JOURNAL OF COMPUTATIONAL PHYSICS 110, 23–38 (1994)

A Grid Generation and Flow Solution Method for the Euler Equations on Unstructured Grids

W. KYLE ANDERSON

NASA Langley Research Center, Hampton, Virginia 23681

Received March 2, 1992; revised May 27, 1993

A grid generation and flow solution algorithm for the Euler equations on unstructured grids is presented. The grid generation scheme utilizes Delaunay triangulation and self-generates the field points for the mesh based on cell aspect ratios and allows for clustering near solid surfaces. The flow solution method is an implicit algorithm in which the linear set of equations arising at each time step is solved using a Gauss Seidel procedure which is completely vectorizable. In addition, a study is conducted to examine the number of subiterations required for good convergence of the overall algorithm. Grid generation results are shown in two dimensions for a NACA 0012 airfoil as well as a two-element configuration. Flow solution results are shown for two-dimensional flow over the NACA 0012 airfoil and for a two-element configuration in which the solution has been obtained through an adaptation procedure and compared to an exact solution. Preliminary three-dimensional results are also shown in which subsonic flow over a business jet is computed.

INTRODUCTION

The use of unstructured grids for the solution of the Euler equations offers several advantages over the use of structured grids. These advantages include the ease with which adaptive methodology can be incorporated into the flow solvers and the relatively short time to generate grids about complex configurations. Although the overall time to generate grids about complex configurations is much shorter for unstructured grids compared to block-structured grids, the computer time required for the unstructured flow solvers has historically been much longer than those of structured grids. While unstructured flow solvers will continue to require longer computer times than structured grids due to indirect addressing, recent advances [1–4] now make three-dimensional computations on unstructured grids much more competitive with those of structured grids.

As mentioned earlier, the success of unstructured grids is due in large part to the relative ease at which grids can be obtained over complex configurations. There are currently two dominant methods of generating unstructured grids. The first of these is the advancing front method in which the cells which make up the interior of the mesh are computed by marching away from the boundaries of the domain [5, 6]. This method has been used with success to generate grids about many complex configurations [7]. Further details of this method can be found in Refs. [5–8] and the references contained therein.

The other method commonly used for generation of unstructured grids, and which is emphasized in the current study, is that of Delaunay triangulation [9, 10]. This technique triangulates a given set of points in a unique way such that the minimum angle of each triangle in the mesh is maximized. This has the advantage that the resulting meshes are optimal for the given point distribution in that they do not usually contain many extremely skewed cells.

The field points for generating grids using the Delaunay triangulation approach are usually generated a priori by generating points about individual components with structured grids [11], a quadtree method [8], or by embedding the geometry into a Cartesian grid [12]. A novel approach to the generation of field points is given by Holmes [13] in which the field points are generated as the triangulation proceeds based on the aspect ratio and cell area of current triangles. This technique generates grids with little skewness since new points are introduced to continually reduce the cell aspect ratios. Unfortunately, grids generated in this manner tend to be too coarse to be used for obtaining accurate flowfield solutions without adaptation.

In the present study, an approach similar to that of Holmes is used and an extension is incorporated which automatically adds new nodes to cluster points in regions of interest. Using the new generator, grids around complex, multi-body configurations are efficiently generated which are suitable for computations.

Many advances have also been made in flow solvers for obtaining flowfield solutions on unstructured grids. Impressive results have been obtained by Mavriplis in Ref. [2] in which solutions are obtained for a wing configuration using a node-based, central differencing scheme with multigrid to achieve rapid convergence. In this reference, solutions on a three-dimensional grid consisting of over two million cells are obtained in about one hour.

0021-9991/94 $6.00

For upwind solvers, Frink [1] has generated results for many steady state applications using a cell-centered, multistage time-stepping scheme and Roe's approximate Riemann solver [14]. For unsteady applications, Batina [15] has developed both explicit and implicit algorithms for obtaining aeroelastic applications while Rausch [16] has coupled some of these methods with adaptive mesh refinement.

In the current study, an implicit algorithm is described for solving the Euler equations. This method is based on the backward Euler time differencing scheme as is the work in Refs. [15, 17], but it is formulated in a manner which permits full vectorization. In addition, the number of subiterations necessary to sufficiently solve the linear problem and to obtain the best convergence rate is examined. Results are shown for both two- and three-dimensional calculations.

TWO-DIMENSIONAL GRID GENERATION

Delaunay Triangulation

The foundation of the proposed grid generation procedure is the Delaunay triangulation procedure described in detail in Ref. [9]. This technique triangulates a set of points by inserting each point one at a time into a current triangulation so that no vertex from one triangle will lie within the circumcircle of any other triangle. This is accomplished by first identifying all the cells whose circumcircle encloses the point which is to be inserted. An example is shown in Fig. 1 in which the point to be inserted lies within the circumcircle of two triangles. The Delaunay cavity, shown in Fig. 2, is then formed from the union of all the triangles identified above. At this stage, a new triangulation is made by simply connecting the new point to each of the nodes lying on the boundary of the Delaunay cavity as depicted in Fig. 3.

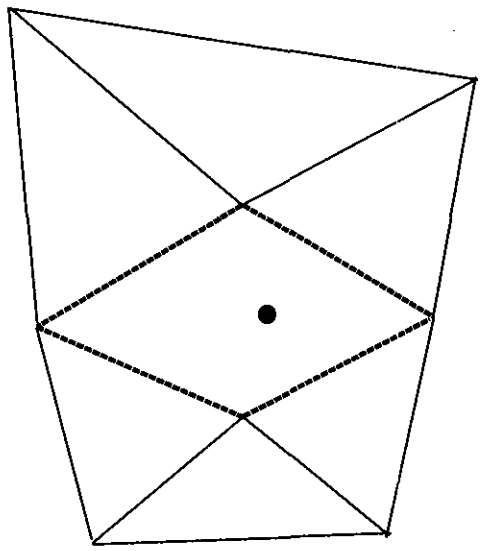

FIG. 2. Delaunay cavity.

For generating grids about arbitrary two-dimensional configurations, an initial triangulation is first formed which simply consists of a square divided into two triangles and whose four corner points are located at sufficient distance from all solid surfaces. The points which define the solid surfaces are then inserted followed by a predetermined number of far field points which are located in a circular pattern a specified radius from the center of the bodies. The cells which make up the interior of the body are then identified according to whether the center of each cell is located inside or outside of one of the bodies.

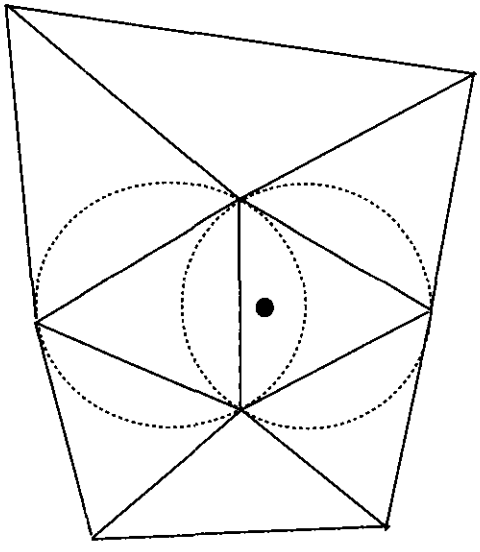

FIG. 1. Identifying cells broken by introducing a new point.

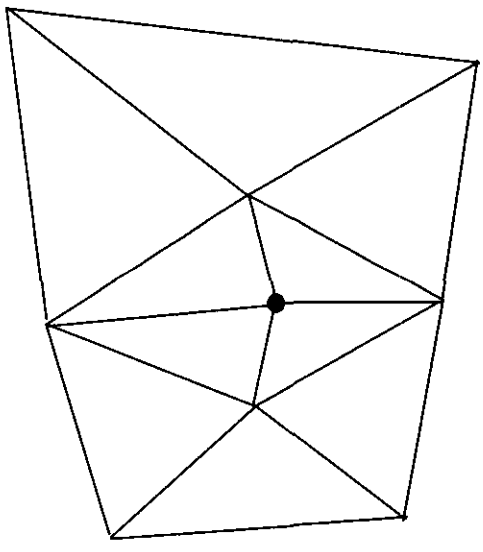

FIG. 3. Reconnection of grid after inserting a new point.

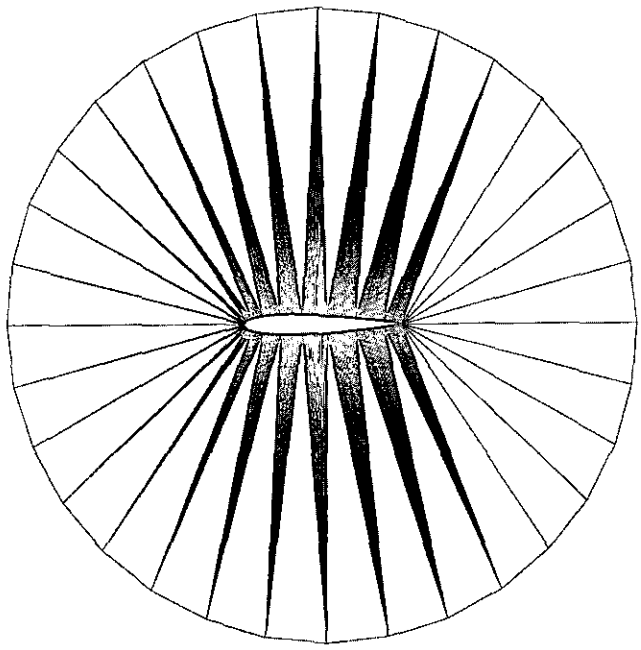

FIG. 4. Initial sample grid around NACA 0012.

