Journal of Computational Physics **181,** 639–653 (2002)
doi:10.1006/jcph.2002.7147

An Approach to Local Refinement of Structured Grids

P. A. Durbin and G. Iaccarino

Department of Mechanical Engineering, Stanford University, Building 500, Stanford, California 94305-3030

Received July 6, 2001; revised May 22, 2002

Structured grids can be refined locally by adding line segments. These are notionally made into complete grid lines by filling the remainder of the line with blanks. A discretization scheme is adapted to this type of grid by adding interpolation points to complete differencing stencils. The Navier–Stokes equations are solved by the artificial compressibility approach. Examples show the efficacy of the method.

Key Words: computational fluid dynamics; structured grids; mesh refinement; artificial compressibility.

1. INTRODUCTION

Structured grid algorithms have advantages in many CFD applications. They can be more computationally efficient than unstructured methods; mesh connectivity is simpler; a higher degree of implicitness is facilitated for stiff problems; they can simplify resolution of thin shear layers.

One drawback to structured gridding is the difficulty of adapting the grid to the solution. Fluid mechanics is notorious for solution fields containing steep gradients and concentrated, vortical flow features. Solution adaptive grid refinement is a desirable capability.

The majority of research on solution adaptive gridding has been for unstructured solution methods [8]. Local adaptation involves inserting nodes within a preexisting mesh; two different refinement strategies are possible: conformal and nonconformal (also called h-type) refinement. In the former grid connectivity between the new nodes and the surroundings is rebuilt; even though this technique is well suited only to triangular (or tetrahedral) grids it has been used extensively in the literature [4]. The nonconformal refinement, on the other hand, can be applied to any type of mesh: New nodes are generated by subdividing preexisting cells. At the interface between refined and nonrefined cells hanging nodes are left and an interpolation/reconstruction of the solution is required to ensure the accuracy of the procedure [8].

0021-9991/02 $35.00

Our interest in adaptive, structured grids arose from work on RANS equation solvers. Vagaries of some turbulence models favor solution by implicit line-relaxation methods. For instance, wall boundary conditions can strongly influence interior nodes. Often turbulence is most intense in thin regions, which require a locally refined grid. Such considerations motivated the present proposal for solution adaptive refinement. The present paper proposes that a method analogous to h-refinement can be developed for structured grid algorithms.

Previous methods for solution-adaptive, structured gridding have involved globally regenerating the grid. Such techniques are referred to as grid movement (see review in Ref. [7]). As examples, the authors of [13] move boundary points to alter the distribution of nodes placed inside the domain by an algebraic grid generator; in [9] and [12] solution gradients are used to modify the control functions of parabolic and elliptic grid generators; in [6] an intermediate grid is modified in a parametric domain. In all cases the entire grid is reformed to capture local gradients.

Another approach is to refine subdomains of a multiblock, structured grid [1]. In this case discontinuous grids meet at block-to-block boundaries and appropriate coupling operators must ensure accuracy and conservation.

A large amount of published literature deals with Cartesian adaptive mesh refinement techniques (this is a special case of h-refinement). The mesh is a collection of regular elements that can be split into subelements; connectivity between elements is treated by tree structures, and an unstructured solver is used, even if the mesh appears as a locally structured grid. In [2] this is done within the context of an Euler solver. The type of grids generated by [2] can readily be adapted to structured solution algorithms by the present approach, although that is not its primary motivation.

The locally adaptive method proposed herein was motivated by the idea of *iblanking* [3] (the terminology *iblank* comes from a variable name used in computer codes). Originally, *iblanking* was a device to insert geometry into a structured grid by extending the grid inside the body, then blanking out the interior portion. The region inside the body is decoupled from the fluid via boundary conditions. In the original approach the interior region is solved, although the equations there are arbitrary—usually formulated as solving $u(\boldsymbol{x}) = constant$, or the like. It is straightforward to revise such algorithms so that the blanked region is skipped, requiring neither storage of variables nor solution of equations. For present purposes, the notionally blanked portion of the grid could be larger than the active portion. Our solution algorithm skips the blanked portion.

Conceptually, our method consists of adding grid lines where needed, and blanking out all but the portion of those lines that lies in the area where higher resolution is required. Since the blanked portion is skipped, the actual method adds line segments (refer to Fig. 8). The line segments lie on an underlying, notional, structured grid. The notional grid would be constructed from active and blanked nodes, as illustrated by Fig. 1. The black circles denote

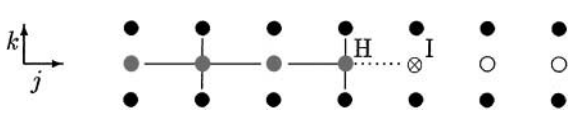


FIG. 1. Active (●, ●) and blanked (○) grid points. Points indicated by ● show the initial grid; those indicated by ● are added through refinement; ⊗ is an interpolated point. Complete (solid) and incomplete (dotted) finite-difference stencils are shown.

an initial, coarse grid. The gray circles indicate points added through local refinement. They amount to inserting an extra grid line in the k direction, a portion of which is active; the rest is blanked—as indicated by the open circles. A discrete solution scheme must be adapted for this class of grids.

2. FINITE DIFFERENCING

For the purpose of initial development, we consider the incompressible Navier–Stokes equations. Discretization is by finite differences. Finite volumes could be used, but a finite-difference formulation makes the method quite apparent. The node labeled **H** in Fig. 1 (usually called a "hanging node") is connected to three active nodes. To complete the finite-difference stencil a solution value is interpolated to the point labeled **I**. Note that **I** is *only* used to complete the stencil at **H**; otherwise it is treated as blanked. The interpolation stencil determines the effective finite-difference scheme and its local accuracy.

For instance, if the value at **I** is linearly interpolated between its vertical neighbors, then the six-point stencil in Fig. 2 is implied by the five-point stencil in Fig. 1. A centrally differenced j derivative is second-order accurate for a symmetric stencil; $\delta_j u = (1/2)(u_{j+1} - u_{j-1})$. However, with an interpolated value as in Fig. 2,

$$\delta_j u_{j,k} = \frac{1}{2}\left(\frac{1}{2}(u_{j+1,k+1} + u_{j+1,k-1}) - u_{j-1,k}\right), \tag{1}$$

the accuracy becomes first order because of the asymmetry, irrespective of the fact that the value $u_{j+1,k} = (1/2)(u_{j+1,k+1} + u_{j+1,k-1})$ is a second-order interpolation in the k direction.

