AIAA JOURNAL
Vol. 40, No. 10, October 2002

Anisotropic Solution-Adaptive Viscous Cartesian Grid Method for Turbulent Flow Simulation

Z. J. Wang*
Michigan State University, East Lansing, Michigan 48824
and
R. F. Chen†
ETA, Inc., Troy, Michigan 48083

An anisotropic viscous Cartesian grid method based on a 2^N tree data structure is developed. The method is capable of handling complex geometries automatically. In addition, viscous boundary layers can be computed with high resolution, using automatically projected high aspect ratio viscous layer grids. Compared with a widely used Octree data structure, the 2^N tree data structure supports anisotropic grid adaptations in any of the coordinate directions. Therefore, key flow features such as shock waves, wakes, and vortices can be captured in a very efficient manner. To handle the adaptive viscous Cartesian grid, an implicit, second-order, finite volume flow solver supporting arbitrary grids has been developed. A linearity-preserving least-squares solution reconstruction algorithm is used to achieve second-order accuracy. Furthermore, several directional adaptation criteria are developed and tested. The overall grid generation, flow simulation, and grid adaptation methodology is then demonstrated for a variety of flow problems, including a case of supersonic turbulent flow over a high-angle-of-attack missile configuration.

Nomenclature

C_p	=	pressure coefficient
D	=	diagonal matrix in the block lower–upper symmetric Gauss–Seidel method
$d_{\max}$	=	meshing parameter, the maximum grid size desired
$d_{\min}$	=	meshing parameter, the minimum grid size desired
$disN$	=	meshing size in the body normal direction
$disT$	=	meshing size in the body tangential direction
F	=	vector of inviscid flux
F_v	=	vector of viscous flux
G	=	geometric entity, defined as a set of points
I	=	intersection operator
I_{xx}, I_{xy}, I_{xz},...,	=	reconstruction coefficients
L	=	number of triangles on the triangulated surface
M	=	number of Cartesian front nodes
N	=	total number of cells
N_{nb}	=	number of neighboring faces sharing a node
Q	=	vector of conserved variables
$\bar{Q}$	=	vector of cell-averaged conserved variables
q	=	any of the primitive variables (density, pressure, and velocity components)
$\boldsymbol{r}$	=	position vector
S	=	a solid, defined as a set of points
S_f	=	face area of face f
V_i	=	volume of control volume i
W	=	matrix of reconstruction coefficients
w	=	weight in the Laplacian smoother
x, y, z	=	Cartesian coordinates
Δt	=	time step
π_{ix}	=	second derivative-based adaptation criterion
τ_{ix}	=	first derivative-based adaptation criterion

Received 7 September 2001; revision received 22 May 2002; accepted for publication 22 May 2002.

*Associate Professor of Mechanical Engineering, Department of Mechanical Engineering, 2555 Engineering Building; zjw@egr.msu.edu. Senior Member AIAA.

†Project Engineer.

I. Introduction

THE unstructured grid-based computational fluid dynamics (CFD) methodology has undergone considerable development in the last two decades, in terms of both grid generation and solution algorithm development. This is largely due to the difficulty in generating structured grids for complex geometries. Unstructured grids provide considerable flexibility in tackling complex geometries and in adapting the computational grids according to flow features. It is generally recognized that unstructured grid-based CFD methods offer the best promise for nearly automated fluid flow simulation. After nearly two decades of intensive development, many new types of unstructured grids have been developed besides the classical triangular or tetrahedral grids[1–5] and unstructured quadrilateral or hexahedral grids.[6,7] These new types include unstructured prismatic grids[8] and mixed grids.[9,10] Tetrahedral grids are the easiest to generate. Many well-known grid generation algorithms, such as the advancing front[11] and the Delauney triangulation method (see Ref. 12) have been developed to generate tetrahedral grids for complex geometries. However, experience has indicated that tetrahedral grids are not as efficient and/or accurate as hexahedral or prismatic grids for viscous boundary layers.[13] On the other hand, prismatic grids and hexahedral grids can resolve boundary layers more efficiently, but they are more difficult to generate than tetrahedral grids. Many CFD researchers have come to the conclusion that mixed grids (or hybrid grids) are the way to go when taking into account of both accuracy and efficiency. For example, a hybrid tetrahedral/prismatic grid approach[9] was successfully demonstrated for complex geometries, and other mixed grid methods can handle many different cell types including tetrahedrons, hexahedrals, prisms, and pyramids.[10] One disadvantage of tetrahedral grids is that tetrahedra are not as efficient as Cartesian cells in filling three-dimensional space given a certain grid resolution. This can be easily understood given that at least five tetrahedra are required to fill a cube without adding any new grid points. In addition, from a quality standpoint of view, an adaptive Cartesian grid is much less skewed than a tetrahedral grid and should present fewer numerical difficulties for stiff problems. Furthermore, there is strong evidence that prismatic grids are much more accurate and efficient than a tetrahedral for the viscous boundary layer.[13] Therefore, it seems a more appealing grid topology is a hybrid of adaptive Cartesian and viscous layer grids, either extruded prism grids or projected prism grids.

The use of Cartesian grids in solving fluid flow problems started many decades ago because it is trivial to generate the computational grids. One difficulty in using Cartesian grids is in the boundary treatment of curved geometries. Recently, there has been a renewed interest in using adaptive Cartesian grids for complex geometries.[14–19] In these Cartesian grid approaches, body surfaces (or geometries) are used to perform cell cutting to preserve the geometric fidelity. With a robust cell-cutting algorithm, the grid generation process can be completely automated. Coupled with a tree-based data structure and solution-based grid adaptation, these methods have been demonstrated to be very viable tools for inviscid flows with very complex geometry. The extension of the Cartesian grid approach to viscous flows was achieved with either the adaptive Cartesian/prism grids[20–22] or with projected viscous layer grids from the Cartesian grid[23–26] (the so-called "viscous" Cartesian grid method). This paper attempts to further develop the viscous Cartesian grid method by incorporating the capability of anisotropic grid adaptations through the use of the 2^N tree data structure[26] instead of the usual Octree. The 2^N tree data structure supports anisotropic grid adaptations of Cartesian cells naturally. The use of anisotropic grid adaptation (vs isotropic grid adaptations) offers the potential of dramatic reduction in the total number of cells to achieve a given level of solution accuracy because most high-gradient flow features such as shock waves, slip lines, vortex sheets, and wakes are anisotropic.

The paper is, therefore, arranged as follows. In the next section, we will first present the viscous Cartesian grid generation method with a 2^N tree data structure. Important steps in the grid generation process will be described. Then, an implicit, second-order, cell-centered finite volume viscous flow solver will be presented. This flow solver is capable of handling arbitrary grids, including the adaptive viscous Cartesian grid. Next, several derivative-based grid adaptation criteria will be described. These criteria are capable of identifying the directions in which the grid should be adapted. After that, several inviscid and viscous flow problems will be presented to demonstrate the capability of the method. In particular, the advantages of anisotropic grid adaptations will be showcased. Finally, conclusions will be summarized.

II. Viscous Cartesian Grid Approach Based on 2^N Tree

The first step in a CFD simulation involving a nontrivial geometry is to import the geometry, define a closed computational domain, and generate a computational grid. Generally, the grid generation process can be broken into the following steps:

1) Acquire the geometry.

2) Define a water-tight (closed) computational domain and repair the geometry if necessary.

3) Generate the computational grid on the boundaries.

4) Generate the computational grid, that is, the volume grid, in the interior of the computational domain.

In a viscous Cartesian grid method, a volume grid is first generated before a surface grid is produced through projections. A unique advantage of the method is that "dirty" geometries may be automatically handled without geometry repair. This is possible through the generalized definition of geometry, which is given hereafter.

A. Generalized Definition of Geometric Entity

In this paper, a geometric entity is defined to be any entity supporting the following two operations:

1) Given a simple solid, for example, a cube or a tetrahedron, the entity is capable of returning a status "intersected" or "nonintersected" based on whether the entity intersects the given solid. Let the geometric entity be represented by G (which is defined as a set of points) and the given solid by S. The intersection operation is $I(G, S)$ is then defined by

$$I(G, S) = \begin{cases} 1 & \text{if} \quad G \cap S \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

2) Given an arbitrary point g in space, the projection p from the given point to the entity is well defined. The line segment from the given point to the projection is the shortest distance from the given point to the entity, that is,

$$p(G, g) := |pg| \leq |rg|_{r \in G} \tag{2}$$

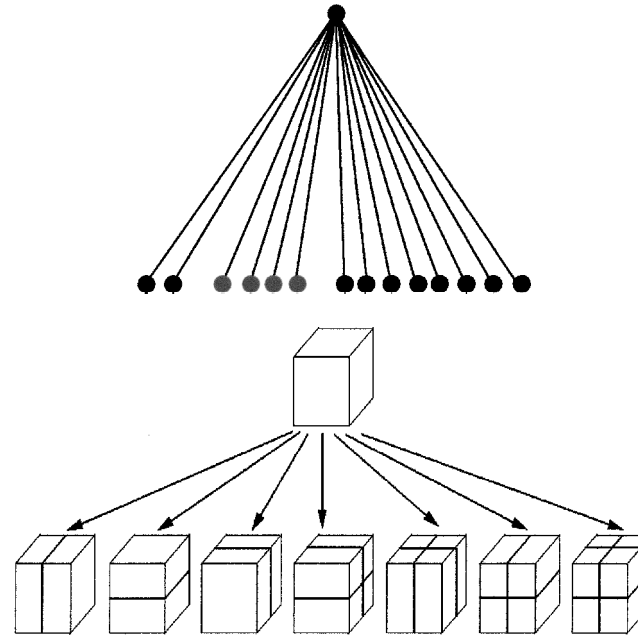

Fig. 1 Cartesian cell subdivisions supported by a 2^N tree data structure.

