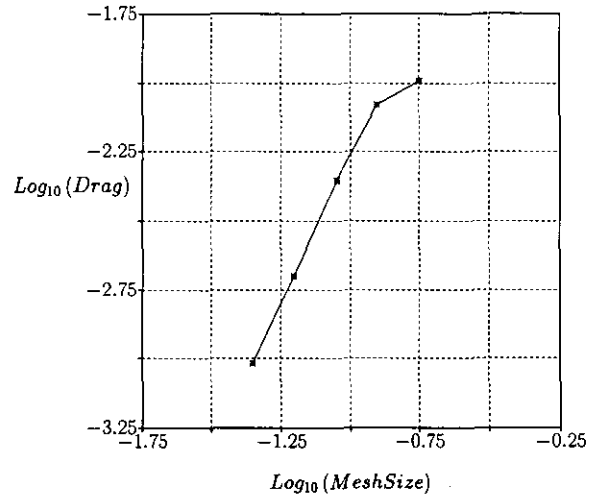

FIG. 6. Convergence rate of drag as a function of mesh size.

coarsest mesh (which has only eight points on the airfoil), is two, demonstrating the second-order global accuracy of the scheme.

3.3. *Approximate Riemann Solver*

The finite-volume form of the Euler equations can be written as

$$\frac{d\mathbf{U}}{dt} = -\frac{1}{A}\sum_{\text{faces}} (\mathbf{F}\,\Delta y - \mathbf{G}\,\Delta x), \tag{8}$$

where A is the area of that cell, Δx and Δy are the changes of x and y along a face (defined so that the integral is carried out in a counterclockwise sense), and $\mathbf{U}$, $\mathbf{F}$, and $\mathbf{G}$ are defined as

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \tag{9a}$$

$$\mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix} \tag{9b}$$

$$\mathbf{G} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vH \end{pmatrix}. \tag{9c}$$

Defining the face length and the normal and tangential velocities as

$$\Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{10}$$

$$u_n = \frac{(u\,\Delta y - v\,\Delta x)}{\Delta s} \tag{11a}$$

$$u_t = \frac{(u\,\Delta x + v\,\Delta y)}{\Delta s}, \tag{11b}$$

the flux through a face may be written as

$$(\mathbf{F}\,\Delta y - \mathbf{G}\,\Delta x) = \begin{pmatrix} \rho u_n \\ \rho u_n u + p\dfrac{\Delta y}{\Delta s} \\ \rho u_n v - p\dfrac{\Delta x}{\Delta s} \\ \rho u_n H \end{pmatrix} \Delta s \equiv \mathbf{\Phi}\,\Delta s. \tag{12}$$

The flux through a face is a function of the values at the face midpoint, given by the reconstruction in the cells to the "left" and "right" of the face. Using Roe's approximate Riemann solver, this flux function is

$$\mathbf{\Phi}(\mathbf{U}_L, \mathbf{U}_R) = \tfrac{1}{2}[\mathbf{\Phi}(\mathbf{U}_L) + \mathbf{\Phi}(\mathbf{U}_R)] - \tfrac{1}{2}\sum_{k=1}^{4} |\hat{a}_k|^* \,\Delta V_k\, \hat{\mathbf{R}}_k \tag{13}$$

with

$$\hat{\mathbf{a}} = \begin{pmatrix} \hat{u}_n - \hat{c} \\ \hat{u}_n \\ \hat{u}_n \\ \hat{u}_n + \hat{c} \end{pmatrix} \tag{14a}$$

$$\Delta\mathbf{V} = \begin{pmatrix} \dfrac{\Delta p - \hat{\rho}\hat{c}\,\Delta u_n}{2\hat{c}^2} \\ \dfrac{\hat{\rho}\,\Delta u_t}{\hat{c}} \\ \Delta\rho - \dfrac{\Delta p}{\hat{c}} \\ \dfrac{\Delta p + \hat{\rho}\hat{c}\,\Delta u_n}{2\hat{c}^2} \end{pmatrix} \tag{14b}$$

$$\hat{\mathbf{R}} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ \hat{u} - \hat{c}\dfrac{\Delta y}{\Delta s} & \hat{c}\dfrac{\Delta x}{\Delta s} & \hat{u} & \hat{u} + \hat{c}\dfrac{\Delta y}{\Delta s} \\ \hat{v} + \hat{c}\dfrac{\Delta x}{\Delta s} & \hat{c}\dfrac{\Delta y}{\Delta s} & \hat{v} & \hat{v} - \hat{c}\dfrac{\Delta x}{\Delta s} \\ H - \hat{u}_n\hat{c} & \hat{u}_t\hat{c} & \dfrac{\hat{u}^2 + \hat{v}^2}{2} & H + \hat{u}_n\hat{c} \end{pmatrix} \tag{14c}$$

and

$$\hat{\rho} = \sqrt{\rho_L \rho_R} \tag{15a}$$

$$\hat{u} = \frac{\sqrt{\rho_L}\,u_L + \sqrt{\rho_R}\,u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{15b}$$

$$\hat{v} = \frac{\sqrt{\rho_L}\,v_L + \sqrt{\rho_R}\,v_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{15c}$$

$$\hat{H} = \frac{\sqrt{\rho_L}\,H_L + \sqrt{\rho_R}\,H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{15d}$$

where $\hat{c}$, $\hat{u}_n$, and $\hat{u}_t$ are calculated directly from $\hat{\rho}$, $\hat{u}$, $\hat{v}$, and $\hat{H}$. The flux difference terms (the summation in Eq. (13)) provide the upwind character which stabilizes the scheme.