In a 2-D grid of N^2 nodes, the number of hanging nodes will be $O(N)$. Hence the global error of a second-order method on complete stencils would not be reduced by first-order accuracy adjacent to interpolation points. However, the local dissipative error due to first-order convection might be undesirable. Local accuracy can be increased by modifying the interpolation stencil. The interpolation formula

$$u_{j+1,k} = \frac{1}{2}(u_{j+1,k+1} + u_{j+1,k-1}) - \frac{1}{2}(u_{j,k+1} + u_{j,k-1} - 2u_{j,k}) \tag{2}$$

substituted in place of $(1/2)(u_{j+1,k+1} + u_{j+1,k-1})$ in (1) provides second-order accuracy.

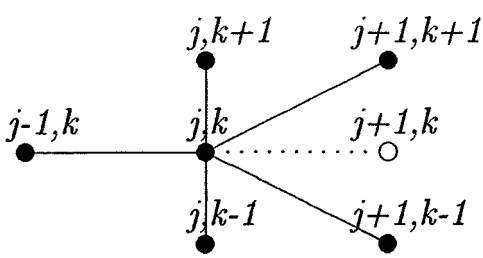


FIG. 2. Effective finite-difference stencil.

Generally, a value for $u_{j+1,k}$ could be obtained by a local reconstruction of the solution. For instance Eq. (2) can be derived from a second-order polynomial reconstruction. Initially $u_{j+1,k}$ is treated as given, and then its value is obtained by eliminating the first-order error from the finite-difference formula. In short, in the region shown by Fig. 2 let

$$u = a + bx + cx^2 + dy + ey^2 + fxy.$$

Then the requirement that this agree with $u_{j,k}$ at the seven points shown in the figure leads to the formula (2).

For a given interpolation stencil, constraints are placed on the allowable grid refinements. Let a coarse grid be set. An initial flow solution is obtained on that grid. That solution is then used to define refinement line segments: For instance, adaptation might be based on distributing the total variation of a solution variable more evently over the grid [7]. Interpolation points are then set to complete the finite-difference stencil at hanging nodes. In the case of Fig. 2, values at $(j + 1, k)$ are interpolated; to set them, solution variables must be available at $(j + 1, k + 1)$ and $(j + 1, k - 1)$.

A definition of "hanging node" is needed. Let Δ_c be the coarsest grid spacing in computational space. Then hanging nodes are those for which at least one neighbor is spaced farther than Δ_c. If the grid is refined by bisection, then after p levels of refinement the finest grid spacing Δ_f has become $\Delta_c/2^p$—with the caveat that much of the finer grid points are *iblanked*, or notional. Identification of the interpolation points is a simple check of whether $|\Delta j|$ or $|\Delta k|$ are greater than Δ_c; if so, both nodes in that direction are identified as hanging nodes, and interpolation points are added to complete their difference stencils.

A few of the fine grid points may lead to unsuitable interpolation stencils; this depends on how the interpolation is performed. With the present method for the linear interpolation (1) to be applied, the interpolated points must lie between two active nodes in at least one direction. As a corollary, no cell face may contain more than one hanging node. It suffices to impose a broad requirement that *interpolation points be connected no more than pairwise.*

This rule can be imposed by revising the mesh after refinement. First hanging nodes are identified: Pairs of nodes for which $|\Delta j| > \Delta_c$ or $|\Delta k| > \Delta_c$ are found. These are locations where interpolation would be required. Then the mesh is repaired: Cycling through the identified interpolation points, those connected to more than one other interpolation point are selectively deleted, and new connections are made. Reconnection is quite a simple matter on the structured grid.

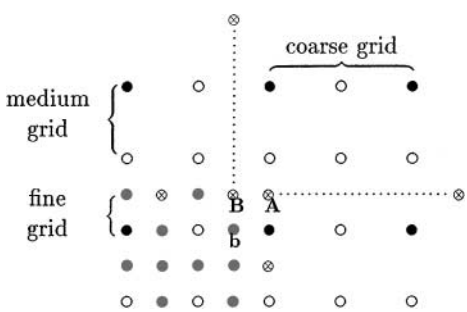


FIG. 3. Upper right corner of a refined grid, showing interpolation points. ●, coarse grid; ○, medium grid; ●, finest grid. ⊗ are interpolated points. The dotted lines show connections to distant interpolation points.

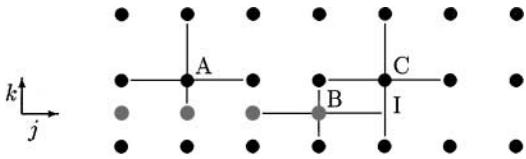


FIG. 4. Five-point stencils for the active points **A**, **B**, **C** (●, coarse grid; ●, fine grid); note that the interpolation point **I** is used for the stencil at **B** but not at **C**.

Figure 3 illustrates the process. This figure shows the upper right portion of the fine region of a twice-refined grid. (The grid is meant to continue beyond the borders of the figure.) An unacceptable initial distribution of interpolation points, ⊗, is left by the two levels of refinement, ● and ○. Points labeled **A** and **B** are each connected to two ⊗'s; this violates the rule stated previously. Note that connections to distant points must be accounted for, even if they lie on the boundary of the computational domain. While it might seem that **A** is allright, because it lies between two active points, that is immaterial, since point **A** does not enter the difference stencil of any active point and so would not be used; such considerations lie behind the broadly stated rule here.

If point **B** is deleted, then **A** becomes acceptable and point **b** becomes identified as an interpolation point. Points **A** and **b** then lie between two active nodes. Note, however, that at **b** formula (2) contains an interpolated value. That affects the second-order accuracy; however, such awkward points arise only in isolated locations—here at a corner in the fine grid. When they arise they are accepted.

Once the interpolation points are properly labeled, discretization stencils are constructed for the active points. Figure 4 represents active points on a locally refined grid. The stencil at point **A** uses active points only, with a nonuniform spacing in the k direction, whereas point **B** uses the interpolated value **I**. Note that the k-line, where **I** and **B** are located, is skipped by the stencil of point **C**; hence the stencil at that point is regularly spaced on the coarse grid.

A complete example is shown in Fig. 5. This is a grid for flow over a backstep. The coarse grid was refined uniformly in two blocks. The upper right corners of each block contained unacceptable interpolation points. By counting connections, these points were located and fixed. The encircled regions contain dislocations where deletions were needed to form acceptable interpolation stencils.