Note that this definition of the geometric entity is very general, and any geometry defined with a solid or a surface patch can be seen as a valid geometric entity because these operations can be easily implemented. Note that any discrete points, lines, curves, and planes are also valid geometric entities. With this definition of geometric entities, the grid generation process is presented next.

B. Adaptive Cartesian Grid Generation

Two meshing parameters, $d_{\min}$ and $d_{\max}$, are specified first. They represent the minimum and maximum sizes of Cartesian grid cells to be generated. The only requirement that the set of geometric entities must satisfy is that the computational domain formed with the entities is "physically" closed if gaps or holes smaller than $d_{\min}$ are ignored. This closure condition is much weaker than the requirement of water-tight geometry imposed by other grid generators. One of the popular data structures for adaptive Cartesian grids is the Octree. The drawback of Octree is that only isotropic grid refinement is supported. In this paper, a 2^N tree data structure is developed and implemented. The 2^N tree is a hierarchical data structure in which each nonleaf tree node can have either 2, 4, or 8 child nodes. As a result, it supports binary, Quadtree, and Octree types of subdivisions and, therefore, allows the adaptive Cartesian grid to be adapted in an anisotropic manner, as shown in Fig. 1.

The adaptive Cartesian grid is generated by recursively subdividing a single coarse root Cartesian cell. Because the root grid cell must cover the entire computational domain, the surface geometry is contained in the root cell. Because the geometric entities support the intersection operation, all of the Cartesian cells intersected by the geometry can be easily determined. The size of the Cartesian cells intersecting the geometry is controlled by two parameters, *disT* and *disN*. Parameter *disN* controls the Cartesian cell size in the geometry normal direction, whereas *disT* specifies the Cartesian cell size in the geometry tangential direction. The ratio *disT*/*disN* determines the maximum aspect ratio in the Cartesian grid. The recursive subdivision process stops when all of the Cartesian cells intersecting the geometries satisfy the length scale requirements. For the sake of solution accuracy, it is very important to ensure that the Cartesian grid is smooth. In the present study, the sizes of any two neighboring cells in any coordinate direction cannot differ by a factor exceeding two. The use of the 2^N tree data structure makes high aspect ratio Cartesian cells possible. This property can translate into considerable efficiency gains when anisotropic grid adaptations are used to resolve flow features.

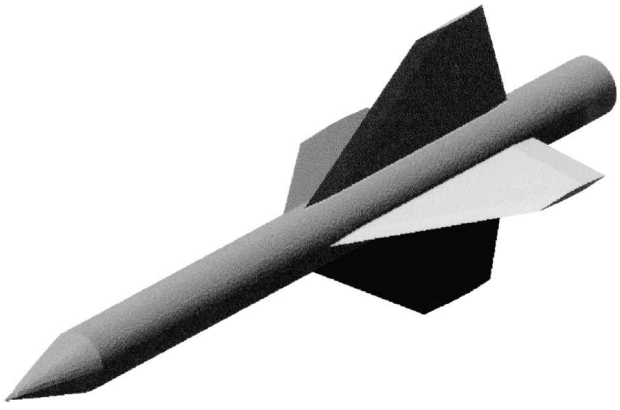

Fig. 2a Missile geometry.

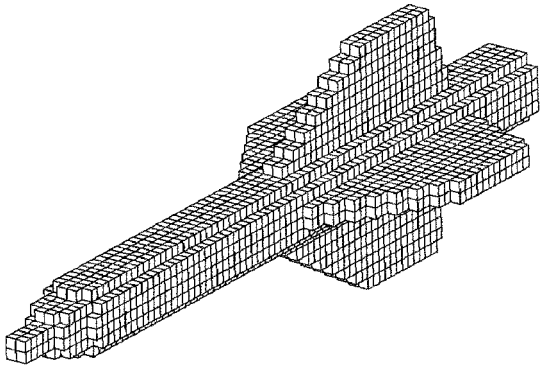

Fig. 2b Stair-step Cartesian front.

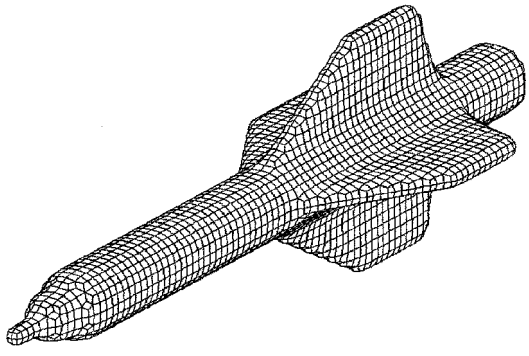

Fig. 2c Smoothed Cartesian front.

C. Cartesian Grid Front Generation and Smoothing

To insert a viscous layer grid between the Cartesian grid and the body surface, Cartesian cells intersected by the geometry must be removed, leaving an empty space between the Cartesian grid and the body surface. Once again, all of the Cartesian cells intersected by the geometry can be determined because the geometry supports the intersection operation. In addition, the intersected cells also serve to divide cells outside the geometry from the cells inside the geometry. Depending on whether the problem is external or internal, cells inside or outside the geometry must be removed. The 2^N tree is not only used to record the recursive cell subdivision process, it is also used to perform efficient intersection operations with the geometry. For example, if a (coarse) Cartesian cell does not intersect a geometric entity, all of the child cells from the Cartesian cell must not intersect the geometric entity.

Once the Cartesian cells intersected by the geometry and cells outside the computational domain are removed, we are left with a volume Cartesian grid. The boundary faces of this volume Cartesian grid form the so-called Cartesian front, and one such front generated by a missile geometry (Fig. 2a) is shown in Fig. 2b. Note that the front is not smooth and includes many sharp corners. Before this front is projected to the geometry, it is smoothed with a Laplacian smoother to produce a smoother front. One can use a very simple smoother in the following form:

$$\boldsymbol{r}_i^{\text{new}} = \boldsymbol{r}_i^{\text{old}}(1 - w) + w\frac{1}{N_{\text{nb}}}\sum \boldsymbol{r}_c \tag{3}$$

where $\boldsymbol{r}_i$ is the position vector of a node on the Cartesian front, N_{nb} is the number of Cartesian faces sharing node i, $\boldsymbol{r}_c$ are the face center position vectors of the faces sharing node i, and w is a relaxation factor in [0, 1]. The Laplacian smoothing algorithm can be applied several times to obtain a reasonably smooth front. Shown in Fig. 2c is the smoothed Cartesian front after the Laplacian smoother is applied four times with $w = 0.5$. Note that the front is much smoother than the stair-step Cartesian front shown in Fig. 2b.

To prevent the smoothed Cartesian front from intersecting the body geometry, Cartesian cells that are within a certain distance of the body are also removed. Although one cannot prove that the smoothed Cartesian front cannot intersect the body, we have not encountered a case in which any intersections are detected.

D. Projection of the Cartesian Front to the Body Surface

After the smoothed front in the Cartesian grid is obtained, each node in the front needs to be connected to the body surface to form a single layer of viscous grids. It can be proved mathematically that the projection lines cannot intersect each other away from the geometric entity. Note that, per definition, the geometric entities must support the projection operation, which comes in handy now. Because the Cartesian front is composed of boundary faces of a solid region, the front is closed and water-tight. After the front is projected to the boundary geometric entities, a water-tight surface grid is generated on the boundary. The "footprints" of the layer grids on the body surface have the same topology (or connectivity) as the Cartesian front. With this assumption, the viscous layer grids are naturally blended with the adaptive Cartesian grid, eliminating the need of cell cutting currently adopted by many Cartesian grid generators. By connecting each point on the Cartesian front and the corresponding projected point on the boundary, we obtain a single layer of prism grids. This single layer can be subdivided into multiple layers with proper grid clustering near the geometry to resolve a viscous boundary layer.