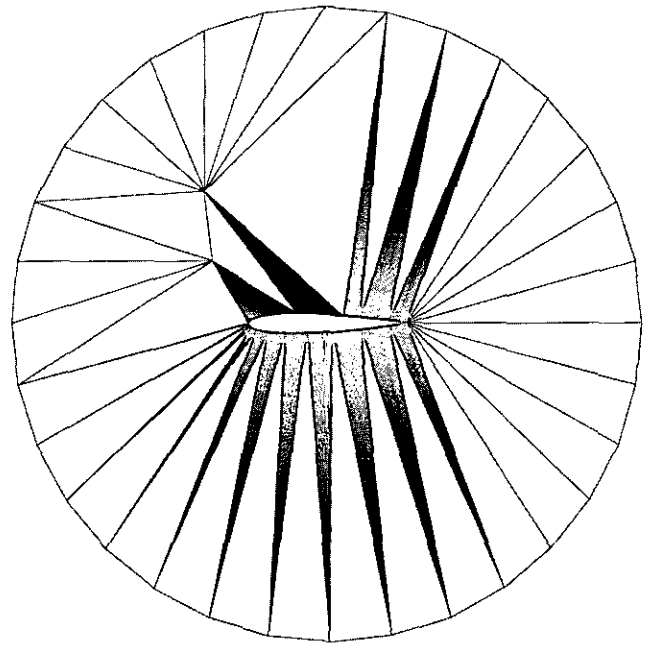

FIG. 6. Sample grid around NACA 0012 after inserting two points.

After this initial phase of the process, a loop is conducted over all of the cells and a new point is immediately introduced at the center of the circumcircle of any triangle whose aspect ratio (defined as the ratio of the circumcircle radius to twice the incircle radius) exceeds a predetermined tolerance which is generally about 1.5. Surface integrity is maintained by rejecting any point that would result in breaking of the cells which make up the interior of the airfoil [10]. Note that when a cell aspect ratio is larger than the tolerance, the new point is immediatetely added into the existing triangulation. This prevents duplicate points from being added due to two triangles whose points define the same circumcircle.

Also note that immediately adding points in this manner eliminates searching which is otherwise necessary in order to identify the first triangle which is broken by the addition of the current point. This is because the cell that has been identified as having an aspect ratio greater than the

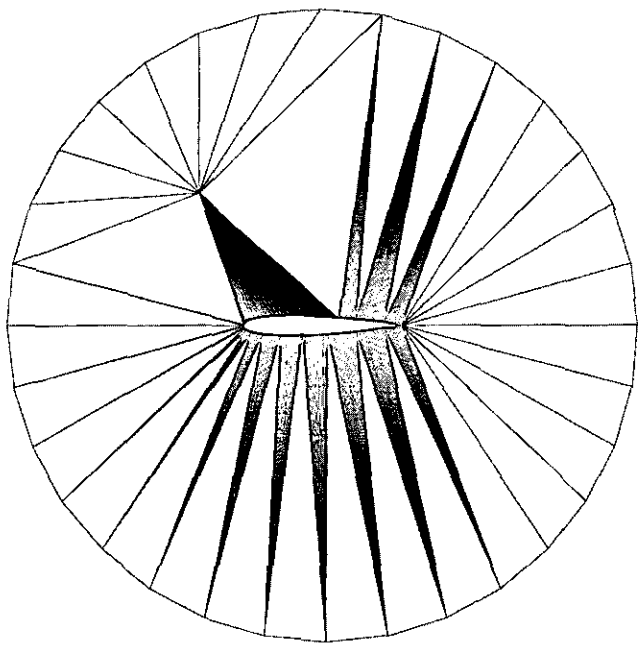

FIG. 5. Sample grid around NACA 0012 after inserting one point.

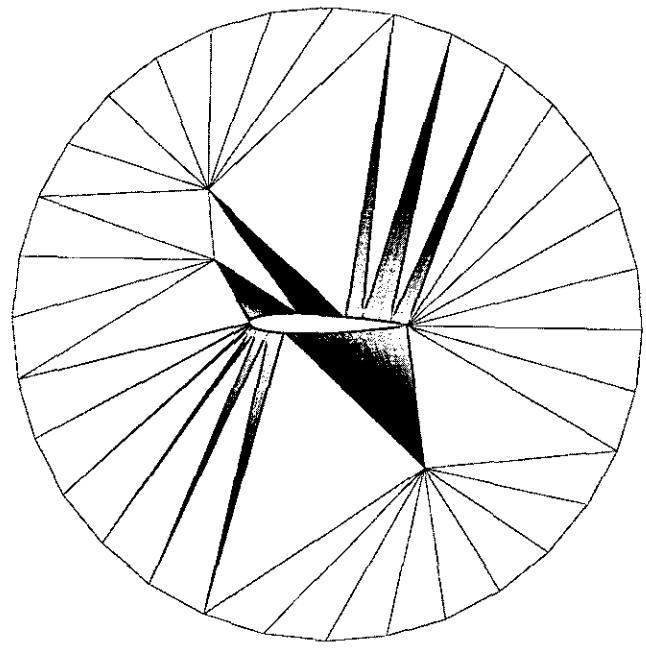

FIG. 7. Sample grid around NACA 0012 after inserting three points.

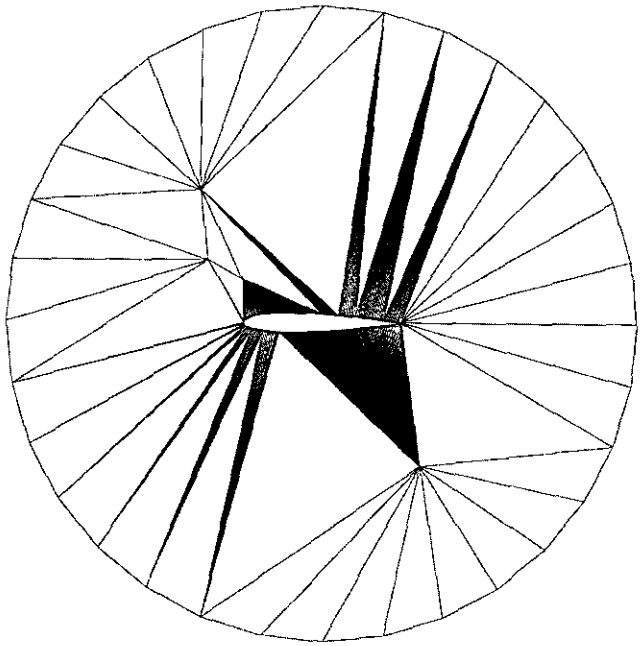

FIG. 8. Sample grid around NACA 0012 after inserting four points.

tolerance will also correspond to one of the triangles that form the Delaunay cavity. Since no searching is required, the computer time required to generate the field points in this manner is very small. The addition of new points in this manner is similar in concept to that of Ref. [13] in which new points are introduced based on both cell area and aspect ratio.

An example of this process is shown in Figs. 4 thru 9 for a sample grid around a NACA 0012 airfoil. The airfoil surface is defined with 200 points along the surface and 32 points placed around the outer boundary. Note that the outer boundary is placed reasonably close to the airfoil for illustrative purposes, allowing the entire grid to be seen. Figure 4 shows the initial triangulation in which only the surface points and the outer boundary points have been included and has cells with aspect ratios as high as 160.

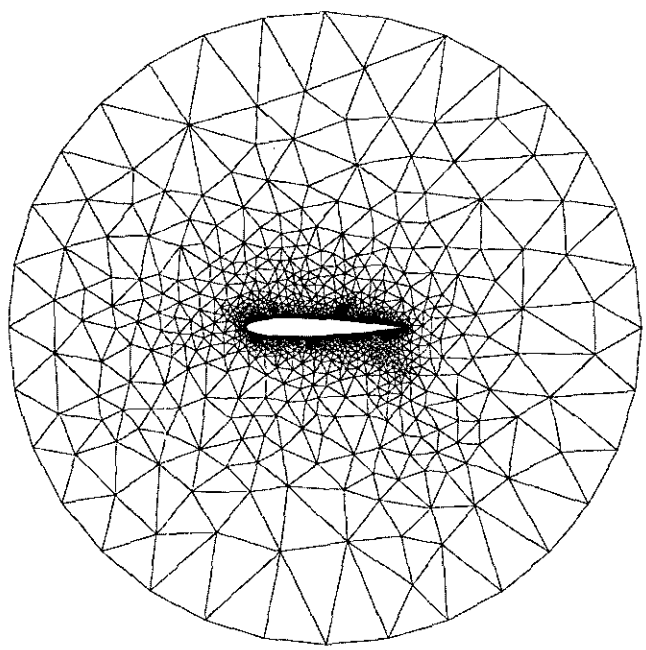

FIG. 9. Final sample grid around NACA 0012 with all aspect ratios <1.5.

New points are now introduced at the center of the circumcircle of any cell whose aspect ratio exceeds 1.5. Figures 5, 6, 7, and 8 show a few of the intermediate triangulations after inserting the first, second, third, and fourth points, respectively.

The final grid obtained by adding field points in this manner is shown in Fig. 9. It consists of 1328 nodes, 3748 faces, 2420 cells and has a maximum aspect ratio of 1.495. Although all the resulting cells are close to equilateral, it is apparent that the grids generated with this technique are relatively coarse a short distance from the airfoil and should not be expected to be sufficient for accurate computations. It is therefore necessary to increase the grid density in the vicinity of the airfoil.

Extensions for Clustering Mesh Points

In order to add new points in the vicinity of the airfoil, a value is first assigned to each existing cell which is the product of the cell area and a weighting function which decreases as the distance from the cell center to a solid surface is increased, i.e.,

$$\phi(A, d) = A \times f(d). \tag{1}$$

In this equation, d is the distance from the cell center to the nearest node lying on a solid boundary. This variable will be used to add subsequent points in cells in which the deviation of ϕ from the average is larger than the standard deviation. For this reason, the average and standard deviation of this variable are first computed:

$$\bar{\phi} = \frac{1}{N} \sum_{i=1}^{N} \phi_i \tag{2}$$

$$\sigma = \sqrt{\sum_{i=1}^{N} (\bar{\phi} - \phi_i)^2 / N}. \tag{3}$$

A list of new points that will be inserted into the existing grid is then constructed from the cell centers of all triangles whose local value of $\phi(A, d)$ exceeds that of the average plus the standard deviation, i.e., whenever $\phi_i \geq \bar{\phi} + \sigma$. The list of new points is then introduced into the existing triangulation as before. By adding new points in this manner, the function ϕ tends to be evenly distributed over the grid and new points are added at larger cells near the body first, and few (if any) new points are introduced far from solid surfaces.