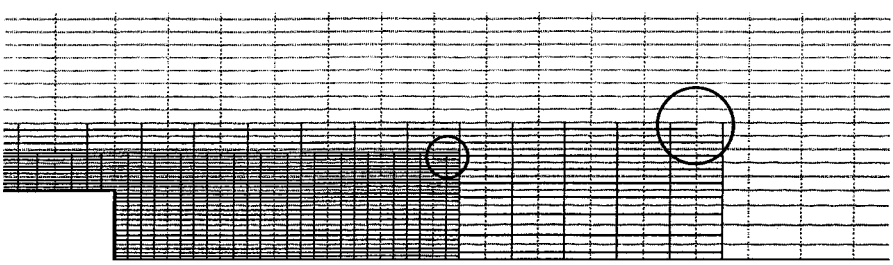


FIG. 5. Block refinement for flow over a backstep. Two levels of refinement are shown. The y axis has been magnified by a factor of 2. Faint lines are the coarse grid. Circles indicate where the interpolation points were revised.

3. NAVIER–STOKES SOLUTION

The steady, incompressible Navier–Stokes equations were solved by the artificial compressibility method, following the approach in [10]. The primary modification was to reformulate the algorithm in finite differences; for present purposes it was also coded for Cartesian grids. The governing equations are

$$\begin{aligned} \partial_\tau p + \beta \partial_j u_j &= 0, \\ \partial_\tau u_i + \partial_j (u_j u_i + p\delta_{ij}) &= \nu \nabla^2 u_i, \end{aligned} \tag{3}$$

where τ is an artificial time step that is used for relaxation. The compressibility parameter β drops out in steady state; it couples the velocity to the pressure during iterations. β also contributes to the numerical dissipation. Its value should be proportional to the square of a reference velocity. In the present computations, the reference velocity was unity and the value of β was 2.

As in [10], Eqs. (3) are differenced; they are then solved by iterating on the linearized finite-difference equations. In Δ form the convective terms become

$$[\Delta \boldsymbol{q} + \delta\tau \delta_x \Delta \boldsymbol{A} + \Delta\tau \delta_y \Delta \boldsymbol{B}] = -\delta\tau \boldsymbol{R}, \tag{4}$$

where $\boldsymbol{q} = (p, u, v)$ in two dimensions. Here $\Delta \boldsymbol{A} = \boldsymbol{\partial_q A} \cdot \Delta \boldsymbol{q}$ and $\Delta \boldsymbol{B} = \boldsymbol{\partial_q B} \cdot \Delta \boldsymbol{q}$, in which $\boldsymbol{\partial_q A}$ and $\boldsymbol{\partial_q B}$ are Jacobi matrices of the x- and y-direction fluxes, which will be described in the following. The residual, $\boldsymbol{R}$, is the discretized steady equation. $\Delta \boldsymbol{q}$ is the increment of the solution variable in the pseudo-time step $\delta\tau$ and δ_x is the finite-difference operator. Convergence requires $\|\Delta \boldsymbol{q}\| \to 0$ as $\tau \to \infty$.

The discrete derivatives δ_x and δ_y are defined by first-order upwinding on the left-hand side of (4); third-order upwind biasing is applied on the right. The viscous diffusion term uses second-order central differencing. The only difference with [10] is that Eqs. (4) are discretized as they stand, without being multiplied by the cell volume. This means that the cell face areas do not enter the Jacobian, which is a transparent way to avoid the discontinuous cell area change at hanging nodes (Fig. 1). For the present, incompressible flow application, this differencing scheme is as accurate as flux-based discretization.

From the ∂_x derivative on the left side of (3), $\boldsymbol{A}$ and its Jacobian,

$$\boldsymbol{A} = \begin{pmatrix} \beta u \\ uu + p \\ uv \end{pmatrix}; \quad \frac{\partial \boldsymbol{A}}{\partial \boldsymbol{q}} = \begin{pmatrix} 0 & \beta & 0 \\ 1 & 2u & 0 \\ 0 & v & u \end{pmatrix}, \tag{5}$$

are easily obtained; similar expressions for $\boldsymbol{B}$ and $\boldsymbol{\partial_q B}$ can be written. Following a well-known procedure [11], the Jacobian can be split into parts with positive and negative eigenvalues,

$$\begin{aligned} \boldsymbol{\partial_q A}^+ &= \frac{1}{2c} \begin{pmatrix} \beta & \beta(u+c) & 0 \\ (u+c) & (u+c)^2 & 0 \\ v(1-|u|/c) & v(c+2u-|u|u/c) & c(u+|u|) \end{pmatrix}, \\ \boldsymbol{\partial_q A}^- &= \boldsymbol{\partial_q A} - \boldsymbol{\partial_q A}^+, \end{aligned} \tag{6}$$

where $c^2 = u^2 + \beta$.

The split into positive and negative matrices provides numerical dissipation. There is a degree of freedom in how this is done; we have invoked the following. First note that a derivative biased toward $j-1$ can be written

$$\begin{aligned}\delta_x \boldsymbol{A} &= \frac{(\boldsymbol{A}_j - \boldsymbol{A}_{j-1})}{\Delta x} \\ &= \frac{(\boldsymbol{A}_{j+1} - \boldsymbol{A}_{j-1})}{2\Delta x} + \left[\frac{(\boldsymbol{A}_j - \boldsymbol{A}_{j-1}) - (\boldsymbol{A}_{j+1} - \boldsymbol{A}_j)}{2\Delta x}\right] \end{aligned} \tag{7a}$$

and a derivative biased toward $j+1$ is

$$\begin{aligned}\delta_x \boldsymbol{A} &= \frac{(\boldsymbol{A}_{j+1} - \boldsymbol{A}_j)}{\Delta x} \\ &= \frac{(\boldsymbol{A}_{j+1} - \boldsymbol{A}_{j-1})}{2\Delta x} - \left[\frac{(\boldsymbol{A}_j - \boldsymbol{A}_{j-1}) - (\boldsymbol{A}_{j+1} - \boldsymbol{A}_j)}{2\Delta x}\right]. \end{aligned} \tag{7b}$$

Expression (7b) is linearized using the positive part of $\boldsymbol{A}$ for the square-bracketed term:

$$2\Delta x \delta_x \boldsymbol{A} = \partial_{\boldsymbol{q}} \boldsymbol{A}_j (\boldsymbol{q}_{j+1} - \boldsymbol{q}_{j-1}) + \partial_{\boldsymbol{q}} \boldsymbol{A}^+_{j-1/2} (\boldsymbol{q}_j - \boldsymbol{q}_{j-1}) - \partial_{\boldsymbol{q}} \boldsymbol{A}^+_{j+1/2} (\boldsymbol{q}_{j+1} - \boldsymbol{q}_j). \tag{8}$$