To prevent expansion shocks, an entropy fix is imposed [19]. A smoothed value, $|\hat{a}^{(k)}|^*$, is defined to replace $|\hat{a}^{(k)}|$

for the two acoustic waves ($k=1, k=4$). For those two waves,

$$|\hat{a}^{(k)}|^* = \begin{cases} |\hat{a}^{(k)}|, & |\hat{a}^{(k)}| \geqslant \frac{1}{2}\delta a^{(k)} \\ \dfrac{(\hat{a}^{(k)})^2}{\delta a^{(k)}} + \dfrac{1}{4}\delta a^{(k)}, & |\hat{a}^{(k)}| \leqslant \frac{1}{2}\delta a^{(k)} \end{cases}$$

$$\delta a^{(k)} = \max(4\Delta a^{(k)}, 0), \qquad \Delta a^{(k)} = a_{\mathrm{R}}^{(k)} - a_{\mathrm{L}}^{(k)}. \tag{16}$$

For each cell, the face fluxes, calculated as above, are summed to give the residual for the cell,

$$\mathrm{Res}(\mathbf{U}) = -\frac{1}{A}\sum_{\text{faces}} \mathbf{\Phi}\, \Delta s. \tag{17}$$

These residuals are then integrated in time, as described below.

3.4. *Time-Stepping Scheme*

The time-stepping scheme used is one of the optimally smoothing multi-stage schemes developed by Tai [20, 21]. The general m-stage scheme is defined as

$$\begin{aligned} \mathbf{U}^{(0)} &= \mathbf{U}^n \\ \mathbf{U}^{(k)} &= \mathbf{U}^{(0)} + \alpha_k\, \Delta t\, \mathrm{Res}(\mathbf{U}^{(k-1)}), \quad k = 1, m \\ \mathbf{U}^{n+1} &= \mathbf{U}^{(m)}. \end{aligned} \tag{18}$$

The five-stage scheme which gives optimal damping of Fourier modes in the range $\pi/2 \leqslant k\,\Delta x \leqslant \pi$ has multi-stage coefficients

$$\begin{aligned} \alpha_1 &= 0.0695 \\ \alpha_2 &= 0.1602 \\ \alpha_3 &= 0.2898 \\ \alpha_4 &= 0.5060 \\ \alpha_5 &= 1.0000 \end{aligned} \tag{19}$$

and CFL number 1.1508.

Local time-stepping is used and, indeed, is necessary. The meshes generated have extremely large differences in area from cell to cell, due to the cut cells. A representative convergence history is shown in Fig. 7.

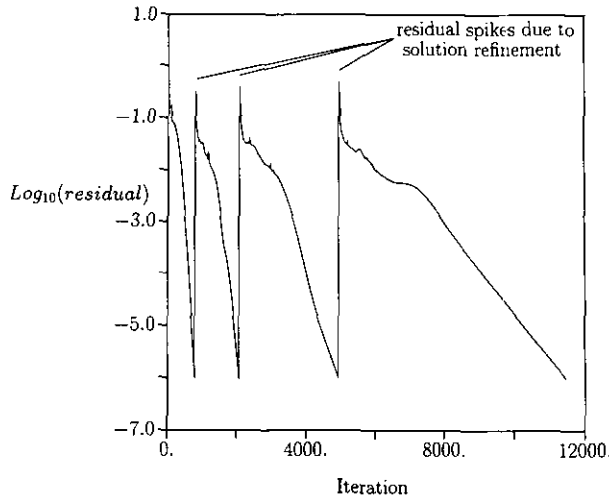

FIG. 7. Representative residual convergence history.

3.5. *Boundary Procedures*

Boundary procedures are handled by the approximate Riemann solver. For flow-through boundaries, free-stream conditions are specified in "ghost cells" just outside the computational domain, and the Riemann-problem solution at these boundary faces gives the boundary flux. For solid-wall boundaries, the mass and energy flux are set to zero, and only the pressure terms are kept in the momentum fluxes. The pressure used in the wall flux is the value given at the center of the wall face by the reconstruction procedure. This corresponds to a linear extrapolation of the pressure to the wall.

4. POST-PROCESSING

Post-processing requires transferring the known cell-centered values to nodal values for plotting as accurately as possible. To do this, the limited cell gradients are used to extrapolate to each cell's nodes as shown in Fig. 8.

The function as reconstructed at the nodes is multi-valued; there is one value resulting from the representation in each cell sharing that node. These multiple values at each node are averaged to yield a single accurate, bounded value there.

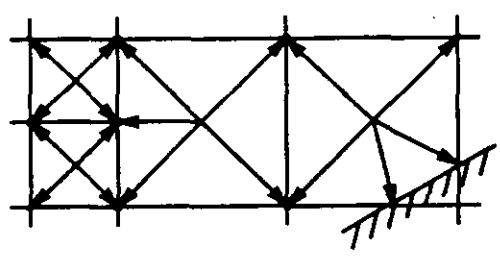

FIG. 8. Obtaining post-processed nodal values.