The weighting function used in the current study is given by

$$f(d) = \frac{1}{1 + e^{\beta(d - d_0)}}. \tag{4}$$

In this equation, d_0 is a distance measured from the surface inside of which clustering will tend to occur. A plot of this function is shown in Fig. 10 for several values of β and $d_0 = 0.5$. As seen, the transition of this function at $d = 0.5$ steepens as β increases and the value decreases as the distance from the airfoil increases. In this manner, the transition between clustered and non-clustered regions can be made smoothly and the distance away from the airfoil in which clustering occurs is also controlled. It should be noted that since this procedure only adds one point at the center of each triangle, the amount of clustering for the final grid (i.e., how many new points are introduced) is increased by repeating the above procedure several times. In practice, it has been determined that three or four repetitions in which β is gradually increased leads to grids with good clustering near the surface of the airfoils and a reasonably smooth transition region between the clustered and non-clustered areas is obtained. Further enhancements to the above procedure may be achieved by varying the weighting function.

The final step in the grid generation procedure is to smooth the grid with simple Laplacian-type smoothing as given in Ref. [18]. This is achieved by repositioning the mesh points according to

$$\begin{aligned} x_i^{n+1} &= x_i^n + \frac{\omega}{n} \sum_{k=1}^{n} (x_k - x_i) \\ y_i^{n+1} &= y_i^n + \frac{\omega}{n} \sum_{k=1}^{n} (y_k - y_i), \end{aligned} \tag{5}$$

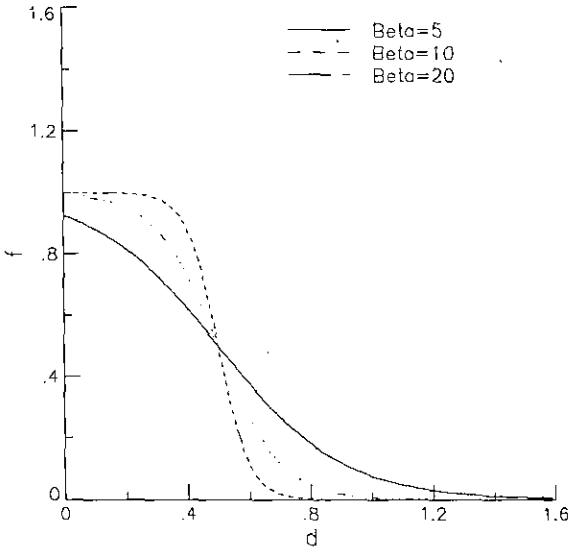

FIG. 10. Weighting function for several values of β and $d_0 = 0.5$.

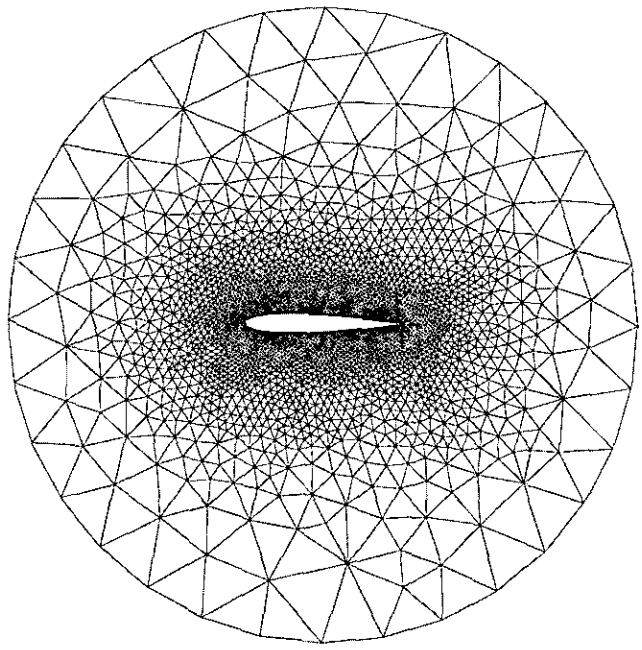

FIG. 11. Final sample grid around NACA 0012.

where ω is a relaxation factor and the sum is over all edges meeting at node i. For the current study, a relaxation factor of 0.2 is typically used and 100–200 iterations of smoothing are performed.

The final sample grid for the NACA 0012 airfoil is shown in Fig. 11. This grid demonstrates the success of the clustering procedure and is a clear improvement to the grid previously shown in Fig. 9.

EULER SOLVER

The Euler flow solver is an implicit, cell-centered, upwind-differencing code in which the fluxes on cell faces are obtained using the Van Leer flux-vector-splitting [19]. The solution at each time step is updated using an implicit algorithm which uses the linearized backward-Euler, time-differencing scheme. At each time step, the linear system of equations is solved with a subiterative procedure in which the cells in the mesh are divided into groups (colors) so that no two cells in a given group share a common edge. For each subiteration, the solution is obtained by solving for all the unknowns in a given color before proceeding to the next color. Since the solution of the unknowns in each group depends on those from previous groups, a Gauss–Seidel type of procedure is obtained which is completely vectorizable.

The choice of a cell-centered scheme over a node-based scheme for the Euler equations is based on two observations. First, since the number of unknowns on a given mesh

is greater for the cell-centered scheme, the accuracy obtained will be increased over that of a node-based scheme. This is due to the fact that the increased number of unknowns in the mesh results in a decreased spacing between unknowns and hence lower truncation error. The increase in accuracy is obtained at a slightly greater expense and, whether or not the accuracy gained offsets the extra work required, depends on many factors such as the details of the implementation and the type of computer used. Second, since the scheme used for updating the solution involves an iterative procedure at each time step to obtain an approximate solution to a linear system of equations, it is beneficial to efficiency on a vector computer that the number of off-diagonal terms for each cell is constant. However, it should be noted that the suitability of a cell centered scheme for Navier–Stokes computations is not clear due to uncertainties in approximating second-order terms on highly stretched meshes. For example, it has been shown by several authors [20, 21] that in two dimensions using a node-based scheme, approximating a Laplacian using Galerkin formulas will satisfy a discrete maximum principle provided that the mesh is a Delaunay triangulation. However, there is apparently no similar analysis for cell-centered schemes.

Governing Equations

The governing equations are the time-dependent Euler equations, which express the conservation of mass, momentum, and energy for an inviscid gas. The equations are given by

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{1}{A}\oint_{\Omega} \vec{\mathbf{F}} \cdot \hat{\mathbf{n}}\, d\Omega = 0, \tag{6}$$

where the state vector $\mathbf{Q}$ and the flux vectors $\vec{\mathbf{F}}$ are given as

$$\mathbf{Q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} \tag{7}$$

$$\vec{\mathbf{F}} \cdot \hat{\mathbf{n}} = \hat{\mathbf{F}} = \begin{bmatrix} \rho U \\ \rho U u + \hat{n}_x p \\ \rho U v + \hat{n}_y p \\ (E + p)\, U \end{bmatrix}, \tag{8}$$

where U is the velocity in the direction of the outward pointing unit normal to a cell face

$$U = \hat{n}_x u + \hat{n}_y v. \tag{9}$$

The equations are closed with the equation of state for a perfect gas

$$p = (\gamma - 1)[E - \rho(u^2 + v^2)/2]. \tag{10}$$

Flux-Vector and Residual Calculation

For the computations shown in this report, the flux vectors in Eq. (8) are upwind differenced using the flux-vector-spliting technique of van Leer [19]. These flux vectors are given in terms of the Mach number normal to the cell face defined as $M_n = U/a$. For supersonic flow in the direction of a face normal, ($M_n \geqslant 1$),

$$\hat{\mathbf{F}}^{+} = (\vec{\mathbf{F}} \cdot \hat{\mathbf{n}})^{+} = \mathbf{F}, \qquad \hat{\mathbf{F}}^{-} = (\vec{\mathbf{F}} \cdot \hat{\mathbf{n}})^{-} = 0, \tag{11}$$

whereas for supersonic flow in the opposite direction of the face normal ($M_n \leqslant -1$),

$$\hat{\mathbf{F}}^{-} = (\vec{\mathbf{F}} \cdot \hat{\mathbf{n}})^{-} = \mathbf{F}, \qquad \hat{\mathbf{F}}^{+} = (\vec{\mathbf{F}} \cdot \hat{\mathbf{n}})^{+} = 0. \tag{12}$$

For subsonic flow ($|M_n| < 1$), the fluxes are split into two contributions, $\hat{\mathbf{F}}^{+}$ and $\hat{\mathbf{F}}^{-}$, such that the Jacobian matrix of $\hat{\mathbf{F}}^{+}$ has positive eigenvalues and the Jacobian matrix of $\hat{\mathbf{F}}^{-}$ has negative eigenvalues. The split fluxes are given by

$$\hat{\mathbf{F}}^{\pm} = (\vec{\mathbf{F}} \cdot \hat{\mathbf{n}})^{\pm} = \begin{cases} f^{\pm}_{\text{mass}} \\ f^{\pm}_{\text{mass}}\{[\hat{n}_x(-U \pm 2a)/\gamma] + u\} \\ f^{\pm}_{\text{mass}}\{[\hat{n}_y(-U \pm 2a)/\gamma] + v\} \\ f^{\pm}_{\text{energy}}, \end{cases} \tag{13}$$

where

$$f^{\pm}_{\text{mass}} = \pm \rho a (M_n \pm 1)^2/4 \tag{14}$$

and

$$f^{\pm}_{\text{energy}} = f^{\pm}_{\text{mass}} \left[\frac{(1-\gamma)\, U^2 \pm 2(\gamma - 1)\, Ua + 2a^2}{(\gamma^2 - 1)} + \frac{(u^2 + v^2)}{2} \right]. \tag{15}$$

The steady state residual, given by

$$R = -\oint_{\Omega} \vec{\mathbf{F}} \cdot \hat{\mathbf{n}}\, d\Omega \tag{16}$$

is calculated using trapezoidal integration by summing the fluxes over each of the faces that make up the control volume. For example, the residual in a triangular cell is calculated as

$$R = -\oint_{\Omega} \vec{\mathbf{F}} \cdot \hat{\mathbf{n}}\, d\Omega = -\sum_{i=1}^{i=3} (\hat{\mathbf{F}}^{+}(\mathbf{Q}_i^{-}) + \hat{\mathbf{F}}^{-}(\mathbf{Q}_i^{+}))\, l_i. \tag{17}$$

Here, $\hat{\mathbf{F}}^{\pm}(\mathbf{Q}^{\mp})$ represents the split fluxes on the cell faces formed from an upwind interpolation of the data to each face. For first-order accurate differencing, the data on the face is obtained from the data in the center of the cells on each side of the cell face. For higher-order differencing, the primative variables are extrapolated to cell faces using a Taylor series expansion about the center of the cell so that the data on the face is given by

$$\mathbf{q}_{\text{face}} = \mathbf{q}_{\text{center}} + \nabla\mathbf{q} \cdot \vec{\mathbf{r}}, \tag{18}$$

where $\vec{\mathbf{r}}$ is the vector extending from the center of the cell to the center of the cell face.