Correspondingly, expression (7b) is linearized using the negative part of $\partial_{\boldsymbol{q}} \boldsymbol{A}$. Then, on the left side of Eq. (4), $\delta_x \Delta \boldsymbol{A}$ is differenced by adding the positive and negative splits to the central term, as

$$\begin{aligned} 2\Delta x \delta_x \Delta \boldsymbol{A} = {} & \partial_{\boldsymbol{q}} \boldsymbol{A}_j (\Delta \boldsymbol{q}_{j+1} - \Delta \boldsymbol{q}_{j-1}) + \partial_{\boldsymbol{q}} \boldsymbol{A}^+_{j-1/2} (\Delta \boldsymbol{q}_j - \Delta \boldsymbol{q}_{j-1}) \\ & - \partial_{\boldsymbol{q}} \boldsymbol{A}^+_{j+1/2} (\Delta \boldsymbol{q}_{j+1} - \Delta \boldsymbol{q}_j) - \partial_{\boldsymbol{q}} \boldsymbol{A}^-_{j-1/2} (\Delta \boldsymbol{q}_j - \Delta \boldsymbol{q}_{j-1}) \\ & + \partial_{\boldsymbol{q}} \boldsymbol{A}^-_{j+1/2} (\Delta \boldsymbol{q}_{j+1} - \Delta \boldsymbol{q}_j). \end{aligned} \tag{9}$$

Reference [10] uses the telescoping form $\partial_{\boldsymbol{q}} \boldsymbol{A}_{j+1} \Delta \boldsymbol{q}_{j+1} - \partial_{\boldsymbol{q}} \boldsymbol{A}_{j-1} \Delta \boldsymbol{q}_{j-1}$ for the first term. The last four terms are just the usual form of matrix dissipation [11], often written as

$$\left|\partial_{\boldsymbol{q}} \boldsymbol{A}_{j-1/2}\right| (\Delta \boldsymbol{q}_j - \Delta \boldsymbol{q}_{j-1}) - \left|\partial_{\boldsymbol{q}} \boldsymbol{A}_{j+1/2}\right| (\Delta \boldsymbol{q}_{j+1} - \Delta \boldsymbol{q}_j).$$

At convergence $\Delta \boldsymbol{q} = 0$, at which point this dissipation vanishes; the intent of the upwind discretization is simply to accelerate convergence. The converged accuracy is controlled by the differencing of the right side of (4), discussed in the following.

The system of equations (4) is iterated to steady state with symmetric Gauss–Seidel line relaxation, updating $\boldsymbol{q}$ to $\boldsymbol{q} + \Delta \boldsymbol{q}$ after each complete relaxation step. In the present calculations a step consisted of one forward and one backward sweep in each of the j and k directions. Velocity and pressure at points denoted by $\otimes$ are evaluated explicitly by interpolation on the previous iteration.

It is peculiar to the present application that the line lengths are variable, due to the refinement procedure of Fig. 1. Disjoint line segments are packed into a single vector: Thus, if the middle row of Fig. 1 has active points further to the right, they are filled consecutively into the vector $\{\boldsymbol{q}_1, \boldsymbol{q}_2 \ldots\}$, skipping blank and interpolated points. The disjoint line segments are numerically independent. The block matrices $\partial_{\boldsymbol{q}} \boldsymbol{A}$ for the line segments are compressed into block tridiagonal matrices corresponding to the packed solution vector.

A notable aspect of this procedure is that generic linear algebra routines can be used to solve the discrete equations. The only caveat is that each j and each k line can have a different vector length. Storage is indexed on a single variable, which runs through the active points. For instance, let $m_{j,k}$ be a memory location; $v(m_{j,k})$ stores a velocity if j, k is an active point; if j, k is not active, no memory is allocated. With this form of storage, $m_{j,k}$ is defined on the underlying full grid and the increment between neighboring active points is not $\Delta j = \Delta k = 1$. Hence, one needs a large array, $m_{j,k}$, at the finest grid resolution. Then the dependent fields are efficiently stored in smaller arrays, only at nodes that $m_{j,k}$ points to as being active. These are simply matters of bookkeeping, which have no bearing on the fully structured gridding. Memory access becomes consecutive once a line of data is packed into a sequential array; hence, the method can be made cache efficient.

The right side of (4) contains the flux term $\delta_x \boldsymbol{A}$ and $\delta_y \boldsymbol{B}$. A third-order, upwind biased derivative for the former is given by

$$\delta_x \boldsymbol{A} = \frac{1}{6\Delta x}(\boldsymbol{A}_{j-2} - 6\boldsymbol{A}_{j-1} + 3\boldsymbol{A}_j + 2\boldsymbol{A}_{j+1}).$$

This contributes a flux gradient

$$\frac{1}{2\Delta x}(\boldsymbol{A}_{j+1} - \boldsymbol{A}_{j-1}) + \frac{1}{6\Delta x}[2(\boldsymbol{A}^+_j - \boldsymbol{A}^+_{j-1}) - (\boldsymbol{A}^+_{j+1} - \boldsymbol{A}^+_j) - (\boldsymbol{A}^+_{j-1} - \boldsymbol{A}^+_{j-2})] \tag{10}$$

from the positive eigenvalues. The last expression consists of a central difference plus a dissipative term for flux with positive eigenvalues; a similar term is added for the negative eigenvalues to give

$$\begin{aligned}\delta_x \boldsymbol{A} = {} & \frac{1}{2\Delta x}(\boldsymbol{A}_{j+1} - \boldsymbol{A}_{j-1}) + \frac{1}{6\Delta x}[2(\boldsymbol{A}^+_j - \boldsymbol{A}^+_{j-1}) - (\boldsymbol{A}^+_{j+1} - \boldsymbol{A}^+_j) - (\boldsymbol{A}^+_{j-1} - \boldsymbol{A}^+_{j-2})] \\ & + \frac{1}{6\Delta x}[2(\boldsymbol{A}^-_{j+1} - \boldsymbol{A}^-_j) - (\boldsymbol{A}^-_{j+2} - \boldsymbol{A}^-_{j+1}) - (\boldsymbol{A}^-_j - \boldsymbol{A}^-_{j-1})].\end{aligned} \tag{11}$$

Split flux differences like $\boldsymbol{A}^+_{j+1} - \boldsymbol{A}^+_j$ are evaluated from the Jacobian (6) as

$$\boldsymbol{\partial_q} \boldsymbol{A}^+_{j+1/2} \cdot (q_{j+1} - q_j).$$

The centered difference is evaluated without linearization. Then (11) consists of a central difference plus a fourth-order diffusive contribution. This is the procedure used in [10] to introduce higher order numerical dissipation in the residual. It can occur that $j \pm 2$ is a blanked or boundary point; then the stencil for the dissipative term is shifted forward or backward in the j direction.