5. CURRENT RESULTS

The method described above has been tested on several internal and external flows. The cases were chosen so as to show the flexibility of the Cartesian mesh approach and the fidelity of results available by use of solution-based refinement.

5.1. *Transonic NACA Airfoil*

The steady-state flow was computed about a NACA 0012 airfoil at $M_\infty = 0.85$ and $\alpha = 1°$. For these conditions, shocks exist on the upper and lower surfaces of the airfoil.

The outer boundary of the mesh was set at a 2048-chord radius. This virtually eliminated the need for a vortex boundary condition; simply specifying the free-stream state at the outer boundary faces proved adequate. The initial mesh was refined based on curvature, to resolve the leading and trailing edge. Three levels of solution-based refinement were done, as well.

As can be seen in Fig. 9 and 10, both shocks, as well as the wake, are well resolved. The pressure coefficient, plotted in Fig. 11, results in a lift of 0.3670 and a drag of 0.0581.

To show the effect of the outer boundary conditions on the solution, this case was run for outer boundaries ranging from a four-chord to a 2048-chord radius. Geometry-based refinement was carried out so as to ensure an equivalent mesh in the vicinity of the airfoil for each case. Three levels of solution-based refinement were also done for each case. As can be seen in Fig. 12, this wide range of outer boundary radius resulted in only a small change in the total number of computational cells. Even though the crudest possible outer boundary condition (free-stream flow) was enforced, the lift and drag are converged by the 512 chord case. Thus, by use of mesh refinement, simple outer boundary conditions may be enforced, at a radius far from the body, at very little computational cost.

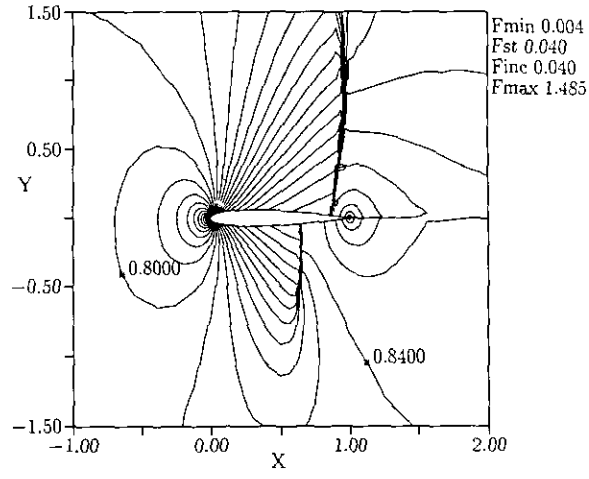

FIG. 10. NACA 0012, Mach number line contours: 9609 cells, level 4 mesh, lift = 0.3670, drag = 0.0581.

5.2. *"Subsonic" Three-Element Airfoil*

For this case, $M_\infty = 0.2$ and $\alpha = 0°$. Portions of the mesh are shown in Fig. 13, 14, and 15. As can be seen, the geometry-adaptive refinement gives sufficient resolution of leading and trailing edges. Contours of Mach number are shown in Figs. 16, 17, and 18. Due to the high effective camber caused by the flaps, the flow actually expands to supersonic speed about the leading edge, and a weak shock is present. The Mach number and pressure coefficient distributions on the surface are shown in Figs. 19 and 20. Despite large variations in cell size on the body (the smallest cut cell is a factor 10^6 smaller than its uncut neighbor) the solution is smooth. The total pressure loss, $1 - p_0/p_{0_\infty}$, is

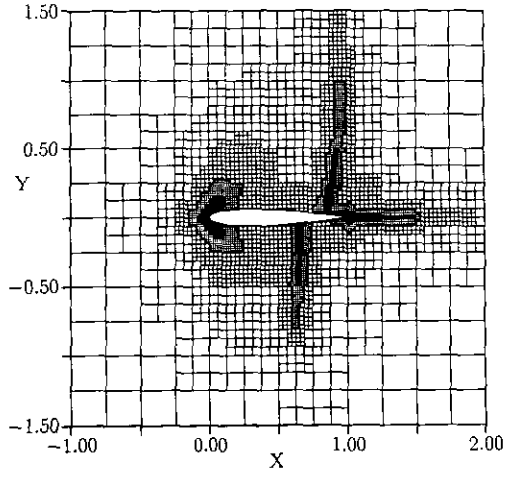

FIG. 9. NACA 0012 mesh: 9609 cells, level 4 mesh, lift = 0.3670, drag = 0.0581.

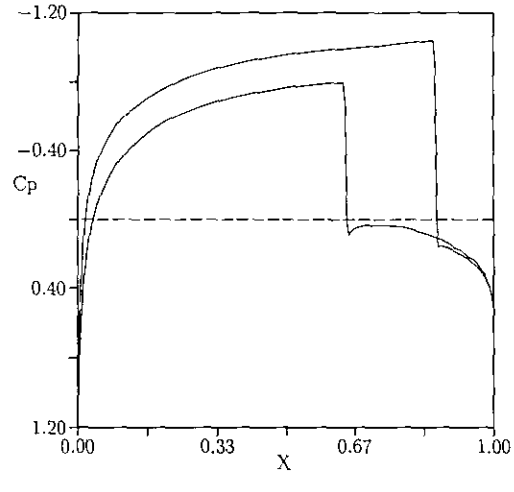

FIG. 11. NACA 0012, pressure coefficient cross-sections, *Cp* on the body: 9609 cells, level 4 mesh, lift = 0.3670, drag = 0.0581.