For evaluating the gradient, $\nabla\mathbf{q}$, the data is first interpolated to the nodes using inverse distance weighting and the gradient is then evaluated using Greens theorem. This method is that of Frink and is discussed in further detail in Ref. [22]. It should be noted that obtaining the data at the nodes has also been accomplished using a linear least squares fit of the data in the surrounding cells with no apparent differences observed in the solutions obtained with either method.

Boundary Conditions

Since the current scheme is a cell-centered scheme, information from the interior of the mesh must be utilized along with physical constraints in order to evaluate the fluxes on the boundaries. The flow variables on the body are set according to characteristic type boundary conditions similar to those in Ref. [23]. The density, pressure, and velocity components on the body are set according to

$$p_{\text{body}} = p_{\text{ref}} + \rho_{\text{ref}} a_{\text{ref}} (\hat{n}_x u + \hat{n}_y v) \tag{19}$$

$$\rho_{\text{body}} = \rho_{\text{ref}} + (p_{\text{body}} - p_{\text{ref}})/a_{\text{ref}}^2 \tag{20}$$

$$u_{\text{body}} = u_{\text{ref}} - \hat{n}_x (\hat{n}_x u + \hat{n}_y v)_{\text{ref}} \tag{21}$$

$$b_{\text{body}} = v_{\text{ref}} - \hat{n}_y (\hat{n}_x u + \hat{n}_y v)_{\text{ref}} \tag{22}$$

from which the energy is set using the equation of state given in Eq. (10). The reference conditions for Eqs. (19) thru (22) are taken from the first cell in the interior of the grid. Other procedures have been used such as setting the flux on the surface using only the pressure on the body which has been set to be identical to that in the adjoining cell with little difference observed in the results.

Since an implicit scheme is used in the current study, implicit bounary conditions are implemented by assuming that

$$\Delta\rho_{\text{body}} = \Delta\rho_{\text{ref}} \tag{23}$$

$$\Delta(\rho u)_{\text{body}} = \Delta(\rho u)_{\text{ref}} - \hat{n}_x (\hat{n}_x \Delta(\rho u) + \hat{n}_y \Delta(\rho v))_{\text{ref}} \tag{24}$$

$$\Delta(\rho v)_{\text{body}} = \Delta(\rho v)_{\text{ref}} - \hat{n}_y (\hat{n}_x \Delta(\rho u) + \hat{n}_y \Delta(\rho v))_{\text{ref}} \tag{25}$$

$$\Delta E_{\text{body}} = \Delta E_{\text{ref}}. \tag{26}$$

In this manner, the entries in the matrix which correspond to cells lying adjacent to a solid surface can be easily modified to include the influence of the boundary.

For the far field, explicit boundary conditions are used in which the velocity and speed of sound are obtained from two locally one-dimensional Riemann invariants given by

$$R^{\pm} = U \pm \frac{2a}{\gamma - 1}. \tag{27}$$

These invariants are considered constant along characteristics defined normal to the outer boundary. For subsonic conditions at the boundary, R^- can be evaluated locally from free-stream conditions outside the computational domain and R^+ is evaluated locally from the interior of the domain. The local normal velocity and speed of sound on the boundary are calculated using the Riemann invariants as

$$U_{\text{boundary}} = \tfrac{1}{2}(R^+ + R^-) \tag{28}$$

$$a_{\text{boundary}} = \frac{\gamma - 1}{4}(R^+ - R^-). \tag{29}$$

The Cartesian velocities are determined on the outer boundary by decomposing the normal and tangential velocity vectors into components yielding

$$\begin{aligned} u_{\text{boundary}} &= u_{\text{ref}} + \hat{n}_x (U_{\text{boundary}} - U_{\text{ref}}) \\ v_{\text{boundary}} &= v_{\text{ref}} + \hat{n}_y (U_{\text{boundary}} - U_{\text{ref}}), \end{aligned} \tag{30}$$

where the subscript "ref" represents values obtained from one point outside the domain for inflow and from one point inside the domain for outflow.

The entropy is determined using the value from either outside or inside the domain, depending on whether the boundary is an inflow or outflow boundary. Once the entropy is known, the density on the far field boundary is calculated from the entropy and the speed of sound as

$$\rho_{\text{boundary}} = \left(\frac{a_{\text{boundary}}^2}{\gamma S_{\text{boundary}}} \right)^{1/(\gamma - 1)}. \tag{31}$$

The energy is then calculated from the equation of state.

Time Advancement Scheme

IMPLICIT ALGORITHMS. The starting point for the time advancement algorithm is the linearized backward-Euler time differencing scheme which yields a system of linear equations for the solution at each step given by

$$[\mathbf{A}]^n \{\Delta\mathbf{Q}\}^n = \{\mathbf{R}\}^n, \tag{32}$$

where

$$[\mathbf{A}]^n = \frac{A}{\Delta t} I + \frac{\partial \mathbf{R}^n}{\partial \mathbf{Q}}. \tag{33}$$

The solution of Eq. (32) can, in principle, be obtained by a direct inversion of $[\mathbf{A}]^n$ and has the advantage that if the exact linearization of $\mathbf{R}^n$ is used in forming $[\mathbf{A}]^n$, the resulting scheme becomes Newton iteration in the limit as the time step approaches infinity. Although this technique is quite successful in two dimensions [24], the solution at each time step requires a great deal of memory to store the components of $[\mathbf{A}]^n$ as well as extensive computer time to perform the matrix inversions. Therefore, this approach is currently not very feasible for practical calculations in three dimensions.

Since the number of operations required to invert a matrix depends on the bandwidth of the matrix, first-order accurate approximations on the left-hand side of Eq. (32) are often utilized in order to reduce both the required storage as well as the computer time. With this simplification, consistency between the left- and right-hand sides of Eq. (32) requires that first-order approximations also be used on the right-hand side in order to achieve quadratic convergence. However, with first-order approximations on the left-hand (implicit) side and second-order on the right-hand side, this scheme remains stable for large time steps. Therefore, first-order differencing of the left-hand side with higher-order differencing on the right-hand side is considered in the present study. It should be pointed out that the use of a first-order scheme on the left-hand side is primarily useful for steady-state computations.

A sample configuration of triangles in which the cells are randomly ordered is shown in Fig. 12. The corresponding form of the matrix $[\mathbf{A}]^n$ is shown in Fig. 13, where a circle represents the non-zero entries.

Although the solution of the system of equations may be obtained through a direct inversion of $[\mathbf{A}]^n$, as previously mentioned, the need for large memory can be circumvented through the use of a variety of relaxation schemes in which the solution of Eq. (32) is obtained through a sequence of iterates in which an approximation of $\Delta\mathbf{Q}$ is continually refined.

To facilitate the derivation of these schemes, $[\mathbf{A}]^n$ is first written as a linear combination of three matrices representing the diagonal, subdiagonal, and superdiagonal terms, i.e.,

$$[\mathbf{A}]^n = [\mathbf{D}]^n + [\mathbf{M}]^n + [\mathbf{N}]^n. \tag{34}$$

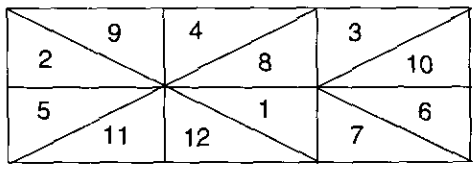

FIG. 12. Sample cell configuration.

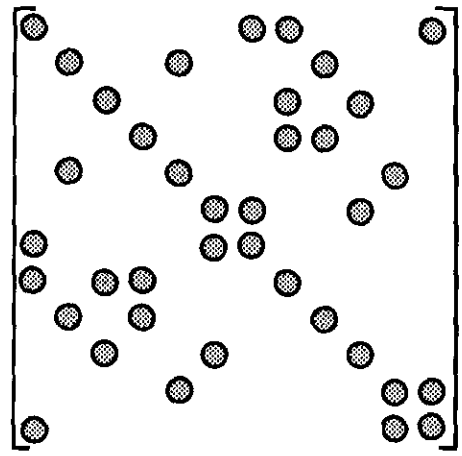

FIG. 13. Form of matrix for cells in Fig. 12.

The simplest iterative scheme for obtaining a solution to the linear system of equations is a Jacobi-type method in which all the off-diagonal terms of $[\mathbf{A}]^n \{\Delta\mathbf{Q}\}$, (i.e., $[\mathbf{M}]^n \{\Delta\mathbf{Q}\} + [\mathbf{N}]^n \{\Delta\mathbf{Q}\}$), are taken to the right-hand side of Eq. (32) and are evaluated using the values of $\{\Delta\mathbf{Q}\}^i$ from the previous subiteration level i. This scheme can be represented as

$$\begin{aligned}[\mathbf{D}]^n \{\Delta\mathbf{Q}\}^{i+1} &= [\{\mathbf{R}\}^n - [\mathbf{M}+\mathbf{N}]^n \{\Delta\mathbf{Q}\}^i] \\ &= [\{\mathbf{R}\}^n - [\mathbf{O}]^n \{\Delta\mathbf{Q}\}^i].\end{aligned} \tag{35}$$

The disadvantage of the above scheme is that the sequence of Jacobi iterations may converge somewhat slowly. In order to accelerate the convergence, a Gauss–Seidel procedure may be employed in which values of $\{\Delta\mathbf{Q}\}$ are used on the right-hand side of Eq. (35) as soon as they are available. An example of this scheme can be written as

$$\begin{aligned}[\mathbf{D}]\{\Delta\mathbf{Q}\}^{i+1} = [\{\mathbf{R}\}^n &- [\mathbf{M}]^n \{\Delta\mathbf{Q}\}^{i+1} \\ &- [\mathbf{N}]^n \{\Delta\mathbf{Q}\}^i],\end{aligned} \tag{36}$$

where the latest values of $\{\Delta\mathbf{Q}\}$ from the subdiagonal terms are immediately used on the right-hand side of the iteration equation. A slight modification to the above algorithm in which the latest values of $\{\Delta\mathbf{Q}\}$ from the superdiagonal are used results in a very similar scheme which is given by

$$\begin{aligned}[\mathbf{D}]\{\Delta\mathbf{Q}\}^{i+1} = [\{\mathbf{R}\}^n &- [\mathbf{M}]^n \{\Delta\mathbf{Q}\}^i \\ &- [\mathbf{N}]^n \{\Delta\mathbf{Q}\}^{i+1}].\end{aligned} \tag{37}$$

Yet another variation of this algorithm can be obtained by alternating the use of Eq. (36) with Eq. (37), such that a symmetric Gauss–Seidel type procedure is obtained.