The efficiency of the projection operation in three dimensions is critical to the success of the method. In a typical application, we usually have a triangulated surface with L triangles and a Cartesian front with M nodes. A brute-force exhaustive search based projection algorithm would take $\mathcal{O}(ML)$ operations, which is too expensive even for medium-sized applications. Instead an alternating digital tree[27] is used to record the bounding boxes of the triangles. Given a node to project, only triangles close to the node are identified from the tree-based search operation and are projected to. This new algorithm reduces the number of operations from $\mathcal{O}(ML)$ to about $\mathcal{O}(M \log L)$. The speed-up for a medium-sized problem ($L = 100{,}000$ and $M = 100{,}000$) can be more than several orders of magnitude.

A projection based on the minimum distance rule usually misses geometrically important concave features, such as the corner points in Fig. 3. To preserve these features, they must be detected or specified first. One criterion is to detect all sharp edges based on the angle between the two faces sharing an edge. Then a feature-preservation technique is used to reconnect the front nodes to those features. This technique is schematically shown in Fig. 3. Example viscous Cartesian grids around the missile geometry, without and with feature preservation, are displayed in Fig. 4. Note that the fin–body intersection was heavily smeared without feature preservation but was resolved clearly with feature preservation.

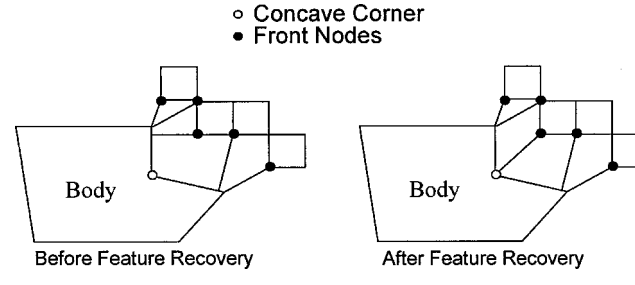

Fig. 3 Illustration of a smeared concave corner in front projection.

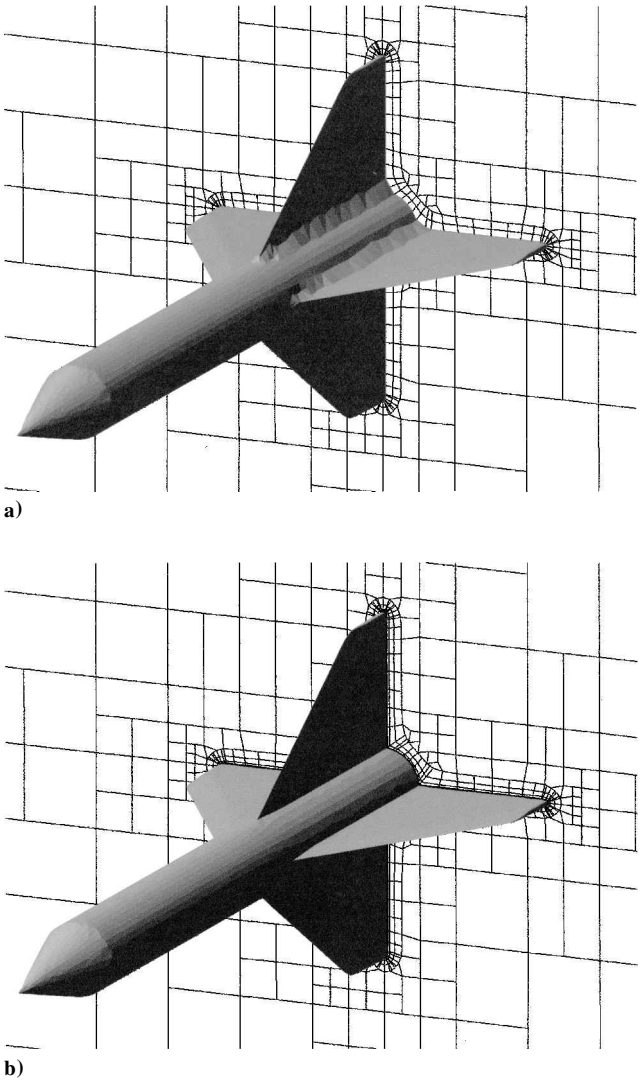

Fig. 4 Viscous adaptive Cartesian grid for a missile a) without and b) with feature preservation.

III. Finite Volume Flow Solver for Arbitrary Grids

A flow solver capable of handling arbitrary polyhedrons has been developed to uniformly handle the adaptive Cartesian and the viscous layer grids. This flow solver is an extension of a two-dimensional solver developed by Wang.[22] The so-called hanging nodes problem actually disappears because of the use of a cell-centered finite volume method supporting arbitrary grid cells. A Cartesian face with a hanging node is actually treated as four separate faces. The hanging nodes are, in fact, not visible to the flow solver. This simple treatment is not only accurate, but fully conservative as well.

The Reynolds-averaged Navier–Stokes equations can be written in the following integral form:

$$\int_V \frac{\partial Q}{\partial t}\,\mathrm{d}V + \int_S (F - F_v)\,\mathrm{d}S = 0 \tag{4}$$

The integration of Eq. (4) in an arbitrary control volume V_i gives

$$\frac{\mathrm{d}\bar{Q}_i}{\mathrm{d}t} V_i + \sum_f F_f S_f = \sum_f F_{v,f} S_f \tag{5}$$

where F_f and $F_{v,f}$ are the numerical inviscid and viscous flux vectors through face f and S_f is the face area. The overbar will be dropped from here on. Two major ingredients of the flow solver, data reconstruction and time integration, are briefly described in the following subsections. Roe's flux-difference splitting[28] has been used to compute the inviscid, whereas the viscous flux is computed using a simple and robust approach presented in Ref. 22 without a separate viscous reconstruction.

A. Reconstruction

In a cell-centered finite volume method, flow variables are known in a cell-average sense. No indication is given as to the distribution of the solution over the control volume. To evaluate the inviscid flux through a face, flow variables are required at both sides of the face. This task is fulfilled through data reconstruction. In this paper, a

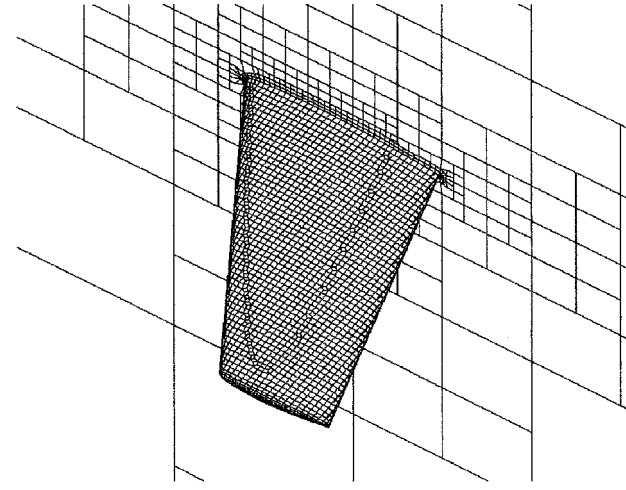

Fig. 5a Initial viscous Cartesian grid.

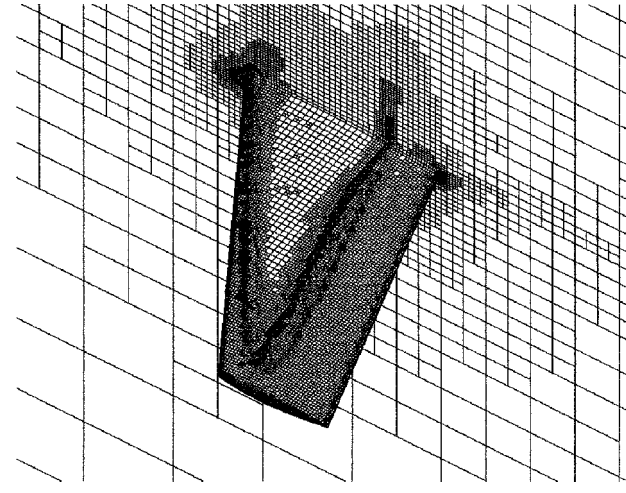

Fig. 5b Level-3 solution-adaptive Cartesian grids using Octree.