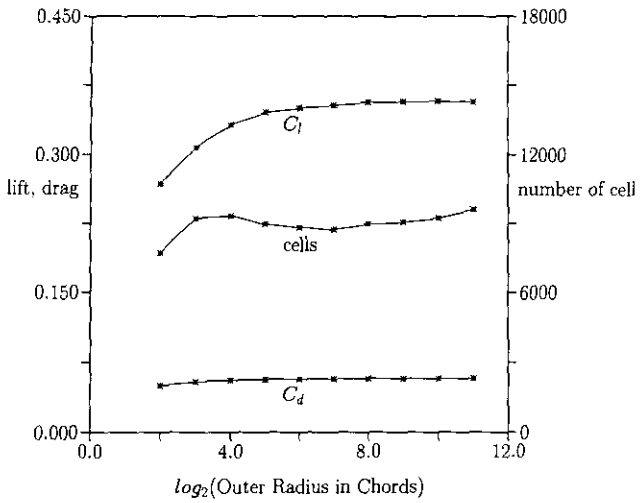

FIG. 12. NACA 0012, $M_\infty = 0.85$ and $\alpha = 1°$, lift, drag, and number of cells vs. boundary radius.

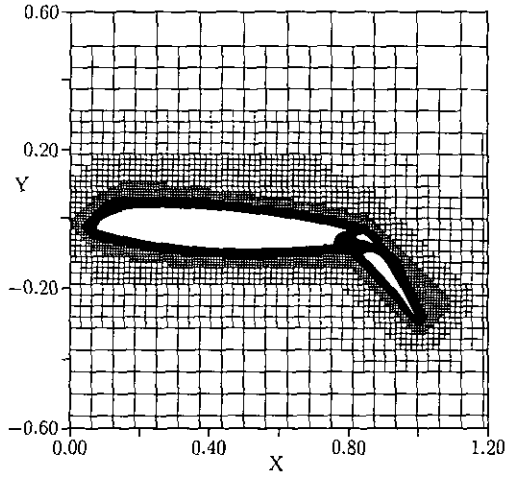

FIG. 13. Three-element airfoil mesh: 21,034 cells.

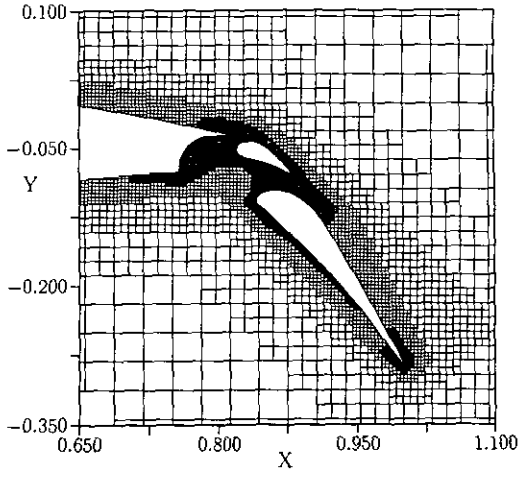

FIG. 14. Three-element airfoil mesh (detail).

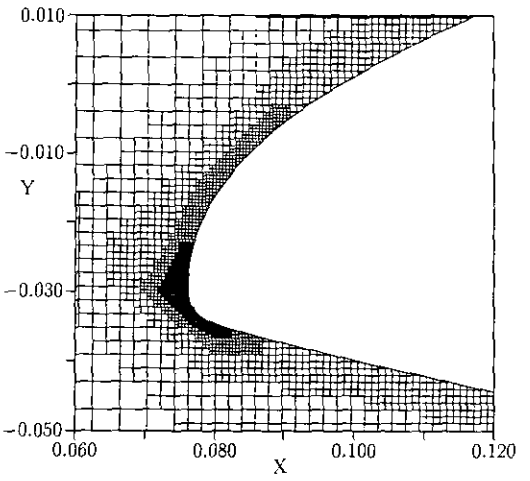

FIG. 15. Three-element airfoil mesh (detail).

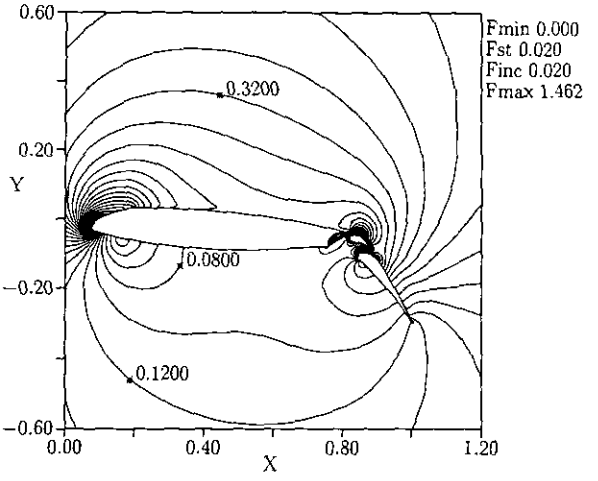

FIG. 16. Three-element airfoil Mach number contours.