The first term on the right side of Eq. (11) is a second-order central difference on a uniform grid. At a 2 : 1 grid refinement, say where $x_{j+1} - x_j = 2(x_j - x_{j-1})$, its accuracy becomes first order. At such locations, a second-order centered difference is evaluated as

$$\frac{1}{4\Delta x_j}(\boldsymbol{A}_{j+1} + 3\boldsymbol{A}_j - 4\boldsymbol{A}_{j-1}), \tag{12}$$

where $\Delta x_j = x_{j+1/2} - x_{j-1/2}$. The dissipative term of (11) is not modified for the nonuniform spacing. Hence the local grid refinement can be incorporated into this artificial compressibility scheme with only minor revision.

The viscous diffusion term on the right side of (3) was central differenced and was included in the implicit matrix and residual, on the left and right sides of (4). The split between implicit and explicit contributions is via the Euler implicit treatment: The diffusion term is evaluated at the $n+1$ iteration, with $\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \Delta\boldsymbol{q}$.

Where the grid spacing doubles, a three-point second difference would become

$$\delta^2 u \propto u_{j+1} + 2u_{j-1} - 3u_j,$$

if $x_{j+1} - x_j = 2(x_j - x_{j-1})$. This has first-order accuracy, with a leading order dispersive error. A second-order four-point scheme simply skips the value at x_{j-1},

$$\delta^2 u \propto u_{j+1} + u_{j-2} - 2u_j,$$

assuming $x_j - x_{j-1} = x_{j-1} - x_{j-2}$. However, in the present computations the three-point stencil was retained, even at 2-1 refinements. Doing so gives the viscous term a first-order formal accuracy at jumps in grid spacing.

4. EXAMPLES

A flow computation was performed on the grid of Fig. 5. The full domain extends over $-4 < x < 35$, $0 < y < 6$ with a symmetry condition at $y = 6$. Streamlines, velocity vectors, and convergence history are displayed in Fig. 6. This particular grid is simply block-refined; it was not adapted locally to the solution. It serves primarily to illustrate the effectiveness of the solution algorithm and to show that the solution continues smoothly across the interpolation points. Also, no numerical instabilities were encountered.

Second-order formulas for convection (12) and interpolation (2) were used in this computation. The convergence history of Fig. 6 is for a pseudo-time step of $\Delta\tau = 2.5$ based on free-stream speed and step height of unity. With $\Delta\tau = 10.5$ the same level of convergence (i.e., to single precision) was obtained in only 300 iterations.

The next example assesses the order of accuracy. Flow in a square cavity with a moving upper wall [5] was chosen for this purpose. The Reynolds number based on cavity height H and wall velocity was 1000. A stress singularity occurs where the stationary and moving

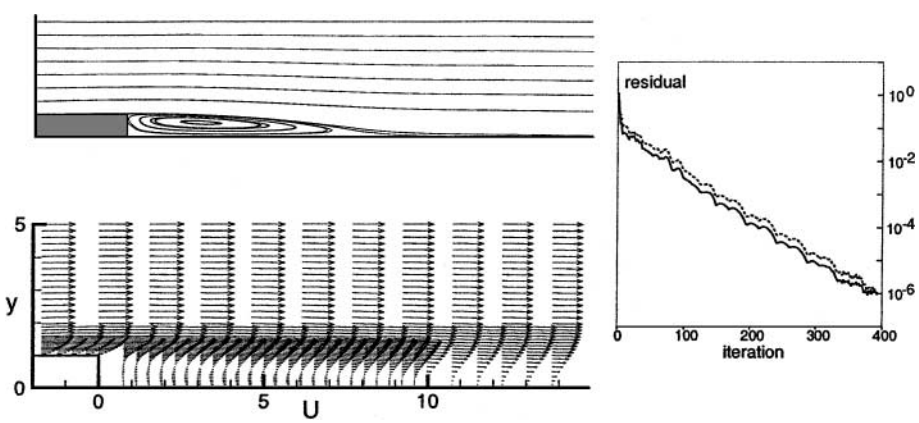


FIG. 6. Streamlines and velocity vectors for a backstep at step height Reynolds number of 200 on the grid of Fig. 5. Only part of the domain is shown. Residual plot shows the maximum absolute residual (solid) and maximum divergence (dashed).

walls meet in the geometry of Ref. [5]. Therefore, for testing convergence, we modified this geometry by leaving a gap of $H/2$ between the moving and stationary walls, with inflow boundary conditions on the left side and outflow on the right (Fig. 7).

Results for uniform grids are reported in Fig. 7a. Errors are measured relative to a solution on a 320×320 grid. The global norms, L_1 and L_2, show nearly second-order accuracy (the slopes were obtained by fitting an exponential though the data points). Lower accuracy is achieved locally (L_∞). However, the maximum error selected by the L_∞ norm occurs near the exit plane, where the outflow and solid wall meet; it is not associated with local refinement. The exit condition consists of zero gradient for velocities and constant pressure. It was imposed by first-order extrapolation; clearly, the global error was not hurt.

The accuracy of the present code on a nonuniform mesh is evaluated in Fig. 7b. Vertical lines were added in the central region of the cavity, but no hanging nodes are present. The scheme remains approximately second order. Slightly larger overall errors were observed if the standard scheme is used without the correction (2), but the accuracy was still second order. This is consistent with the idea that a first-order stencil at jumps in grid spacing does not degrade the global accuracy.