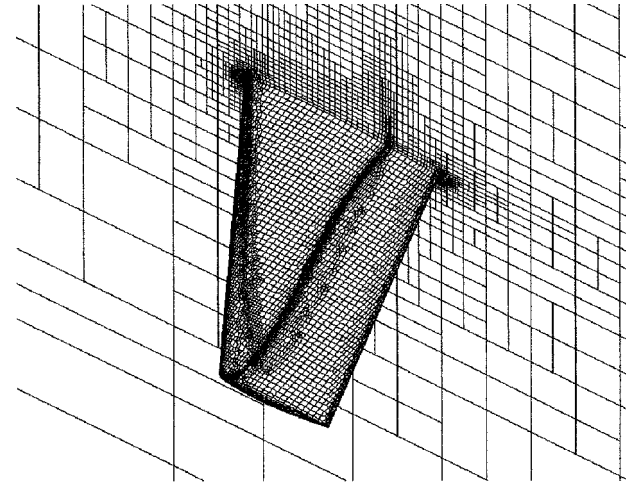

Fig. 5c Level-3 solution-adaptive Cartesian grids using 2^N tree.

least-squares reconstruction algorithm capable of preserving linear functions on arbitrary grids is employed. This linear reconstruction also makes the finite volume method second-order accurate in space. This reconstruction is briefly described next. The reconstruction problem reads as follows: Given cell-averaged primitive variables (denoted by q) for all of the cells of the computational grid, build a linear distribution for each cell, for example, c, using data at the cell itself and at its neighboring cells sharing a face with c. We use that the cell-averaged solutions can be taken to be the point solutions at the cell centroids without sacrificing the second-order accuracy. Therefore, we seek to reconstruct the gradient (q_x, q_y) for cell c, which produces the following linear distribution:

$$q(x, y) = q_c + q_x(x - x_c) + q_y(y - y_c) + q_z(z - z_c) \qquad (6)$$

where (x_c, y_c, z_c) are the cell-centroid coordinates. The following expressions can be easily derived from a least-squares approach:

$$\begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} = W \begin{bmatrix} \sum_n (q_n - q_c)(x_n - x_c) \\ \sum_n (q_n - q_c)(y_n - y_c) \\ \sum_n (q_n - q_c)(z_n - z_c) \end{bmatrix} \qquad (7)$$

where

$$W = \frac{1}{\Delta} \begin{bmatrix} I_{yy}I_{zz} - I_{yz}^2 & I_{xz}I_{yz} - I_{xy}I_{zz} & I_{xy}I_{yz} - I_{xz}I_{yy} \\ I_{xz}I_{yz} - I_{xy}I_{zz} & I_{xx}I_{zz} - I_{xz}^2 & I_{xy}I_{xz} - I_{xx}I_{yz} \\ I_{xy}I_{yz} - I_{xz}I_{yy} & I_{xy}I_{xz} - I_{xx}I_{yz} & I_{xx}I_{yy} - I_{xy}^2 \end{bmatrix}$$

$$\Delta = I_{xx}\left(I_{yy}I_{zz} - I_{yz}^2\right) + I_{xy}(2I_{xz}I_{yz} - I_{xy}I_{zz}) + I_{xz}^2 I_{yy}$$

$$I_{xx} = \sum_n (x_n - x_c)^2, \qquad I_{yy} = \sum_n (y_n - y_c)^2$$

$$I_{zz} = \sum_n (z_n - z_c)^2, \qquad I_{xy} = \sum_n (x_n - x_c)(y_n - y_c)$$

$$I_{yz} = \sum_n (z_n - z_c)(y_n - y_c), \qquad I_{xz} = \sum_n (x_n - x_c)(z_n - z_c)$$

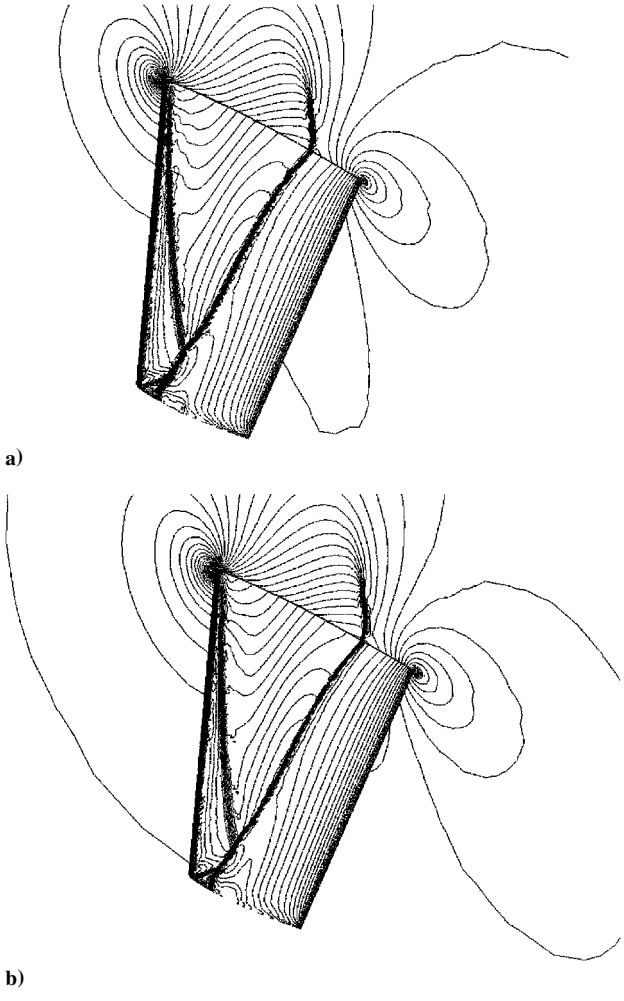

Fig. 6 Computed pressure contour on the level-3 solution-adaptive Cartesian grids using a) Octree and b) 2^N tree.

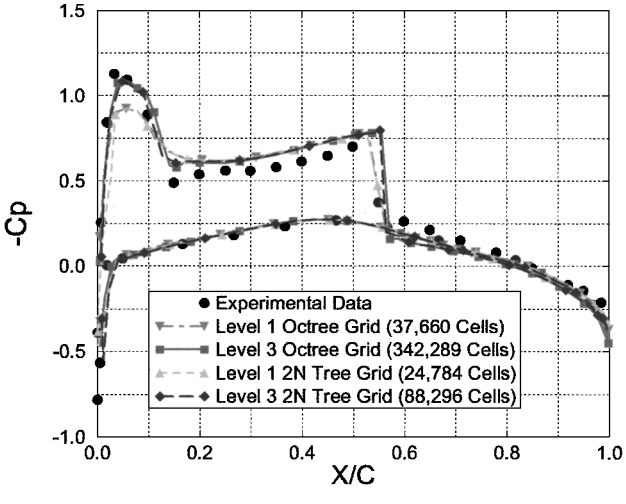

Fig. 7 Comparison between computed and experimental surface pressure coefficient at 44% semispan station.

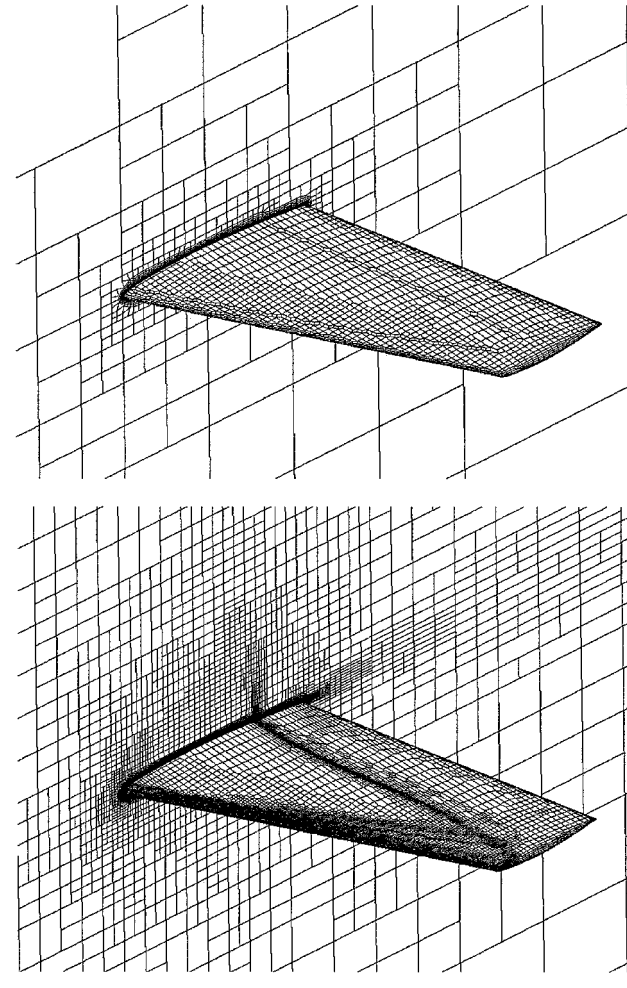

Fig. 8 Initial and level-3 adaptive grids for turbulent flow case.