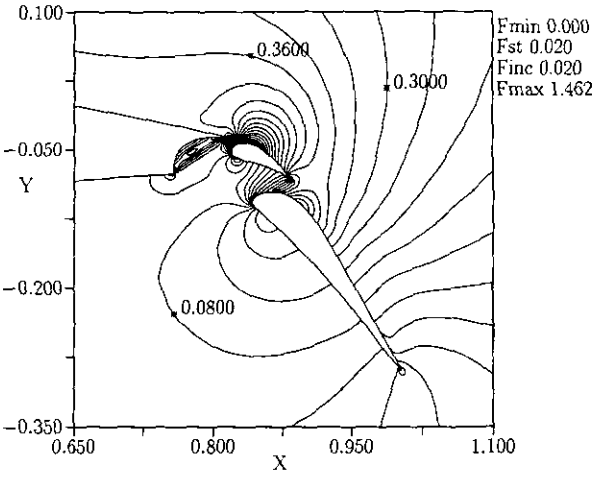

FIG. 17. Three-element airfoil Mach number contours (detail).

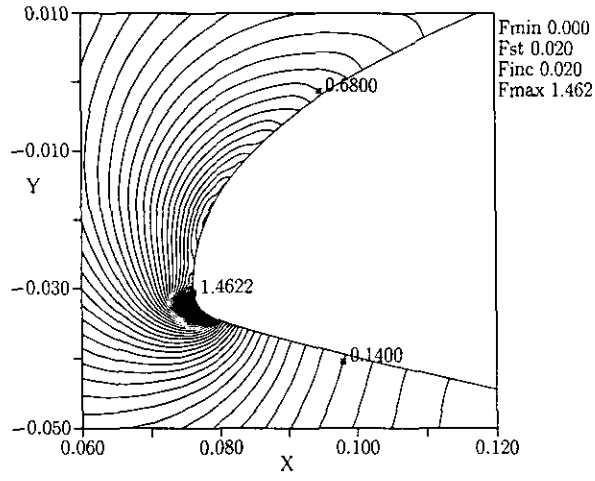

FIG. 18. Three-element airfoil Mach number contours (detail).

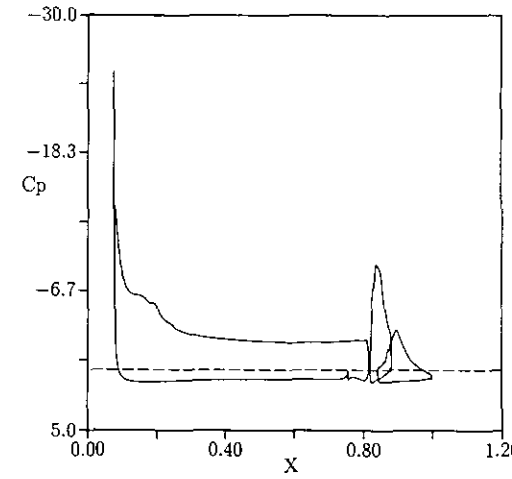

FIG. 20. Three-element airfoil, pressure coefficient cross-sections, C_p on the surface; lift = 3.7842, drag = 0.0159.

shown in Fig. 21. There is some total pressure loss on the upper surface of the main element ($\approx 0.5\%$), due to the shock at the leading-edge. On the lower surface of the main element, the loss is less than 0.1%. There are some spikes in the losses at leading and trailing edges of each element. The lift computed for the system is $C_l = 3.7842$; the drag computed is $C_d = 0.0159$.

5.3. *Supersonic Double Ellipse*

For this case, $M_\infty = 8.15$ and $\alpha = 30°$. The mesh was refined four times based on the solution; the resulting mesh is shown in Fig. 22. The refined regions correspond to the nose, the bow shock, and the canopy shock. The Mach number and pressure contours are shown in Figs. 23 and 24, respectively. The shocks and the expansion about the nose are all well resolved. Finally, the pressure coefficient on the body is plotted in Fig. 25. Despite the presence of small cut cells on the body, the pressure distribution is very smooth and the canopy shock is captured cleanly.

5.4. *Supersonic Channel Flow*

In this case, the steady-state flow was computed in a channel with a 15° compression corner, followed by a 15° expansion corner. The free-stream Mach number is $M_\infty = 2.0$. There is an attached shock at the compression corner, which reflects from the top wall, forming a small Mach stem. The shock reflects from the bottom wall as well, before exiting the channel. The expansion corner acts to weaken the reflected shock. There is also a slip line, emanating from the triple point near the upper wall.