Note that the algorithms given above by Eqs. (36) and (37) can both be implemented by sweeping sequentially through each mesh cell and simply using the latest values of $\{\Delta \mathbf{Q}\}$ for all the off-diagonal terms which have been taken to the right-hand side. This can be represented as

$$[\mathbf{D}]\{\Delta \mathbf{Q}\}^{i+1} = [\{\mathbf{R}\}^n - [\mathbf{O}]^n \{\Delta \mathbf{Q}\}^{\substack{i+1 \\ i}}], \quad (38)$$

where $\mathbf{Q}^{\substack{i+1 \\ i}}$ is the most recent value of $\mathbf{Q}$ and will be at sub-iteration level $i+1$ for the cells which have been previously updated and will be at level i for the cells which remain to be updated. The distinction between algorithm (36) and (37) comes about by sweeping forward through the cells (Eq. (36)) or backward through the cells (Eq. (37)).

There are two disadvantages of the scheme as described above. The first disadvantage is that since the solution of each point must be obtained before proceeding to the next one, this process is not vectorizable. The second disadvantage of this scheme is that although the off-diagonal terms may be updated and immediately used on the right-hand side, the solution of the next unknown may or may not depend on previously determined quantities. For example, as can be seen from Fig. 12, when solving for unknown number two using Eq. (36), the updated value of the solution at point one is not used, so the solution for point two remains a Jacobi step.

Note that for structured grids in which the cells are ordered in a natural manner (e.g., left to right and top to bottom), the latest information will be used immediately for calculation of the next unknown. This is because the ordering of the cells produces a banded matrix with terms grouped along the diagonal. The fact that the latest obtained data is not necessarily used for updating information in unstructured grids is strictly due to the random ordering of the cells.

Therefore, an improvement to the scheme described above can be obtained by simply renumbering the cells in such a way as to group terms along the diagonal of the matrix. In this manner, the solution of each point will tend to ensure that previously updated information from the surrounding cells is used as soon as it is available. An example of this is shown in Fig. 14, where the same sample set of cells used in Fig. 12 is simply renumbered from bottom to top and left to right. The resulting form of the matrix, shown in Fig. 15 shows that the grouping along the diagonal is greatly improved. It is expected that the ordering of the cells

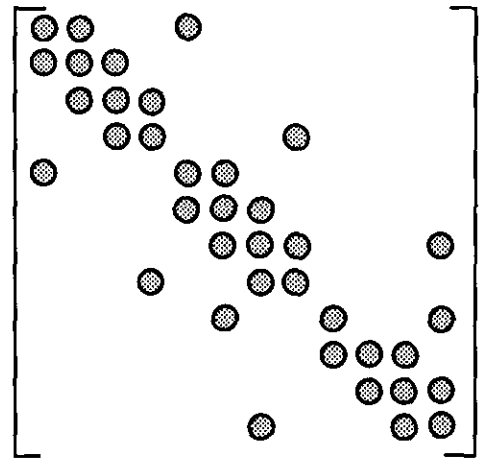

FIG. 15. Form of matrix for cells in Fig. 14.

in this way should result in somewhat faster convergence of the linear problem than a random ordering of cells. Note that although the ordering of the cells in this example groups unknowns along the diagonal, other procedures such as the Cuthill–McKee method described in Ref. [25] are more effective for general configurations. Again, it should be noted that several variations of this scheme can be obtained by using various combinations of Eqs. (36) and (37). An important disadvantage of this scheme, however, is that it still suffers from the fact that the contribution of the off-diagonal terms on the right-hand side of Eq. (38) is not vectorizable.

The Jacobi, Gauss–Seidel, and symmetric Gauss–Seidel schemes described above have all been used in practice by various researchers. An example is given in Ref. [17] where these schemes have been used to solve the Euler equations for transonic flow over a circular arc in a channel. In this reference, it is determined that the symmetric Gauss–Seidel scheme exhibited the fastest convergence rate of these three schemes. In Ref. [26] successful use of a symmetric Gauss–Seidel algorithm for transonic flow over airfoils is described in which grouping the unknowns along the diagonal is enhanced by sorting them according to the x coordinate direction.

VECTORIZATION OF GAUSS–SEIDEL. The numbering of cells used in the current study is that shown in Fig. 16. The ordering is obtained by grouping cells so that no two cells in a given group share a common edge. The resulting matrix

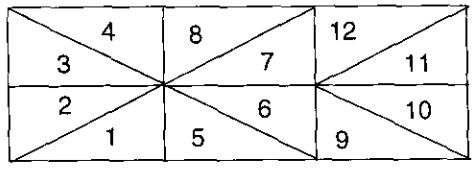

FIG. 14. Sample cells.

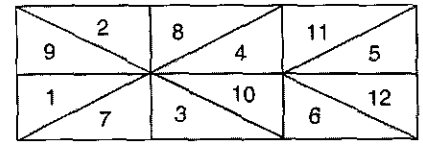

FIG. 16. Sample cells.

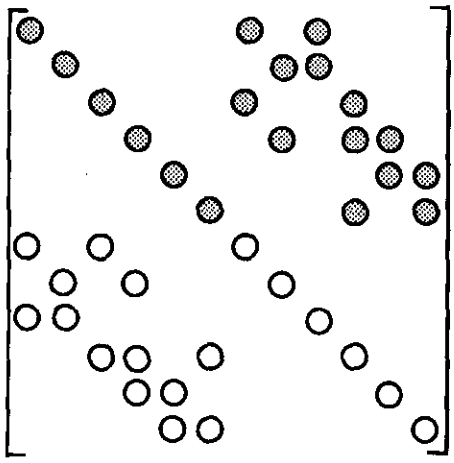

FIG. 17. Form of matrix for cells in Fig. 16.

form for [A] is given in Fig. 17. Note that for the current example, only two groups are formed; in practice, at most four groups will be formed for two-dimensional calculations and five groups are formed for three-dimensional calculations. The first group for the present example consists of the cells numbered 1 thru 6 and the second group contains cells 7 through 12.

The solution scheme can be written as before using Eq. (38) and is implemented by solving for all the unknowns in a group at a time. In this manner, the cells in the first group are solved using a Jacobi-type iteration while the cells in all the subsequent groups are obtained by using the most recently updated values of $\{\Delta \mathbf{Q}\}$ from the off-diagonal contributions. In this way, a Gauss–Seidel type scheme is obtained which is easily implemented and is fully vectorizable. Note that a symmetric Gauss–Seidel type of procedure is not necessary and is not used; stability is achieved as long as the matrix [A] maintains block diagonal dominance which occurs when first-order differencing is used on the implicit side of the equation [27].

For coloring the cells in the mesh, a simple algorithm has been used that consists of simply proceeding through a list of cells that remain to be colored and checking to see if one of its neighboring cells has already been placed in the current color. If none of the neighbors is in the current color, the cell is "tagged" and placed in the current color. Otherwise, the cell is placed in a list of uncolored cells and the process continues with the next cell in the list of uncolored cells. The resulting coloring is in no way optimal but is reasonably efficient, requiring only a few passes through the grid.

Note that in the discussions above, the exact number of subiterations required in order to sufficiently converge the linear problem (Eq. (32)) has not been specified. The number of subiterations used for each global time step has been determined through numerical experiments which will be presented in the results.

Time Step Calculation

In order to enhance the convergence to steady state, local time stepping is used. The time step calculation for each cell is given by

$$\Delta t = \mathrm{CFL}\,\frac{l}{\sqrt{u^2+v^2}+a}, \tag{39}$$

where l is a length scale for the cell defined as the area of the cell divided by the perimeter.

Results

Flow field calculations for several demonstration cases are presented below. The first case is that of an NACA 0012 at a freestream Mach number of 0.8 and an angle of attack of 1.25°. The grid has an outer boundary placed approximately 50 chord lengths away from the body and consists of 3624 nodes, 7012 cells, and 10,636 faces. A near field view of the grid is shown in Fig. 18.

The pressure coefficient distribution along the surface of the airfoil is shown in Fig. 19. As seen, a moderately strong shock is captured on the upper surface of the airfoil and a weaker shock is captured on the lower surface. Also, note that since a flux-limiter has not been used for the present calculation, an "overshoot" is evident ahead of the upper surface shock. The corresponding Mach number contours for this case are shown in Fig. 20.

For this calculation, the residual of the continuity equation has been reduced to machine zero in about 400 global iterations as seen in Fig. 21. The CFL number was started at 50 and linearly ramped to 200 over 100 iterations. The CFL numbers used for the current calculation are not necessarily optimal for the present case but have been found to give reasonably good convergence for a wide range of test problems and grid densities. The memory required corresponds to about 180 words per cell. For each global iteration, 20 subiterations have been used to solve the linear system each time resulting in a computational rate of approximately 45 μs per cell per global time step on a CRAY YMP using a single processor. This computational rate, however, depends on the number of subiterations performed. For the current study, this number is based on results of a numerical study in which the number of subiterations has been varied for a wide variety of CFL numbers. A typical plot of computer time required to obtain a four-order-of-magnitude reduction in the residual is shown in Fig. 22. This plot clearly indicates that 15–20 subiterations for each global iteration produce the fastest convergence rate. A similar study has been conducted on other grids and other cases with similar results. For this reason, between 15 to 20 subiterations are used for all cases shown in this report. While this number of subiterations has