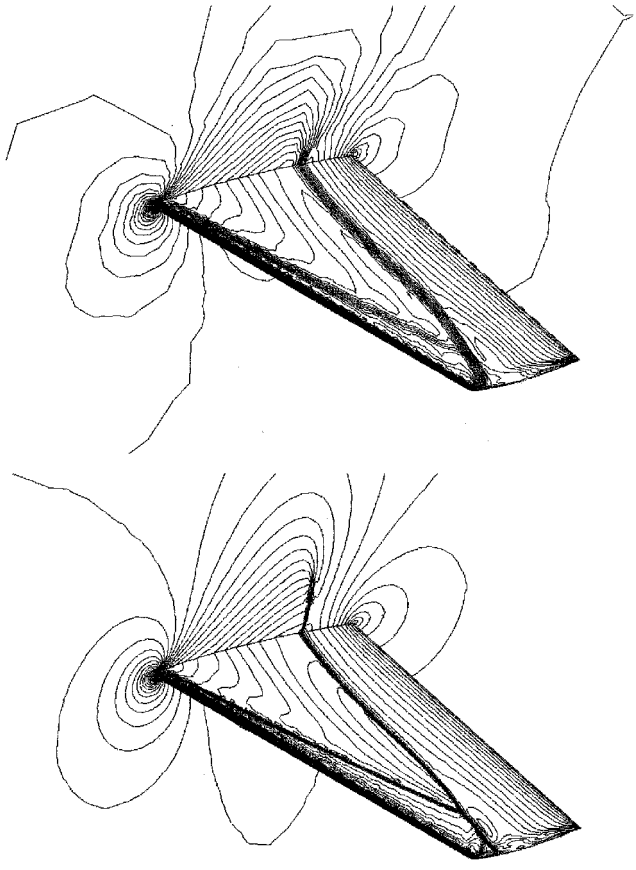

Fig. 9 Pressure contours on the initial and level-3 grids.

where n loops over the supporting neighboring cells. Note that matrix W is symmetric and dependent only on the computational grid. If one stores six elements of W for each cell, the reconstruction can be performed efficiently through a loop over all faces.

To handle steep gradients or discontinuities, a limiter due to Venkatakrishna[3] is used. This limiter is further modified by Wang[22] for adaptive grids.

B. Time Integration

Although explicit schemes are easy to implement, and are often useful for steady-state, inviscid flow problems, implicit schemes are found to be much more effective for viscous flow problems with highly clustered computational grids. An efficient block lower–upper symmetric gauss–seidel (LU-SGS) implicit scheme[29] has been developed for time integration on arbitrary grids. This block LU-SGS (BLU-SGS) scheme takes much less memory than a fully (linearized) implicit scheme, only having essentially the same or better convergence rate than a fully implicit scheme. The basic idea is presented here. Using backward Euler scheme for time-integration, we obtain

$$\frac{Q_i^{n+1} - Q_i^n}{\Delta t} V_i + \sum_f \left(F_f^{n+1} - F_{v,f}^{n+1}\right) S_f = 0 \tag{8}$$

Equation (8) can be further written in the delta form

$$\frac{\Delta Q_i}{\Delta t} V_i + \sum_f (\Delta F_f - \Delta F_{v,f}) S_f = -\sum_f \left(F_f^n - F_{v,f}^n\right) S_f \equiv \text{Res} \tag{9}$$

where $\Delta Q_i = Q_i^{n+1} - Q_i^n$, $\Delta F_f = F_f^{n+1} - F_f^n$, and $\Delta F_{f,v} = F_{f,v}^{n+1} - F_{f,v}^n$. In Eq. (9) the left-hand side fluxes can be replaced by their first-order counterparts to further simplify the implicit operator. In addition, the flux difference can be linearized in the following manner:

$$\Delta F_f(Q_i, Q_n) = \frac{\partial F_f}{\partial Q_i} \Delta Q_i + \frac{\partial F_f}{\partial Q_n} \Delta Q_n \tag{10}$$

where cell n and cell i share face f. The viscous flux difference can also be similarly linearized. Then Eq. (9) can be written as

$$\left[\frac{I}{\Delta t} V_i + \sum_f \left(\frac{\partial F_f}{\partial Q_i} - \frac{\partial F_{v,f}}{\partial Q_i}\right)\right] \Delta Q_i + \sum_f \left(\frac{\partial F_f}{\partial Q_n} - \frac{\partial F_{v,f}}{\partial Q_n}\right) \Delta Q_n = \text{Res} \tag{11}$$

Equation (11) is then solved with the following BLU-SGS scheme with multiple inner iterations. One can either fix the number of inner iterations or prescribe a convergence tolerance. Given the solutions $\Delta \boldsymbol{Q}^{(k-1)}$ at sweep level $k-1$, we compute the solution at the kth sweep using the following algorithm for the forward sweep:

$$D \Delta Q_i^* + \sum_{f, n<i} \left(\frac{\partial F_f}{\partial Q_n} - \frac{\partial F_{v,f}}{\partial Q_n}\right) \Delta Q_n^* + \sum_{f, n>i} \left(\frac{\partial F_f}{\partial Q_n} - \frac{\partial F_{v,f}}{\partial Q_n}\right) \Delta Q_n^{(k-1)} = \text{Res} \tag{12}$$

where

$$D = \frac{I}{\Delta t} V_i + \sum_f \left(\frac{\partial F_f}{\partial Q_i} - \frac{\partial F_{v,f}}{\partial Q_i}\right)$$

For the backward sweep we use

$$D \Delta Q_i^{(k)} + \sum_{f, n>i} \left(\frac{\partial F_f}{\partial Q_n} - \frac{\partial F_{v,f}}{\partial Q_n}\right) \Delta Q_n^{(k)} + \sum_{f, n<i} \left(\frac{\partial F_f}{\partial Q_n} - \frac{\partial F_{v,f}}{\partial Q_n}\right) \Delta Q_n^* = \text{Res} \tag{13}$$

Normally, only three inner iterations are sufficient. More details on the BLU-SGS scheme can be found in Ref. 29.

To simulate flow turbulence, the classical two-equation k–ε turbulence model with wall function was used.[30] Numerical tests with the wall functions indicate that an average y^+ value of about 30 usually gives reasonable results.

IV. Solution-Based Grid Adaptation

Solution-based grid adaptations have the potential of achieving the highest accuracy with minimum computer resources. Furthermore, to achieve automation in flow simulation, solution-based grid adaptation is essential. An ideal adaptation criterion would indicate regions that are causing the large errors, as well as regions that have too fine grids. A variety of adaptation criteria have been studied,[31–35] and it was shown that one can obtain a misleading converged solution if one is not careful.[34,35] Flow problems are usually of convection and diffusion type. Numerical errors produced in one region may significantly degrade the solution accuracy in other regions. In other words, one may not improve the solution much by refining the regions that have the largest absolute and relative errors, as demonstrated in a recent paper by Gu and Shih.[36] The pursuit of an optimum grid adaptation criterion is still a very intensive research area in CFD. In practice, it seems derivative-based adaptation criteria usually give acceptable results. In this paper, two directional adaptation criteria are developed and implemented. These directional criteria take full advantage of the anisotropic grid adaptation capability offered by the 2^N tree. The three coordinate directions of each Cartesian cell are examined independently for possible grid adaptations. Because the viscous layer grid is generated by projecting the Cartesian front to the geometry, it cannot be independently adapted. However, the number of viscous layers, and the grid clustering factor (or the minimum cell size in the wall normal direction), can be adapted based on local flow properties, such as the local cell Reynolds number or the y^+ value. Both first and second derivative-based grid adaptation criteria have been developed. The following

cellwise parameters are used as the adaptation indicators in the x direction

$$\tau_{ix} = \left|\frac{\partial q}{\partial x}\right|_i \Delta x_i^{1+u} \tag{14}$$

$$\pi_{ix} = \left|\frac{\partial^2 q}{\partial x^2}\right|_i \Delta x_i^{2+v} \tag{15}$$

where q can be any flow variable (pressure, total velocity, or density) and u and v are positive constants. The employment of positive exponents u and v have the effects of pushing the grid toward more uniformity, and guarding against overadaptation near discontinuities, which was identified to be the main cause of converging to the wrong solution.[34] In the tests we performed, we found that values of $u = 0.5$ and $v = 1$ gave reasonable results. The adaptation indicators in other directions can be computed similarly. Next, the standard deviation of the parameter is computed as

$$\tau = \left(\frac{\sum_{i=1}^{N}\tau_{ix}^2 + \sum_{i=1}^{N}\tau_{iy}^2 + \sum_{i=1}^{N}\tau_{iz}^2}{3N}\right)^{\frac{1}{2}} \tag{16}$$

where N is the total number of Cartesian cells. Finally, the following conditions are used for grid adaptation:

1) If $\tau_{ix} > \alpha^*\tau$, cell i is to be refined in the x direction.

2) If $\tau_{ix} < \beta^*\tau$, cell i is to be coarsened in the x direction.

where α is a control parameter determining the total number of cells to be refined, whereas β controls the number of cells to be coarsened. In this study, α is chosen to be 1, and β is set to be 0.1. The adaptation criteria are similar in the y and z directions.

In the steady-state test cases to be presented later, we have always started from very coarse initial grids that resolve the geometry. We have found that it is not necessary to perform grid coarsening. In an unsteady flow simulation with moving features, it is probably essential to perform both grid refinement and coarsening.

In turbulent flow simulations, the optimum average y^+ value is about 30 for the k–ε turbulence model with wall functions. Therefore, the grid clustering factor in the hyperbolic tangent grid distribution function is adjusted in every grid adaptation step so that the first cells from the wall have an average y^+ value of about 30.