The Mach number contours and mesh are shown in Fig. 26. A blowup of the Mach stem is shown in Figs. 27 and

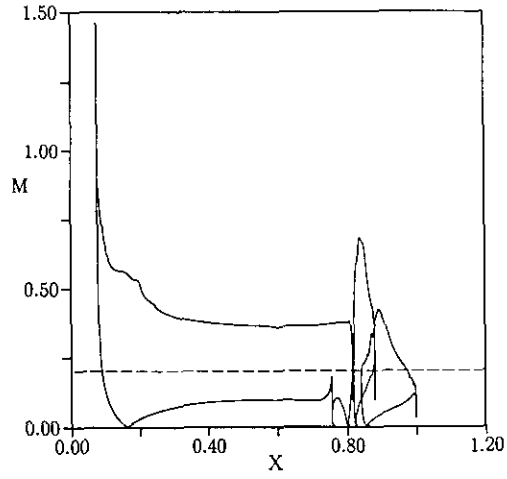

FIG. 19. Three-element airfoil Mach number cross sections.

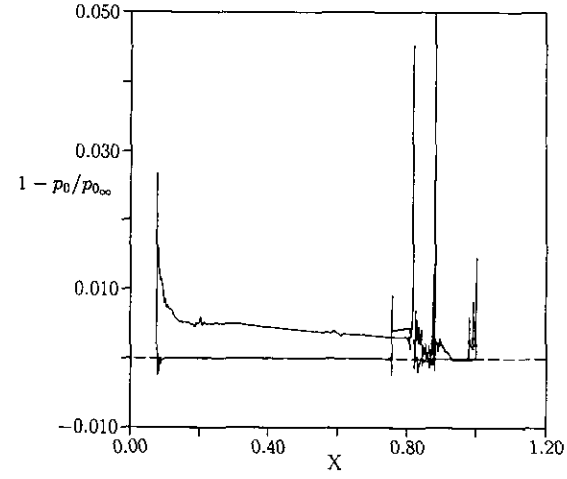

FIG. 21. Three-element airfoil, total pressure loss cross-sections, $1 - p_0/p_{0_\infty}$ on the surface.

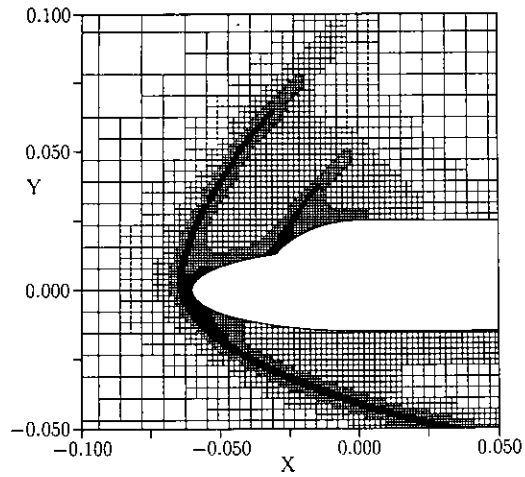

FIG. 22. Double ellipse grid plot: 17,432 cells, level 6 mesh.

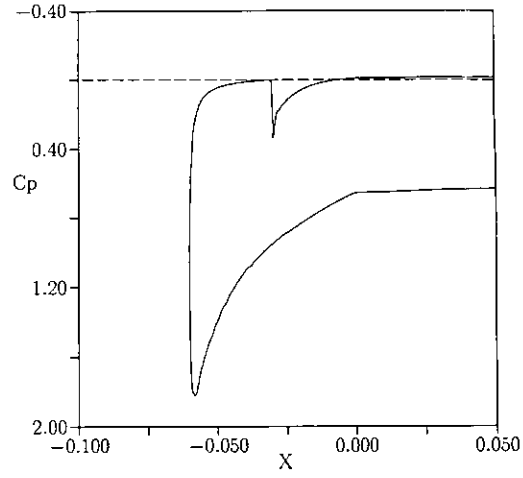

FIG. 25. Double ellipse pressure coefficient cross sections, on the Body.

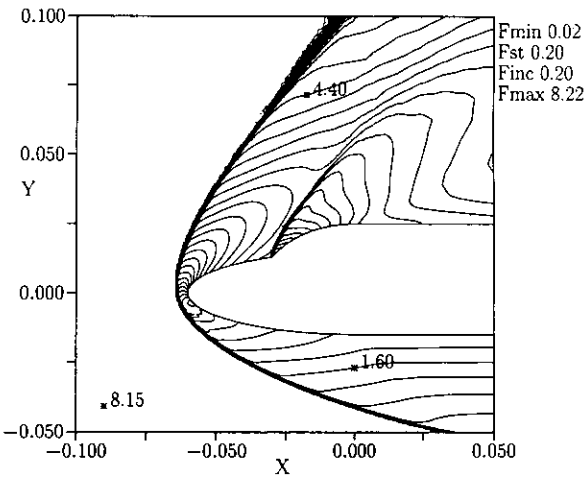

FIG. 23. Double ellipse Mach number contours.

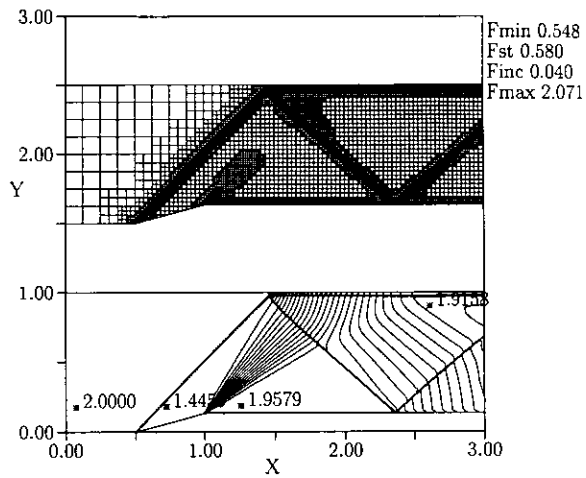

FIG. 26. Fifteen-degree wedge Mach number contours: 26,228 cells, level 7 mesh.