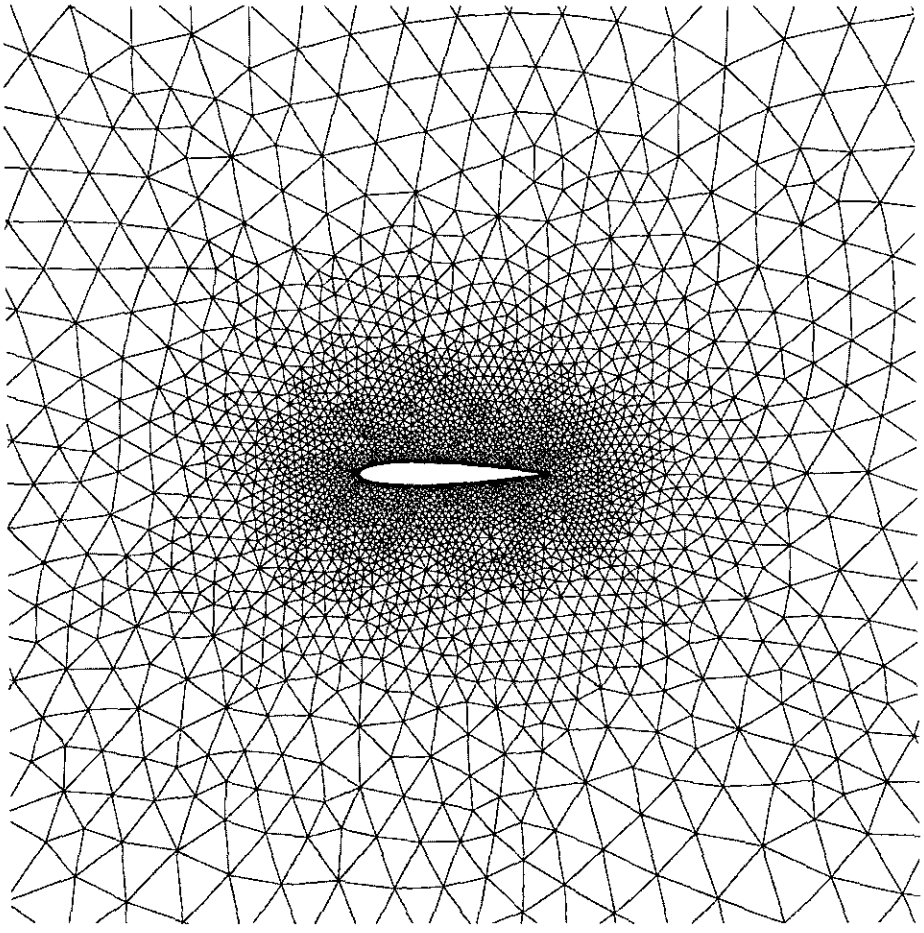

FIG. 18. Near field view of grid around NACA 0012.

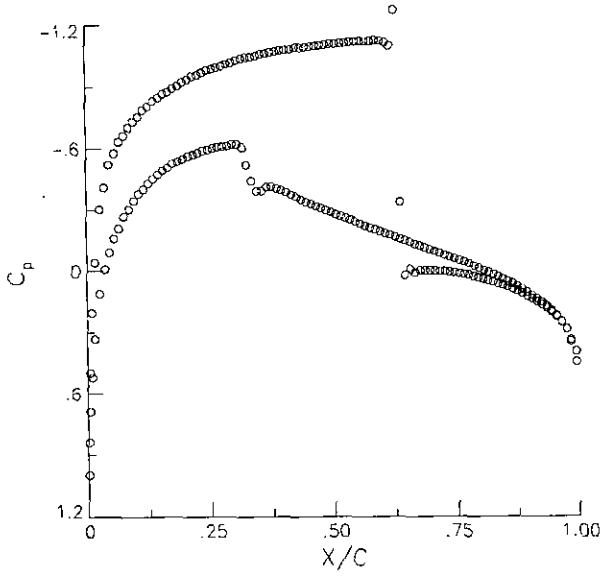

FIG. 19. Pressure distribution for NACA 0012, $M_\infty = 0.8$; $\alpha = 1.25°$.

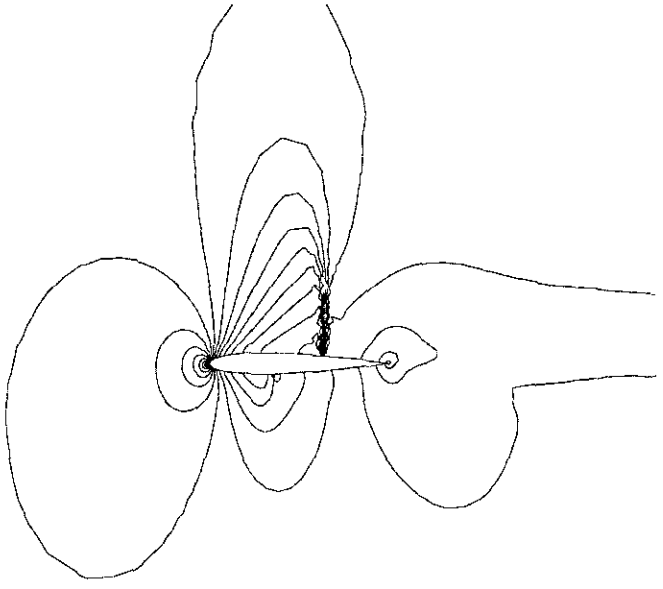

FIG. 20. Mach number contours for NACA 0012, $M_\infty = 0.8$; $\alpha = 1.25°$.

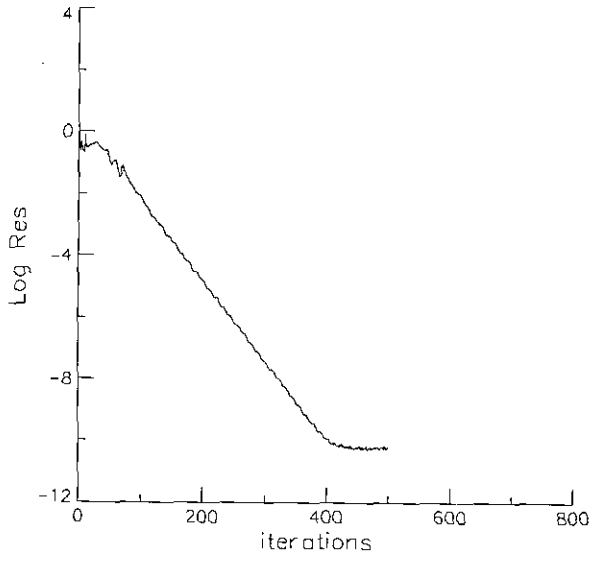

FIG. 21. Convergence history for NACA 0012, $M_\infty = 0.8$; $\alpha = 1.25°$.

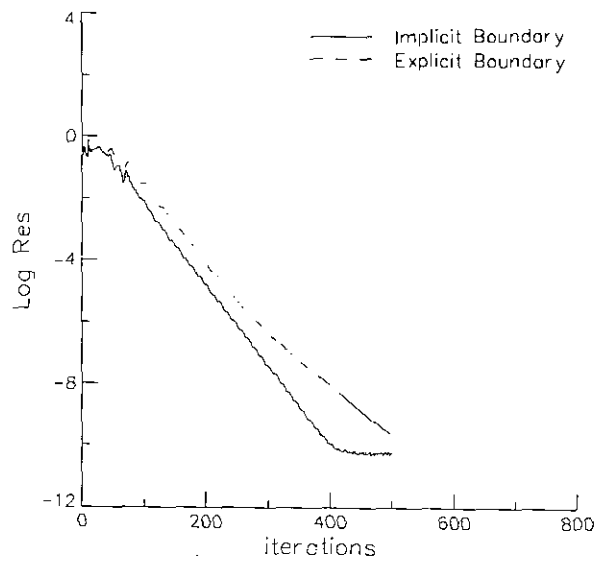

FIG. 23. Comparison of computer time for implicit and explicit boundary conditions; NACA 0012, $M_\infty = 0.8$; $\alpha = 1.25°$.

proven adequate for the current work, further work in optimizing this parameter may prove beneficial.

A comparison of convergence rates obtained with both implicit and explicit boundary conditions on the surface of the airfoil is shown in Fig. 23. As seen, the use of implicit boundary conditions produces a slightly better rate of convergence through the first several orders of magnitude reduction in the residual. In addition, the use of explicit boundary conditions seems to impede the convergence past about seven orders of magnitude. This behavior has been observed for a variety of other cases which are not presented. It is worth mentioning, however, that the use of explicit boundary conditions seems to lead to a more robust code in that ramping of the CFL number has not been necessary when explicit boundary conditions have been used. Note that for most of the calculations in this study, implicit boundary conditions have been used after five iterations and the CFL is ramped from 50 to 200.

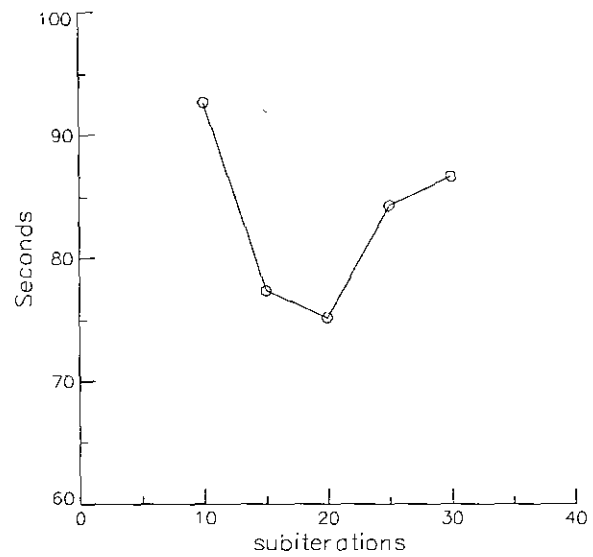

FIG. 22. Computer time for four-order reduction in residual; NACA 0012, $M_\infty = 0.8$; $\alpha = 1.25°$.

The next case presented is that of a two-element airfoil in which an exact incompressible solution exists [28]. The initial grid used for this calculation is shown in Fig. 24 and consists of 1556 points, 2882 cells, and 4439 faces. Both the main element and flap each have 100 points along the surface. Note that for this calculation, no clustering of cells has been performed near the surfaces. This is because the final solution is obtained through an adaptation procedure described in Ref. [29]. The pressure distribution calculated at a freestream Mach number of 0.2 using this initial grid is shown in Fig. 25. As seen, the coarse grid yields results which are in poor agreement with the exact solution.

As previously mentioned, a solution has also been obtained by adapting the grid to the solution. Adaptation is achieved by first identifying a list of cells requiring refinement. New points, which are located at the center of each of these cells, are then introduced into the existing triangulation and the solution is interpolated to the new grid for use in restarting the solution. Since the present flowfield does not contain discontinuities, the list of new cells is identified by flagging all cells in which the undivided velocity gradient exceeds that of the average plus the standard deviation of all the cells in the grid [29, 30].