V. Test Cases

A. Octree vs 2^N Tree for Transonic Flow over ONERA M6 Wing

This first test case is transonic flow over an ONERA M6 wing configuration.[37] The M6 wing has a leading-edge sweep angle of 30 deg, an aspect ratio of 3.8, and a taper ratio of 0.562. The airfoil section of the wing is the ONERA D airfoil, which is a 10% maximum thickness-to-chord ratio conventional section. The flow was first computed at a Mach number of 0.84, an angle of attack of 3.06 deg, and with an inviscid flow assumption. This case was selected to demonstrate the superiority of the 2^N tree over Octree in efficiently capturing flow features. The starting coarse computational grid was generated using the Octree data structure, and is shown in Fig. 5a. The mesh consists of 14,141 cells and 47,904 faces and two layers of projected layer grids. Three levels of solution-based grid adaptations were then performed. The adaptation criteria are pressure and Mach number gradients. With the Octree data structure, if a cell needs to be refined in any of the coordinate directions, the cell is refined in all directions. The level-3 Octree and 2^N tree Cartesian grids are shown in Fig. 5b and 5c. The Octree Cartesian grid is composed of 342,289 cells and 1,105,732 faces, whereas the 2^N tree Cartesian grid has only 88,296 cells and 300,573 faces. However, the solutions on the adapted grids are essentially identical, as shown

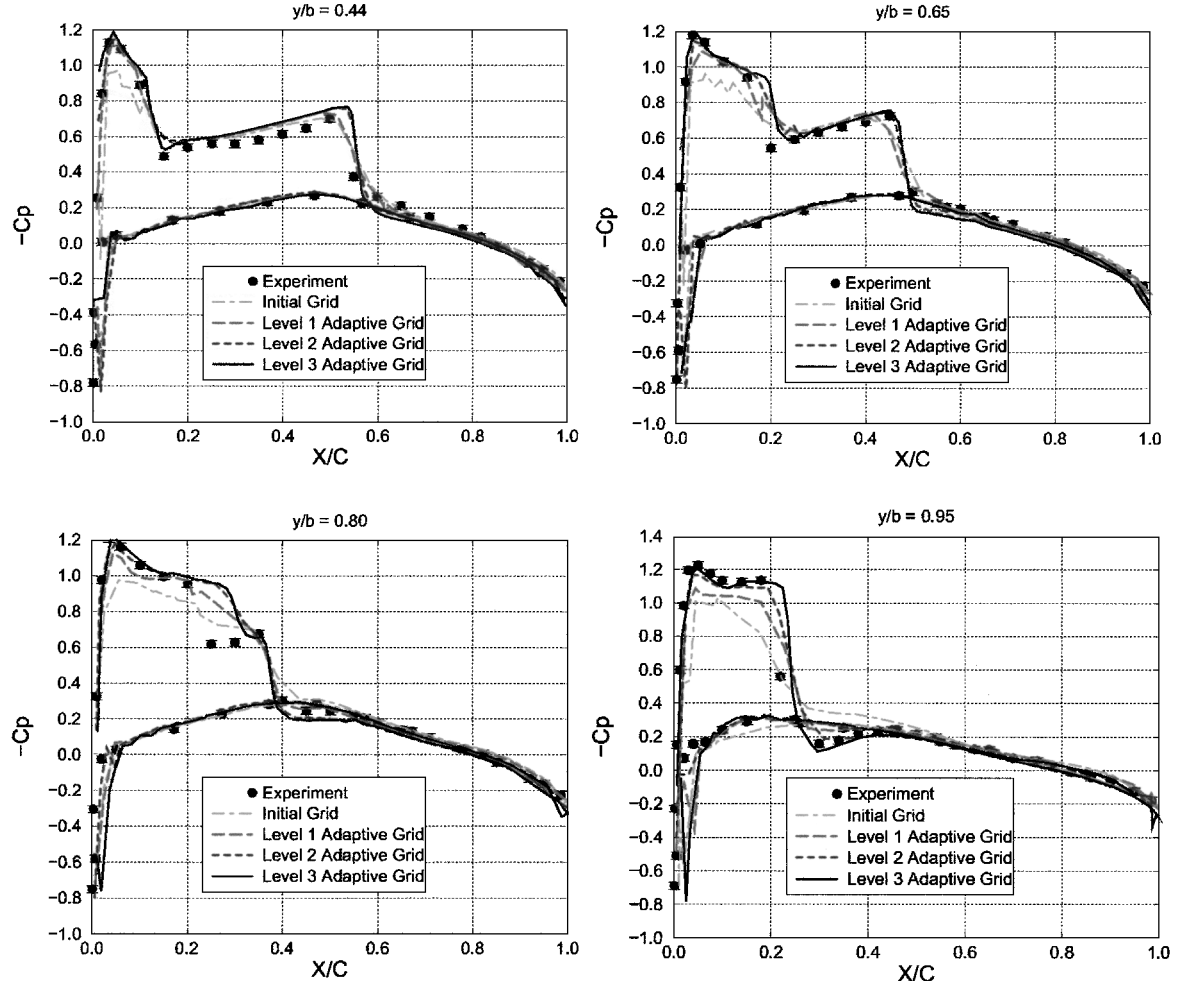

Fig. 10 Comparison of computed C_p profiles with experimental data.

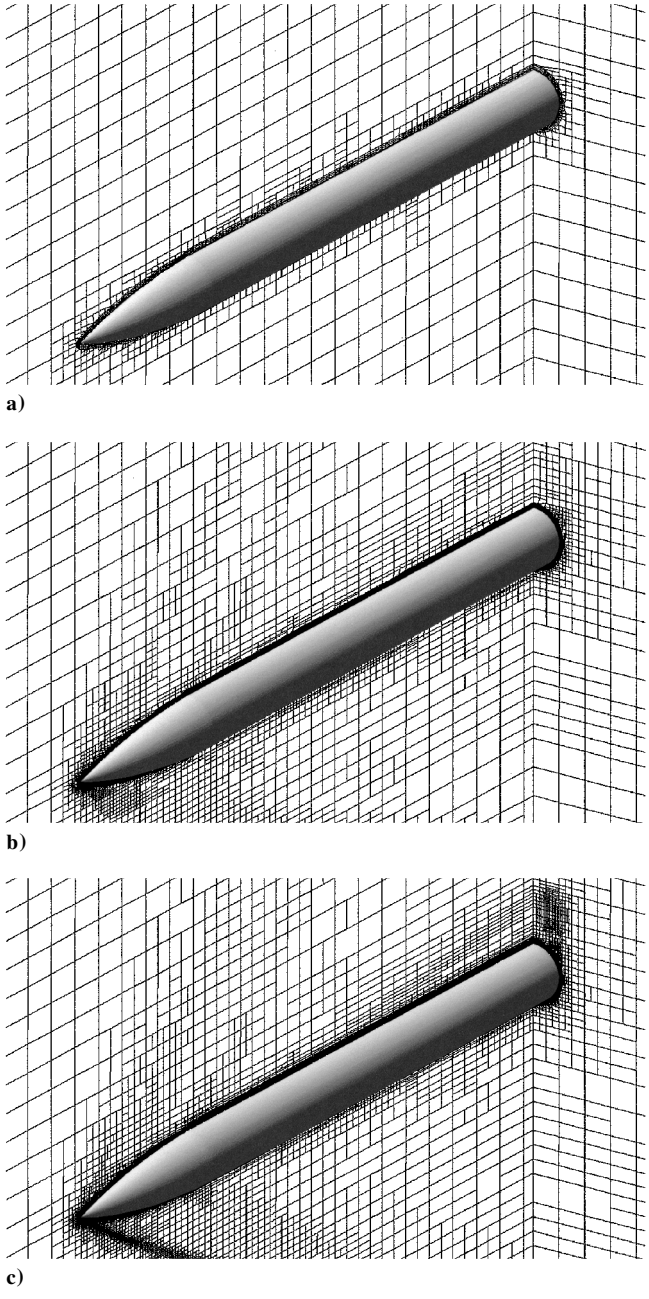

Fig. 11 Level a) 0, b) 2, and c) 4 solution adaptive grids for the missile configuration.

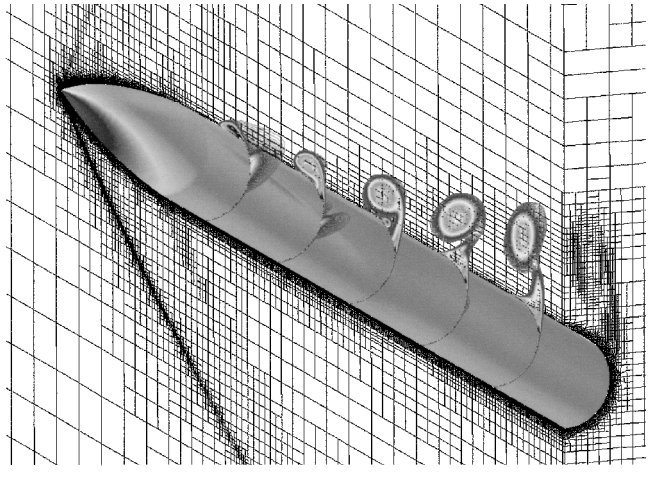

Fig. 12 Computed total pressure contours and surface pressure distribution at Mach = 1.8, α = 14 deg.