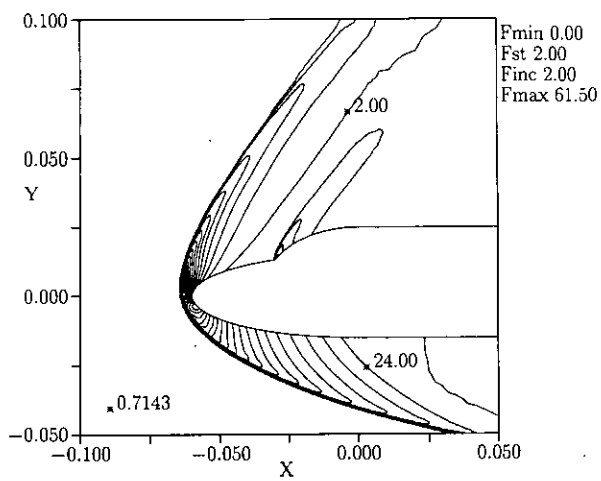

FIG. 24. Double ellipse pressure contours.

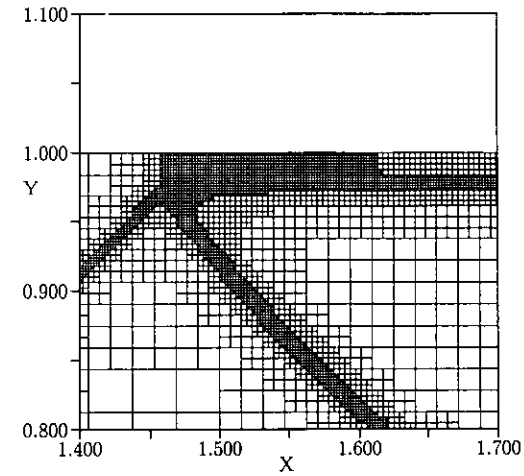

FIG. 27. Fifteen degree wedge grid plot (detail).

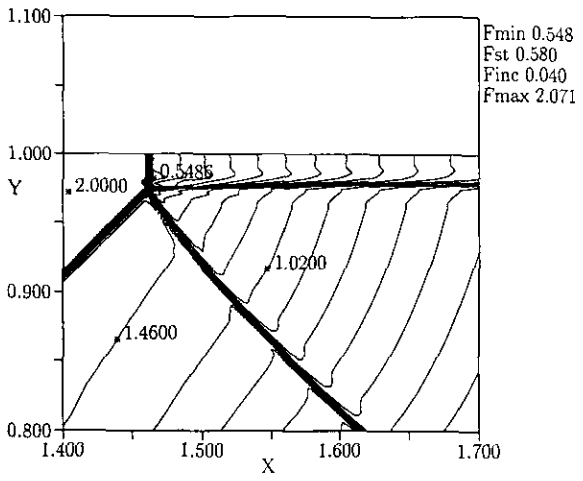

FIG. 28. Fifteen degree wedge Mach number line contours.

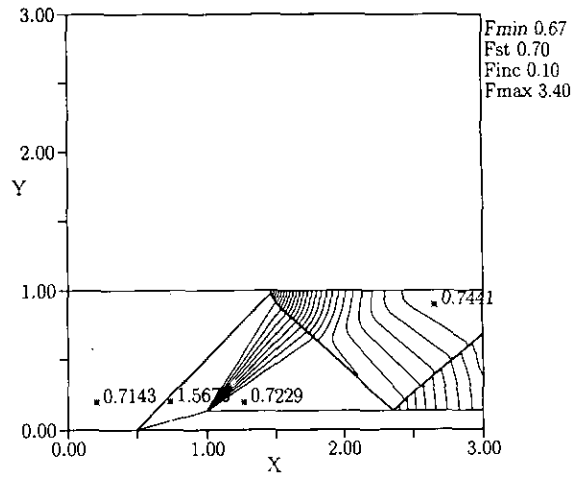

FIG. 29. Fifteen degree wedge pressure line contours.

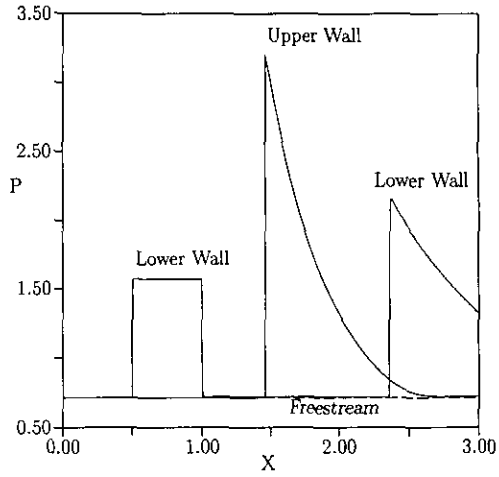

FIG. 30. Fifteen degree wedge pressure cross sections.