The final grid, shown in Fig. 26, consists of 3165 nodes, 6332 cells, and 9190 faces with 148 nodes on the surface of

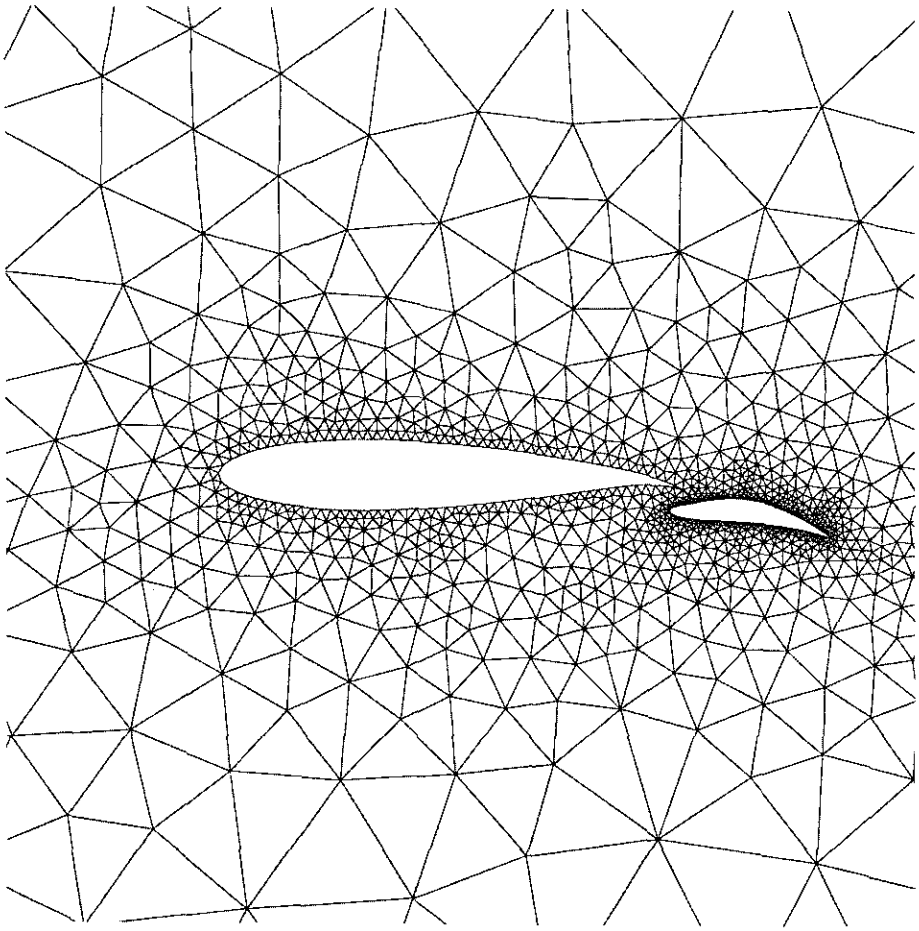

FIG. 24. Grid around two-element configuration.

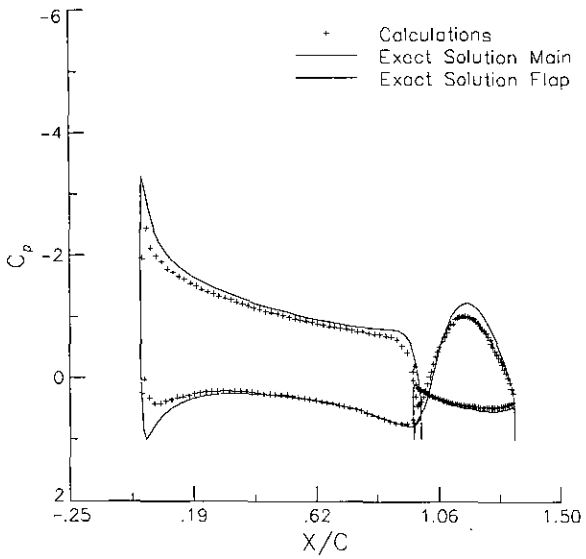

FIG. 25. Pressure distribution over two-element configuration using initial grid.

the main element and 128 nodes on the flap. The pressure distribution obtained on this grid is shown in Fig. 27. As seen, the agreement with the exact solution is much improved over that in Fig. 25. In addition, the calculated lift of 2.026 compares well with the exact value of 2.0281 given in Ref. [28].

The present algorithm has also been implemented in three-dimensions with preliminary results shown below. The case shown is that of a business jet at a freestream Mach number of 0.2 and an angle of attack of 3°. The grid used for the computations has been generated using the advancing front-type of grid generation as described in Ref. [31] and consists of 27,191 nodes, 144,100 cells, and 294,109 faces. The surface grid for this computation, which consists of 11,582 triangles, is shown in Fig. 28.

This case has been run at a constant CFL number of 300 with 15 subiterations with the convergence history for this case shown in Fig. 29. As seen, the residual is reduced between two and three orders of magnitude in 100 global iterations, at which point the convergence slows somewhat.

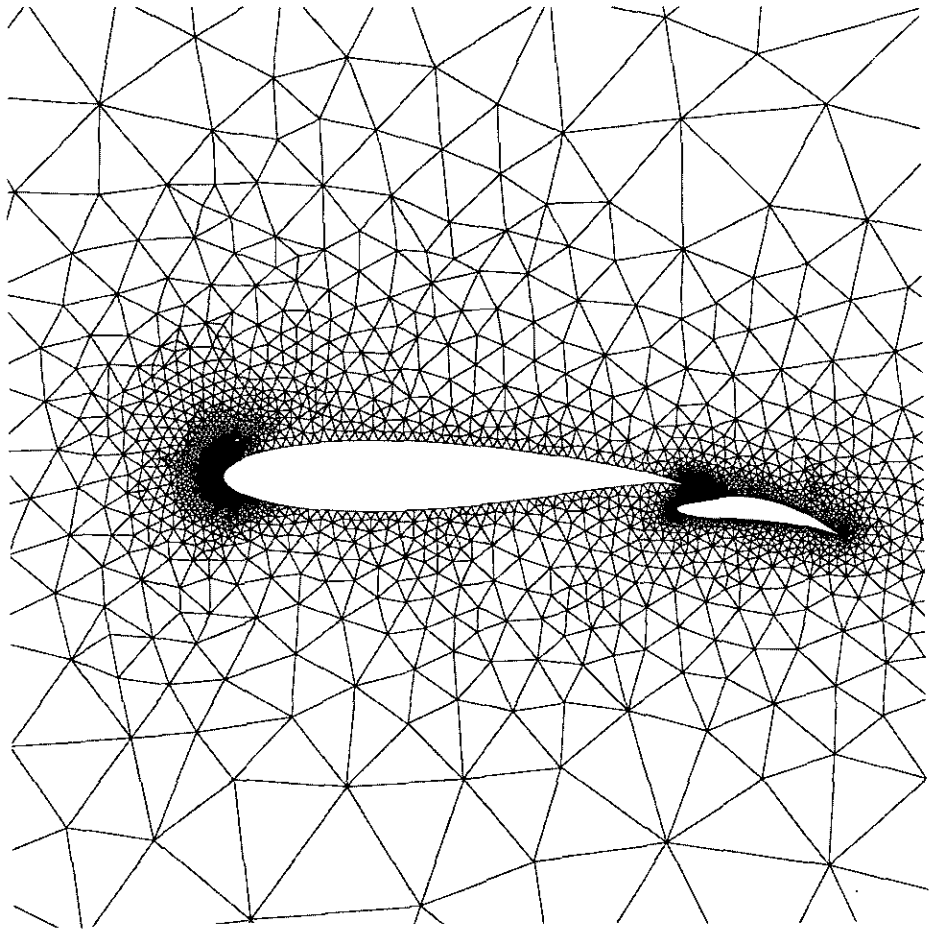

FIG. 26. Grid obtained for two-element airfoil after adaptation.

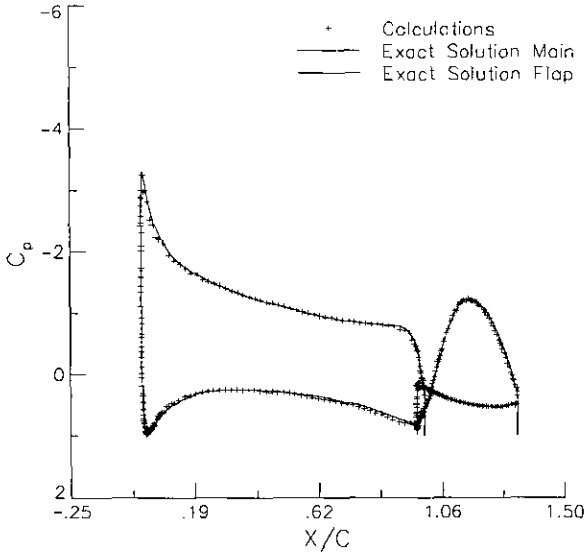

FIG. 27. Solution obtained for two-element airfoil after adaptation.

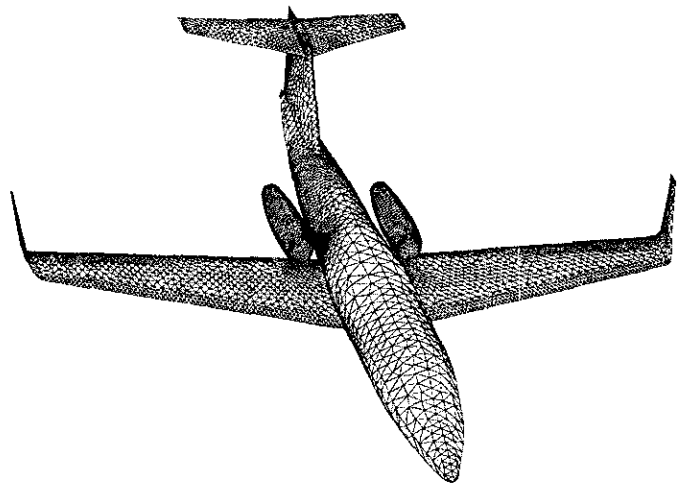

FIG. 28. Surface grid for business jet.