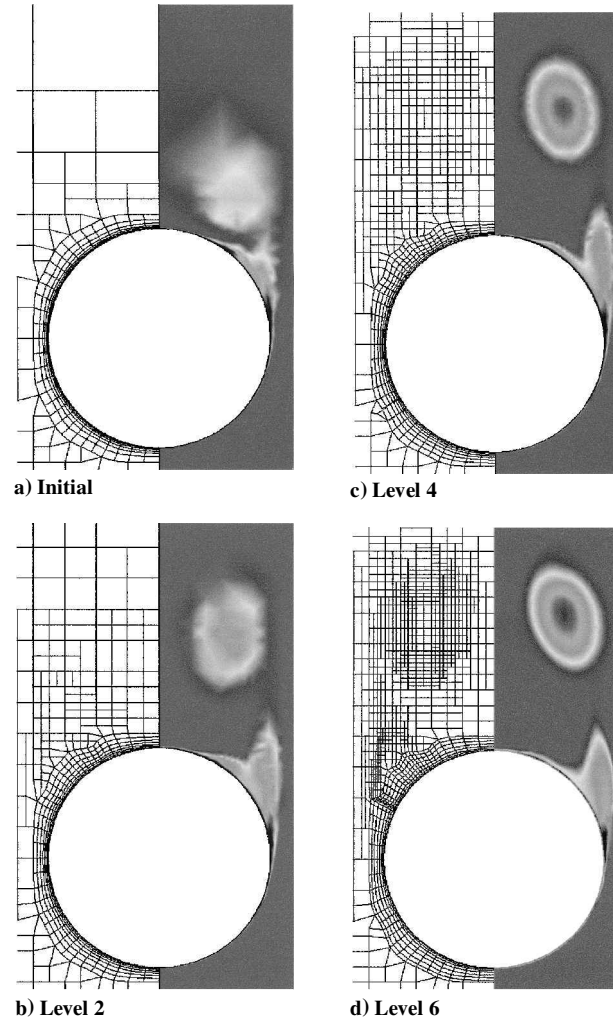

Fig. 13 Computational grids and total pressure distributions at *X/D* = 12.8 at different grid adaptation levels.

in Figs. 6 and 7. Figure 6 compares the pressure contours on the level-3 grids using Octree and the 2^N tree, and Fig. 7 presents the pressure coefficients on the wing surface at 44% semispan station. Note that the C_p profiles on the adapted grids of the same level using Octree and the 2^N tree are essentially the same. With the 2^N tree, a 75% saving in CPU was achieved over Octree.

This case was also computed assuming fully turbulent flow. The Reynolds number is 1.814×10^7. During the grid adaptation process, the average target y^+ value was kept around 30. The initial and final computational grids for the turbulent flow case are shown in Fig. 8. The initial grid has 40,480 cells, and the final level-3 grid has 413,551 cells. The pressure contours on the initial grid are compared with the contours on the final grid in Fig. 9. Note the dramatic difference in the resolution of the overall flow features. The computed surface C_p profiles are compared with experimental data at four spanwise sections in Fig. 10. It is observed that better agreement was obtained with grid adaptations. Note that a trend toward a grid-independent solution with grid adaptations is obvious in Fig. 10.

B. High-Angle-of-Attack Missile Aerodynamics

The second test case is performed for an ogive/cylinder configuration,[38] for which extensive experimental data are available for comparison. The model configuration of interest is a 3-caliber ogive with a 10-caliber cylindrical afterbody. The geometry of the configuration and the initial viscous Cartesian grid are displayed in Fig. 11a. The flow was computed at the following condition: Mach = 1.8, $\alpha = 14$ deg, $Re = 6.56 \times 10^6$/m. The diameter of the cylinder is 3.7 in. The flowfield is characterized by large separation regions and is a considerable challenge for turbulence models.

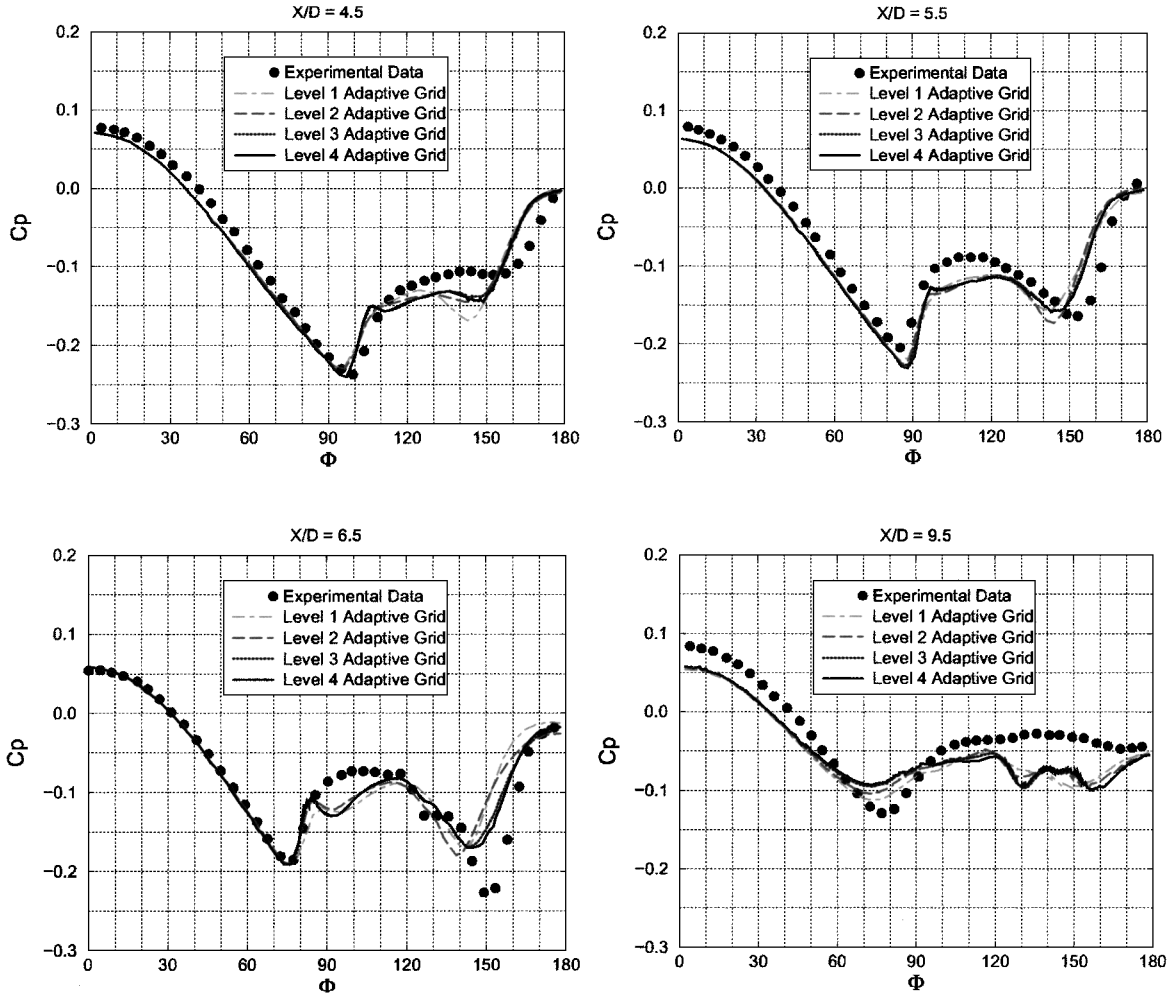

Fig. 14 Comparison between computed and experimental pressure coefficient profiles at several cross section stations.

Several levels of grid adaptations were performed after the solutions were converged on the coarse grids. Again the target average y^+ value is 30, which was achieved on the level-2 grid. The adaptation criterion used is total pressure gradient, which is designed to capture the complex vortex pattern of the flow. The level-2 and level-4 grids are displayed in Figs. 11b and 11c. The level-4 grid has 473,153 cells and 1,516,791 faces. Note that the development of the complex vortex pattern is evident on the level-4 grid. It is seen that the grid was also refined near the shock wave. A picture of the flowfield is shown in Fig. 12, which displays total pressure contours and the computational grid. The surface is colored with pressure distributions. Figure 13 shows the computational grid and total pressure distributions at section $X/D = 12.8$ for different grid adaptation levels. Note that the difference between solutions on the level-4 and level-6 grids are quite small. It is also quite obvious that anisotropic grid adaptations are used very extensively in the flowfield. Computed pressure coefficient profiles are compared with experimental data at several cross section stations in Fig. 14. Note that, at least visibly, the turbulent solutions are approaching grid independence because the difference in C_p computed with the level-3 and level-4 grids is very small. This is the first evidence that grid-independent turbulent solutions are possible with anisotropic grid adaptations. Generally speaking, the agreement between the numerical simulation and the experimental data is fairly good.