28. The shear that emanates from the triple point of the Mach stem is carried cleanly out through the flow, although somewhat weakened by the expansion. The pressure contours plotted in Fig. 29 pass smoothly through the shear, as they should. The pressure on the wall is shown in Fig. 30. Note that the pressure is constant on the incline of the ramp. Cell areas on the incline vary as much as six orders of magnitude from cell to cell without having a detrimental effect on the solution.

6. CONCLUDING REMARKS

An adaptive Cartesian mesh algorithm has been developed and has been used successfully to obtain steady-state solutions of the Euler equations for a variety of internal and external flows. Test cases included a transonic airfoil, a three-element airfoil, a double ellipse, and a channel flow. For airfoil cases, the outer boundary can be placed arbitrarily far from the airfoil without significantly increasing the number of cells, thus eliminating the need for sophisticated outer boundary conditions. Both geometry- and solution-based refinement effectively enhance resolution of regions of high body curvature and large flow gradients. The reconstruction method, coupled with local time stepping, eliminate the problems typically associated with the small cut cells caused by cutting the body from the Cartesian mesh.

Extension of the inviscid algorithm to three dimensions appears promising. The appropriate data structure in the three-dimensional case is an octree, with each parent cell being divided into eight children cells. Extension of the two-dimensional algorithm to viscous-flow cases is less straightforward, but no less promising. Because of the anisotropy of high-Reynolds-number flow, the isotropic refinement scheme of the current algorithm (splitting each parent cell along two Cartesian directions, giving four children cells) must be replaced by a direction refinement. This suggests the use of a binary-tree data structure, with each parent cell being divided into two children cells. In addition, cells near a boundary should be cut, not along a Cartesian direction, but along a body-aligned direction. If an appropriate anisotropic grid-refinement technique such as this can be developed, the solution technique described in this paper can be extended fairly easily to a powerful technique for high-Reynolds-number flows.

ACKNOWLEDGMENTS

This work presented herein has been funded by the National Science Foundation (Grant EET-885-700, monitored by Dr. George Lea) and NASA Langley Research Center (Grant NAG-1-869, monitored by Dr. James Thomas). Calculations were carried out on IBM RS/6000 workstations which were donated by IBM. The authors gratefully

acknowledge their support. The authors also thank Dr. Klaus-Peter Beier of the University of Michigan, for his help with the parametric-spline algorithm.

REFERENCES

1. T. J. Barth, in *Computational Fluid Dynamics*, Lecture Series 1990-04 (Von Kármán Institute for Fluid Dynamics, 1990).
2. R. Löhner and P. Parikh, AIAA Paper 88-0515, 1988.
3. D. J. Mavriplis, *J. Comput. Phys.* **90**, 271 (1990).
4. A. Jameson, T. J. Baker, and N. P. Weatherill, AIAA Paper 86-0103, 1986.
5. J. F. Dannenhoffer III, "Computer-Aided Block Structuring through the Use of Optimization and Expert System Techniques," AIAA 10th Computational Fluid Dynamics Conference, 1991.
6. M. M. Rai, AIAA Paper 87-2058, 1987.
7. D. P. Young, R. G. Melvin, M. B. Bieterman, F. T. Johnson, and S. S. Samant, *J. Comput. Phys.* **92**, 1 (1991).
8. D. K. Clarke, M. D. Salas, and H. A. Hassan, *AIAA J.* **24** (1986).
9. B. Epstein, A. L. Luntz, and A. Nachshon, "Multigrid Euler Solver about Arbitrary Aircraft Configurations with Cartesian Grids and Local Refinement," AIAA 9th Computational Fluid Dynamics Conference, 1989.
10. M. J. Berger and R. J. LeVeque, "An Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries," AIAA 9th Computational Fluid Dynamics Conference, 1989.
11. R. J. LeVeque, ICASE Report 87-68, 1987.
12. K. Nakahashi, "An Automatic Grid Generator for the Unstructured Upwind Method," AIAA 9th Computational Fluid Dynamics Conference, 1989.
13. B. van Leer, *J. Comput. Phys.* **32** (1979).
14. T. J. Barth and P. O. Frederickson, AIAA Paper 90-0013, 1990.
15. P. L. Roe, *J. Comput. Phys.* **43** (1981).
16. J. G. Kallinderis and J. R. Baron, "Adaptation Methods for a New Navier–Stokes Algorithm," AIAA 8th Computational Fluid Dynamics Conference, 1987.
17. M. J. Aftosmis and N. Kroll, AIAA Paper 91-0124, 1991.
18. P. K. Sweby, *SIAM J. Numer. Anal.* **21** (1984).
19. B. van Leer, W. T. Lee, and K. G. Powell, "Sonic-Point Capturing," AIAA 9th Computational Fluid Dynamics Conference, 1989.
20. C.-H. Tai, Ph.D. thesis, University of Michigan, 1990.
21. B. van Leer, C. H. Tai, and K. G. Powell, "Design of Optimally-Smoothing Multi-stage Schemes for the Euler Equations," AIAA 9th Computational Fluid Dynamics Conference, 1989.