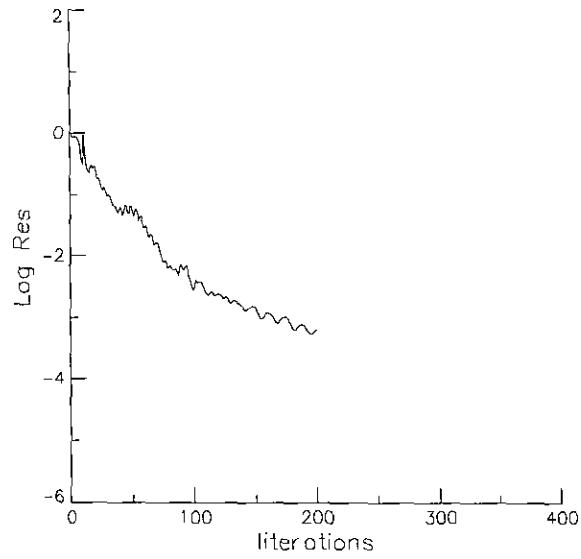

FIG. 29. Convergence history for Lear jet, $M_\infty = 0.2$; $\alpha = 3.0°$.

After 200 iterations, reduction in the residual of slightly greater than three orders of magnitude is obtained. This "tailing off" behavior has not been observed in two dimensions when implicit boundary conditions are used, but it may be due to the low freestream Mach number or the close proximity of the outer boundary which extends about 10 body lengths ahead of and behind the airplane but only about two body lengths above and below. Note that the "tailing off" of the residual may also indicate that the high frequency errors in the scheme have been effectively reduced and that the low frequency errors are beginning to dominate. The use of multigrid to rapidly eliminate these low frequency errors may enhance the convergence.

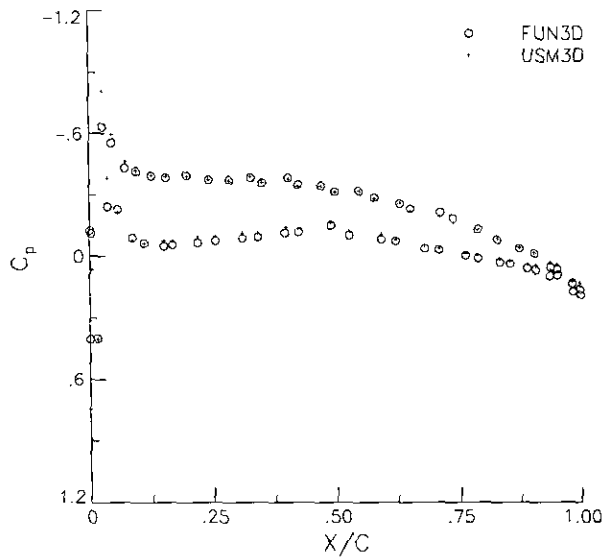

FIG. 30. Comparison of surface pressure distribution, $M_\infty = 0.2$; $\alpha = 3.0°$; $\eta = 0.44$.

A pressure distribution comparison at the $\eta = 0.44$ span station is made in Fig. 30 with the method of Ref. [1]. In Fig. 30, the results referred to as FUN3D are those of the present study while USM3D refers to those obtained using the computer code of Ref. [1] which is an upwind finite volume code which uses multistage time stepping and flux difference splitting. As seen, the comparison between the two codes are reasonably close with the main discrepencies occurring at the leading edge. These differences are due to slight differences in the computation of the boundary fluxes and because the computations with USM3D use Roes's flux-difference-splitting [14] instead of flux-vector-splitting.

The current implementation of this code in three dimensions requires about 185 words of memory per cell and the computational rate on a CRAY YMP is approximately 60 μs per cell per iteration, based on 15 subiterations per global time step.

CONCLUDING REMARKS

A two-dimensional grid generation procedure has been devised which combines automatic point placement with Delaunay triangulation to efficiently produce good quality unstructured meshes. The method uses a Delaunay triangulation algorithm and is based on the work of Holmes [13]. The present algorithm improves upon the previously cited work by allowing for the automatic generation of new mesh points so that clustering of points near surfaces is achieved.

A flow solver is also developed in both two and three dimensions which is both implicit and completely vectorizable. This scheme is based on backward-Euler time differencing for which the linear problem arising at each time step is solved using several iterations of a Gauss–Seidel type of procedure in which the unknowns are divided into groups so that no cells in a given group share an edge. In this manner, all the cells in a group are independent of each other so that their solutions can be obtained simultaneously.

The effect on convergence rate (based on computer time) of the number of subiterations is also examined. It is found that between 15 and 20 subiterations per global time step produce the best results. In addition, it is also shown that the use of implicit boundary conditions improves the convergence rate of the current algorithm.

Results are shown for two-dimensional flow over a NACA 0012 airfoil and a two-element airfoil in which the solution is obtained with adaptation. For the two-element configuration, comparisons are made with the exact solution and it is shown that excellent results are obtained by adaptation. For three dimensions, the calculation of subsonic flow over a business jet is demonstrated.

APPENDIX: SYMBOLS

$\mathbf{A}$	Matrix
A	Area of cell
a	Speed of sound
Body	Conditions on body
CFL	Courant–Friedichs–Lewy number
c	Chord length
$\mathbf{D}$	Diagonal components of $\mathbf{A}$
d	Distance to nearest surface
E	Total energy per unit volume
$\mathbf{F}$	Fluxes of mass, momentum, and energy
$\hat{\mathbf{F}}$	Fluxes normal to cell face
$\hat{\mathbf{F}}^{\pm}$	Split fluxes
$\mathbf{M}$	Components of $\mathbf{A}$ below the diagonal
M_n	Mach number normal to cell face
M_∞	Freestream Mach number
$\mathbf{N}$	Components of $\mathbf{A}$ above the diagonal
N	Total number of cells
$\hat{\mathbf{n}}$	Unit normal
n	Number of edges meeting at a node
$\hat{n}_x, \hat{n}_y$	x and y components of a unit normal
$\mathbf{O}$	All off-diagonal components of $\mathbf{A}$
p	Pressure
$\hat{\mathbf{Q}}$	Conserved state vector, $\mathbf{Q} = [\rho \ \rho u \ \rho v \ E]^T$
$\hat{\mathbf{q}}$	Primative state vector, $\mathbf{q} = [\rho \ u \ v \ P]^T$
$\mathbf{R}$	Residual for a cell
$R^{\pm}$	Riemann invariants
$\vec{r}$	Vector from center of a cell to center of an edge
ref	Denotes reference condition
S	Entropy
t	Time
U	Velocity normal to cell face
u, v	Cartesian velocities in x and y directions
x, y	Cartesian coordinates
α	Angle of attack
β	Parameter used for grid clustering
γ	Ratio of specific heats, taken as 1.4
η	Spanwise location on wing
ρ	Density
σ	Standard deviation of ϕ
ϕ	Function used for grid clustering
$\bar{\phi}$	Average value of ϕ
Ω	Boundary of cell
ω	Relaxation factor

ACKNOWLEDGMENT

The author acknowledges Daryl Bonhaus for generating the grid around the business jet.

REFERENCES

1. N. T. Frink, P. Parikh, and S. Pirzadeh, AIAA 91-0102, Jan. 1991.
2. D. J. Mavriplis, AIAA 91-1549, June 1991.
3. F. Angrand and A. Dervieux, *Int. J. Numer. Methods Fluids* **4** (1984).
4. L. Fezoui and B. Stoufflet, *J. Comput. Phys.* **84**, 174 (1989).
5. J. Peraire, M. Vahdati, K. Morgan, and O. Zienkiewicz, *J. Comput. Phys.* **72**, 449 (1987).
6. R. Lohner and P. Parikh, AIAA 88-0515, Jan. 1988.
7. P. Parikh, R. Lohner, C. Gumbert, and S. Pirzadeh, AIAA 89-0362, Jan. 1989.
8. G. S. Spragle, W. R. McGrory, and J. Fang, AIAA 91-0726, Jan. 1991.
9. A. Bowyer, *Comput. J.* **24**, No. 2 (1981).
10. T. J. Baker, AIAA 87-1124, June 1987.
11. T. J. Barth and D. C. Jespersen, AIAA 89-0366, Jan. 1989.
12. J. C. Vassberg and K. B. Dailey, AIAA 90-2998-CP, 1990.
13. D. G. Holmes and D. D. Snyder, "The Generation of Unstructured Triangular Meshes using Delaunay Triangulation," in *Numerical Grid Generation in Computational Fluid Dynamics '88* (Pineridge, Swansea, U.K., 1988), p. 643.
14. P. Roe, *J. Comput. Phys.* **43**, 357 (1981).
15. J. T. Batina, AIAA 90–1649, June 1990.
16. R. D. Rausch, J. T. Batina, and H. T. Y. Yang, AIAA 91-1106, Apr. 1991.
17. D. L. Whitaker, D. C. Slack, and R. W. Walters, AIAA 90-0697, Jan. 1990.
18. D. Mavriplis and A. Jameson, ICASE Report No. 87–53, 1987.
19. B. V. Leer, "Flux Vector Splitting for the Euler Equations," Lecture Notes in Physics, Vol. 170 (Springer-Verlag, New York/Berlin, 1982), p. 501.
20. T. J. Barth, AIAA 91-0721, Jan. 1991.
21. F. W. Letniowski, *SIAM J. Sci. Stat. Comput.* **13**(3), 765 (1992).
22. N. T. Frink, "Upwind Scheme for Solving the Euler Equations on Unstructured Tetrahedral Meshes," Workshop on Accuracy of Unstructured Grid Techniques, NASA Langley Research Center, Jan. 16–17, 1990.
23. J. M. Janus, Master's thesis, Mississippi State University, Aug. 1984.
24. V. Venkatakrishnan and T. J. Barth, AIAA 89-0364, Jan. 1989.
25. G. F. Carey and J. T. Oden, *Finite Elements; Computational Aspects, Vol. 3* (Prentice-Hall, Englewood Cliffs, NJ, 1984).
26. J. T. Batina, AIAA 90-0936, Apr. 1990.
27. W. Mulder, Ph.D. thesis, Delft University of Technology, June 1985.
28. B. R. Williams, Aeronautical Research Council, R & M 3717, 1973.
29. G. P. Warren, W. K. Anderson, J. L. Thomas, and S. L. Krist, AIAA 91-1592, June 1991.
30. Y. G. Kallinderis and J. R. Baron, *AIAA J.* **27** (1985).
31. P. Parikh, S. Pirzadeh, and R. Lohner, NASA CR-182090, 1990.