VI. Conclusions

A 2^N tree adaptive viscous Cartesian grid method has been developed and tested for both inviscid and viscous turbulent flows. It is confirmed that the 2^N tree is much more efficient in capturing flow features than the Octree data structure. In the inviscid flow case with the M6 wing geometry, a 75% saving in total number of cells can be achieved. With anisotropic grid adaptations for turbulent flow simulation, drastic improvements in solution accuracy are demonstrated for both the M6 wing and the high-angle-of-attack missile cases.

The adaptation criteria are shown to be very effective in capturing shock waves, wakes, and complex vortex structures. The use of anisotropic grid adaptations allows these features to be captured very efficiently.

Acknowledgments

The research was supported by a Navy Small Business Innovation Research project from the Naval Air Warfare Center (NAWC)/Aircraft Division (AD). We thank D. Grove of NAWC/AD for serving as the Technical Monitor. We also thank A. K. Singhal of CFD Research Corp. (CFDRC) for granting the first author a free CFDRC software license.

References

[1] Barth, T. J., and Frederickson, P. O., "Higher-Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction," AIAA Paper 90-0013, Jan. 1990.

[2] Batina, J. T., "Unsteady Euler Algorithm with Unstructured Dynamic Mesh for Complex Aircraft Aerodynamic Analysis," *AIAA Journal*, Vol. 29, No. 3, 1991, pp. 327–333.

[3] Venkatakrishnan, V., "Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters," *Journal of Computational Physics*, Vol. 118, 1995, pp. 120–130.

[4] Anderson, W. K., and Bonhaus, D. L., "An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids," *Computers and Fluids*, Vol. 23, 1994, pp. 1–21.

[5] Connell, S. D., and Holmes, D. G., "Three-Dimensional Unstructured Adaptive Multigrid Scheme for the Euler Equations," AIAA Paper 93-3339, July 1993.

[6]Dannenhoffer, J. F., III, and Baron, J. R., "Adaptation Procedures for Steady State Solution of Hyperbolic Equations," AIAA Paper 84-0005, 1984.

[7]Schneiders, R., "Automatic Generation of Hexahedral Finite Element Meshes," *Proceedings of 4th International Meshing Roundtable*, Sandia National Labs., Albuquerque, NM, 1995, pp. 103–114.

[8]Nakahashi, K., "Adaptive-Prismatic-Grid Method for External Viscous Flow Computations," AIAA Paper 93-3314, July 1993.

[9]Kallinderis, Y., Khawaja, A., and McMorris, H., "Hybrid Prismatic/Tetrahedral Grid Generation for Viscous Flows Around Complex Geometries" *AIAA Journal*, Vol. 34, No. 2, 1996, pp. 291–298.

[10]Coirier, W. J., and Jorgenson, P. C. E., "Mixed Volume Grid Approach for the Euler and Navier–Stokes Equations," AIAA Paper 96-0762, Jan. 1996.

[11]Lohner, R., and Parikh, P., "Generation of Three-Dimensional Unstructured Grids by Advancing Front Method," *International Journal of Numerical Method in Fluids*, Vol. 8, 1988, pp. 1135–1144.

[12]Baker, T. J., "Three-Dimensional Mesh Generation by Triangulation of Arbitrary Point Sets," AIAA Paper 87-1124, Jan. 1987.

[13]Luo, H., Sharov, D., Baum, J. D., and Lohner, R., "On the Computation of Compressible Turbulent Flows on Unstructured Grids," AIAA Paper 2000-0927, Jan. 2000.

[14]Berger, M. J., and LeVeque, R. J., "Adaptive Cartesian Mesh Algorithm for the Euler Equations in Arbitrary Geometries," AIAA Paper 89-1930, June 1989.

[15]DeZeeuw, D., and Powell, K., "An Adaptively Refined Cartesian Mesh Solver for the Euler Equations," AIAA Paper 91-1542, July 1991.

[16]Coirier, W. J., and Powell, K. G., "Solution-Adaptive Cartesian Cell Approach for Viscous and Inviscid Flows," *AIAA Journal*, Vol. 34, No. 5, 1996, pp. 938–945.

[17]Bayyuk, A. A., Powell, K. G., and van Leer, B., "A Simulation Technique for 2D Unsteady Inviscid Flows Around Arbitrarily Moving and Deforming Bodies of Arbitrary Geometry," AIAA Paper 93-3391, July 1993.

[18]Melton, J. E., Enomoto, F. Y., and Berger, M. J., "3D Automatic Cartesian Grid Generation for Euler Flows," AIAA Paper 93-3386, July 1993.

[19]Aftosmis, M. J., Berger, M. J., and Melton, J. E., "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry," AIAA Paper 97-0196, Jan. 1997.

[20]Karman, S. L., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complete Geometries," AIAA Paper 95-0343, Jan. 1995.

[21]Wang, Z. J., "Proof of Concept—An Automatic CFD Computing Environment with a Cartesian/Prism Grid Generator, Grid Adaptor and Flow Solver," *Proceedings of 4th International Meshing Roundtable*, Sandia National Labs., Albuquerque, NM, 1995, pp. 87–99.

[22]Wang, Z. J., "A Quadtree-Based Adaptive Cartesian/Quad Grid Flow Solver for Navier–Stokes Equations," *Computers and Fluids*, Vol. 27, No. 4, 1998, pp. 529–549.

[23]Smith, R. J., and Leschziner, M. A., "A Novel Approach to Engineering Computations for Complex Aerodynamic Flows," *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulations*, Mississippi State Univ., Mississippi State, MS, 1996.

[24]Tchon, K. F., Hirsch, C., and Schneiders, R., "Octree-Based Hexahedral Mesh Generation for Viscous Flow Simulations," AIAA Paper 97-1980, July 1997.

[25]Wang, Z. J., "An Automated Viscous Adaptive Cartesian Grid Generation Method for Complex Geometries," *Proceedings of the 6th International Conference on Numerical Grid Generation in Computational Field Simulations*, Mississippi State Univ., Mississippi State, MS, 1998, pp. 577–586.

[26]Wang, Z. J., Chen, R. F., Hariharan, N., and Przekwas, A. J., "A 2^N Tree Based Automated Viscous Cartesian Grid Methodology for Feature Capturing," *Proceedings of 14th AIAA Computational Fluid Dynamics Conference*, AIAA, Reston, VA, 1999.

[27]Bonet, J. A., and Peraire, "An Alternating Digital Tree (ADT) Algorithm for 3D Geometric Searching and Intersection Problems," *International Journal of Numerical Methods in Engineering*, Vol. 31, 1991, pp. 1–17.

[28]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, 1983, p. 357.

[29]Chen, R. F., and Wang, Z. J., "Fast, Block Lower–Upper Symmetric Gauss–Seidel Scheme for Arbitrary Grids," *AIAA Journal*, Vol. 38, No. 12, 2000, pp. 2238–2245.

[30]Launder, B. E., and Spalding, D. B., "The Numerical Computation of Turbulent Flows," *Computer Methods in Applied Mechanics and Engineering*, Vol. 3, 1974, p. 269.

[31]Aftosmis, M. J., "Upwind Method for Simulation of Viscous Flow on Adaptively Refined Meshes," *AIAA Journal*, Vol. 32, No. 2, 1994, pp. 268–277.

[32]Kallinderis, Y., "Adaptation Methods for a New Navier–Stokes Algorithm," *AIAA Journal*, Vol. 27, No. 1, 1989, pp. 37–43.

[33]Berger, M. J., and Aftosmis, M. J., "Aspects (and Aspect-Ratios) of Cartesian Mesh Methods," *Proceedings of 16th International Conference on Numerical Methods in Fluid Dynamics*, 1998, Springer-Verlag, Heidelberg, Germany, 1998.

[34]Warren, G., Anderson, W. K., Thomas, J., and Krist, S., "Grid Convergence for Adaptive Methods," *Proceedings of 10th AIAA Computational Fluid Dynamics Conference*, AIAA, Washington, DC, 1991.

[35]Venditti, D. A., and Darmofal, D. L., "Grid Adaptation for Functional Outputs of 2D Compressible Flow Simulations," AIAA Paper 2000-2244, 2000.

[36]Gu, X., and Shih, T. I.-P., "Differentiating Between Source and Location of Error for Solution Adaptive Mesh Refinement," *Proceedings of 15th AIAA Computational Fluid Dynamics Conference*, AIAA, Reston, VA, 2001.

[37]Schmitt, V., and Charpin, F., "Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers," *Experiment Data Base for Computer Program Assessment*, AR-138, AGARD, 1979.

[38]Sturek, W. B., Birch, T., Lauzon, M., Housh, C., Manter, J., Josyula, E., and Soni, B., "The Application of CFD to the Prediction of Missile Body Vortices," AIAA Paper 97-0637, Jan. 1997.

P. Givi
Associate Editor