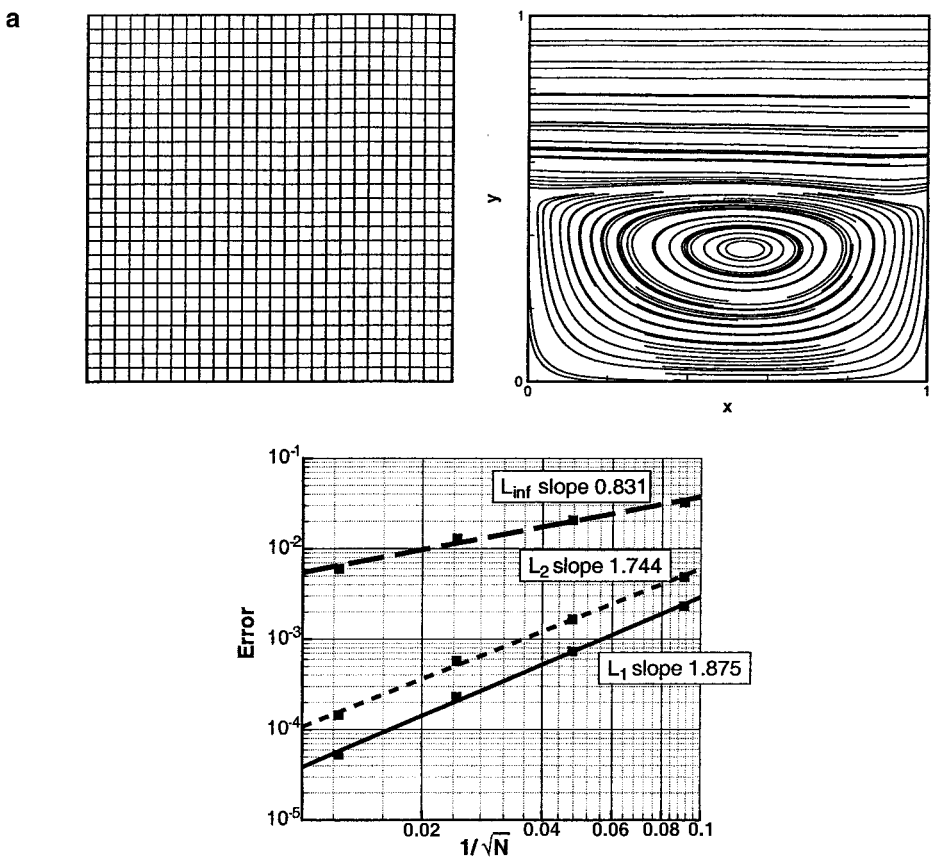
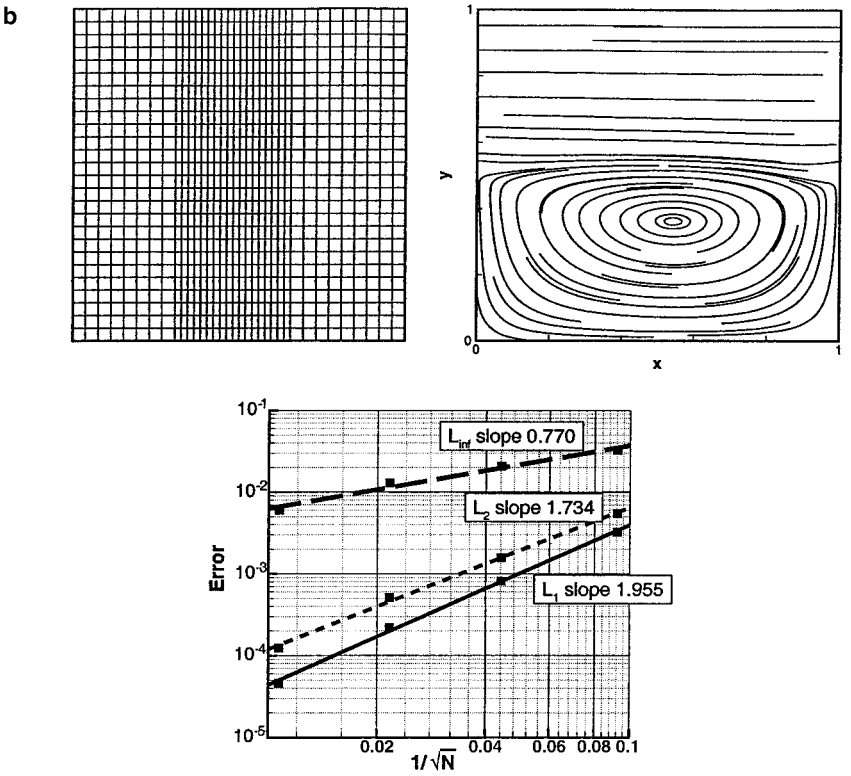


FIG. 7. (a) Uniform mesh. (b) Nonuniform mesh: Vertical grid lines are added in the region $0.3 < x < 0.7$ (no hanging nodes). (c) Streamlines and error norms for the flow in a cavity at $\mathrm{Re} = 1000$. Locally refined mesh: Vertical grid lines and horizontal grid segments are added in the region $0.3 < x < 0.7$ (hanging nodes are interpolated using Eq. (2)).



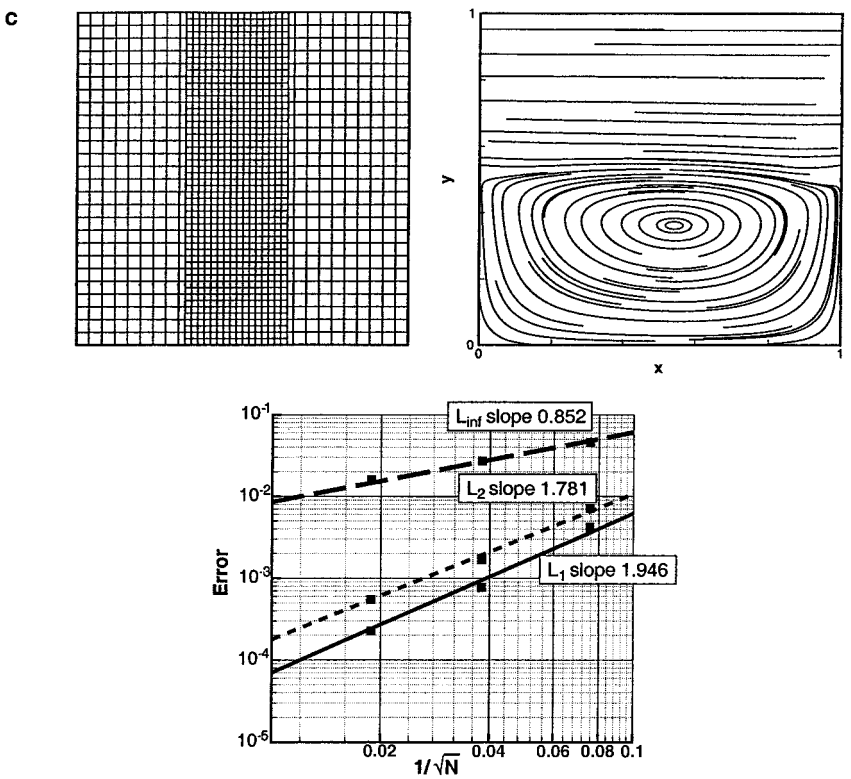


FIG. 7—*Continued*

Finally, in Fig. 7c locally refined grids are tested for order of accuracy. Both vertical grid lines and horizontal grid segments were added in the central region of the cavity. In this case hanging nodes are present and the second-order interpolation formula (2) is used. The second-order accuracy of the method was not changed.

It must be noted that in Figs. 7a,b,c the errors are evaluated as a function of an equivalent grid spacing $\Delta = 1/\sqrt{N}$, where N is the total number of active points. For uniform meshes $\Delta = \Delta x = \Delta y = 1/\sqrt{N}$, but this is a looser definition of spacing in the context of nonuniform grids. The curves in Figs. 7b and 7c would shift to the left or right were grid spacing defined differently, thus preventing a quantitative comparison among the error levels in Figs. 7a–7c. Nevertheless, the slopes would be unchanged, providing a measure of nominal order of accuracy.

As an example of the application of the present technique, an adaptive grid refinement for the original cavity proposed by [5] is reported in Fig. 8. The adaptation function was based on a linear combination of velocity magnitude (V) and pressure (p): The grid is refined where $\phi = (1/2)(10V/V_{\text{in}} + 100p/p_{\max})$ is less than one, where V_{in} is the slip velocity imposed at the top of the driven cavity and $p_{\max}$ is the maximum pressure. The initial (uniform) grid is made up of 20×20 cells and three successive refinement steps were employed. The final mesh is shown at the left in Fig. 8. Active nodes are clustered in the eddies and near the walls. The pressure dependence of the adaptation function, ϕ, was designed for the former; the velocity dependence is to capture the latter. The refinement is distinctly nonisotropic.

The streamlines on the right panel of Fig. 8 show the presence of secondary recirculation regions in the lower corners, in good qualitative agreement with benchmark results [5]. A more quantitative comparison is reported in Fig. 9: Velocity profiles on vertical and horizontal centerlines are reported there for the adapted grid and are compared to a solution on a uniform 100×100 grid. The agreement between these two solutions is quite good. Note that the locally refined solution cuts through the highly irregular grid at the center of the cavity. The grid in Fig. 8 contains 4683 active points; the finest level corresponds to an 80×80 full grid. In Fig. 8 a solution obtained on a uniform 68×68 grid

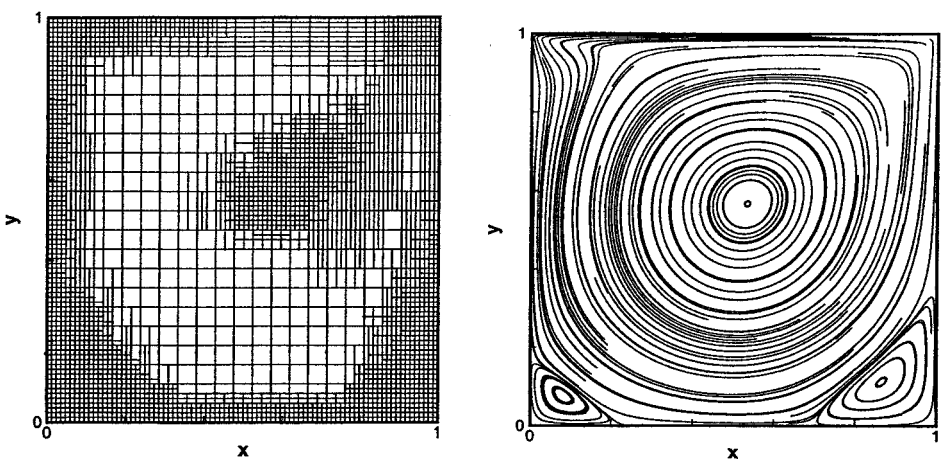


FIG. 8. Computational grid and streamlines for the flow in a square cavity at Re = 1000 using adaptive locally refined grids. Hanging nodes are interpolated using Eq. (2).

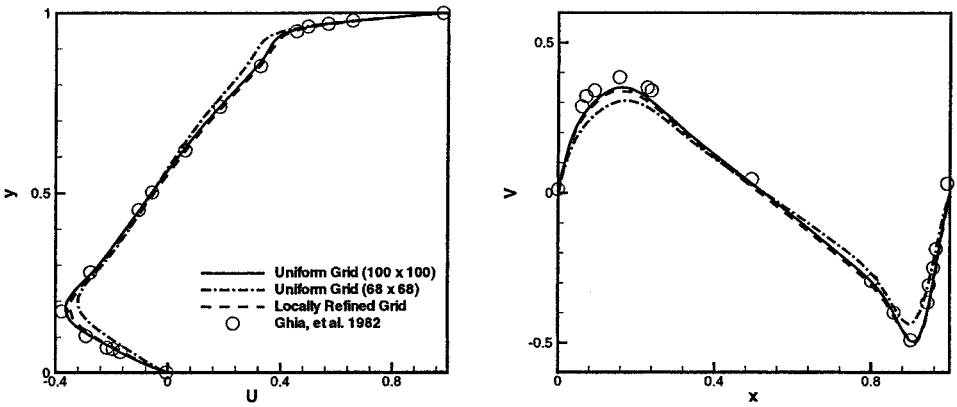


FIG. 9. Velocity components in vertical and horizontal centerline; flow in a square cavity at Re = 1000.

(corresponding to ≈4600 nodes) is reported; this shows the advantage of using locally refined grids.

5. DISCUSSION

It has been shown how local adaptive refinement can be effected on a fully structured grid. A complete grid is imagined to underlie the computation, but flow variables are defined only at active points. Conversely, inactive points are *iblanked*. However, the *iblanking* is largely notional: Variables are not stored, nor are equations solved, at blanked locations, so no computational or memory penalties are incurred. Standard finite-difference schemes apply, as do standard linear algebra solution algorithms. Relatively small modifications need be made to allow for the existence of inactive nodes.

Proof of concept computations were provided for a driven cavity and a backstep. The latter was carried out on a block refined grid; the former invoked local adaptive refinement. The adaptive refinement occurred near boundaries and in the eddies. Comparison to previous studies, and to a uniform fine-grid solution, showed that the present method is effective. Second-order global accuracy was obtained in grid convergence tests. In this paper the method was applied on Cartesian grids; however, it is obviously extensible to curvilinear grids.

Laminar flows were computed to assess the method. Laminar flow provides a clearly defined test. However, one motive for structured grid methods is to facilitate implicit treatment of turbulence models. To complete the story, the application to Reynolds-averaged computation will be illustrated.

The present algorithm was tested on the grid of Fig. 10. The underlying fine grid was concentrated near the walls, with exponential stretching away from the surface. This is required to capture the viscous wall region. The underlying 101×101 grid was *iblanked* to a 4087-point grid; i.e., more than half of the points were deleted.

The lower portion of Fig. 10 is from a RANS v^2–f model computation. It shows skin friction on the lower wall, downstream of the step. Computations on the locally refined grid are compared to a solution on the full grid. In the region $0 < x < 10$ the two grids have the same spacing; in $10 < x < 17$ the locally refined grid is twice as coarse; and in $17 < x < 35$

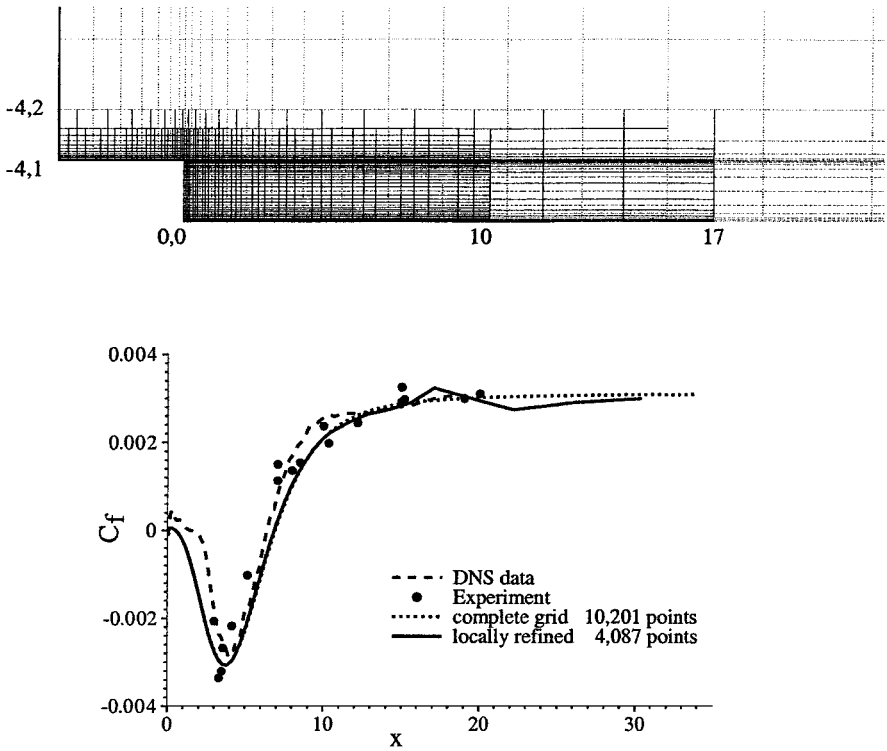


FIG. 10. Block refined, nonuniform grid for RANS. Step height Reynolds number is 5000. Finest grid lies in $-4 < x < 10, 0 < y < 1.5$; medium grid lies in $-4 < x < 17, 0 < y < 2$. The y axis has been magnified by a factor of 2. There is skin friction on the lower wall, beyond the step.

the second grid is four times as coarse: Indeed, combined with the exponential stretching it has become extremely coarse. The skin friction is in excellent agreement with the fine-grid computation (and with data, included for curiosity) until this last region. Other solution fields show similarly good correspondence between locally refined and fully refined cases. The present method clearly is applicable to RANS computation.

ACKNOWLEDGMENTS

This work was supported by the NASA/Stanford Center for Turbulence Research and by the Office of Naval Research.

REFERENCES

1. M. Amato, G. Iaccarino, and L. Paparone, *Application of an Automatic Local Grid Refinement Technique to High-Lift Flows*, ICAS Conf., Sorrento, Italy, 1996.
2. M. J. Aftosmis, M. J. Berger, and J. E. Melton, Adaptive Cartesian mesh generation, in *CRC Handbook of Mesh Generation* (CRC Press, Boca Raton, FL, 1998), pp. 22-1–22-35.
3. J. A. Benek, P. G. Buning, and J. L. Steger, *A 3-D Chimera Grid Embedding Technique*, Technical Paper 85–1523 (AIAA Press, Washington, DC, 1985).
4. H. L. De Cougn and M. S. Shepard, Parallel refinement and coarsening of tetrahedral meshes, *Int. J. Numer. Methods Eng.* **46**, 1101 (1999).

5. U. Ghia, K. N. Ghia, and C. T. Shin, High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, *J. Comput. Phys.* **48**, 387 (1985).
6. R. Hagmeiger, Grid adaptation based on modified anisotropic diffusion equations formulated in the parametric domain, *J. Comput. Phys.* **115**, 169 (1994).
7. R. Hagmeiger and J. Kok, Adaptive generation of structured grids, in *von Karman Inst. for Fluid Dynamics, Lecture Series* 1996–06 (AGARD Publication, 1996).
8. D. J. Mavripilis, Unstructured mesh generation and adaptivity, in *von Karman Inst. for Fluid Dynamics, Lecture Series* 1995–02 (AGARD Publication, 1995).
9. R. W. Noack and D. A. Anderson, Solution-adaptive grid generation using parabolic partial differential equations, *AIAA J.* **28**, 1016 (1990).
10. S. E. Rogers and D. Kwak, Upwind differencing scheme for the time-accurate incompressible Navier–Stokes equations, *AIAA J.* **28**, 253 (1990).
11. J. C. Tannehill, D. A. Anderson, and R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer* (Taylor & Francis, London, 1997).
12. H. Thornburg, B. K. Soni, and K. Boyalakuntlal, A structured grid based solution-adaptive technique for complex separated flows, *Appl. Math. Comput.* **89**, 259 (1998).
13. J. C. Yang and B. Soni, Structured adaptive grid generation, *Appl. Math. Comput.* **65**, 265 (1994).