Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Fluids 33 (2004) 891–916
computers & fluids
www.elsevier.com/locate/compfluid

A block LU-SGS implicit dual time-stepping algorithm for hybrid dynamic meshes

L.P. Zhang [a], Z.J. Wang [b,*]

[a] *Computational Aerodynamics Institute, China Aerodynamics Research and Development Center, P.O. Box 211, Mianyang, Sichuan 621000, China*
[b] *Department of Mechanical Engineering, Michigan State University, 2555 Engineering Building, East Lansing, MI 48824, USA*

Received 26 February 2003; received in revised form 22 September 2003; accepted 6 October 2003

Abstract

A block lower–upper symmetric Gauss–Seidel (BLU-SGS) implicit dual time-stepping method is developed for moving body problems with hybrid dynamic grids. To simulate flows over complex configurations, a hybrid grid method is adopted in this paper. Body-fitted quadrilateral (quad) grids are generated first near solid bodies. An adaptive Cartesian mesh is then generated to cover the entire computational domain. Cartesian cells which overlap the quad grids are removed from the computational domain, and a gap is produced between the quad grids and the adaptive Cartesian grid. Finally triangular grids are used to fill this gap. With the motion of moving bodies, the quad grids move with the bodies, while the adaptive Cartesian grid remains stationary. Meanwhile, the triangular grids are deformed according to the motion of solid bodies with a 'spring' analogy approach. If the triangular grids become too skewed, or the adaptive Cartesian grid crosses into the quad grids, the triangular grids are regenerated. Then the flow solution is interpolated from the old to the new grid. The fully implicit equation is solved using a dual time-stepping solver. A Godunov-type scheme with Roe's flux splitting is used to compute the inviscid flux. Several sub-iteration schemes are investigated in this study. Both supersonic and transonic unsteady cases are tested to demonstrate the accuracy and efficiency of the method.

1. Introduction

The use of unstructured grids in computational fluid dynamics (CFD) has become widespread during the last two decades due to their ability to discretize arbitrarily complex geometries and the

* Corresponding author. Tel.: +1-517-432-9132; fax: +1-517-353-1750.
E-mail addresses: zhanglp@my-public.sc.cninfo.net (L.P. Zhang), zjw@egr.msu.edu (Z.J. Wang).

0045-7930/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compfluid.2003.10.004

flexibility in supporting solution-based grid adaptations to enhance the solution accuracy and efficiency [1–7]. In the early days of unstructured grid development, triangular/tetrahedral grids were employed primarily in dealing with complex geometries. Recently, mixed or hybrid grids including many different cell types have gained popularity because of the improved efficiency and accuracy over pure tetrahedral grids. For example, hybrid prism/tetrahedral grids [8], mixed grids including tets/prism/pyramid/hex cells [9], and adaptive Cartesian grid methods [10–18] have been used in many applications with complex configurations. In addition, solution algorithms for computing steady flows on unstructured and hybrid grids have evolved to a high degree of sophistication. The state-of-the-art spatial discretization algorithm is probably the second-order Godunov-type finite volume method [19]. For time-integration, explicit algorithms such as multi-stage Runge–Kutta schemes are the easiest to implement. Convergence acceleration techniques such as local time-stepping and implicit residual smoothing [1] have also been employed in this context. However, for large-scale problems and especially for the solution of viscous turbulent flows, implicit schemes are required to speed up the convergence rate. In the last decade, significant progress has been made in the development of implicit numerical algorithms for steady flow simulations on unstructured grids [20–24]. For example, a block lower–upper symmetric Gauss–Seidel (BLU-SGS) scheme [24] was shown to dramatically speed up the convergence of complex turbulent flow computations with arbitrary grids, while its memory requirement is comparable to a point implicit scheme. The idea of the BLU-SGS scheme was first suggested by van Leer and Mulder [25]. It appears the idea was never implemented until Chen and Wang [24], and Jameson and Caughey [26] independently rediscovered it, and showed excellent convergence rate either as an implicit solver [24] or as a multi-grid smoother [26]. A unique feature of the BLU-SGS scheme is that it retains some of the nonlinearities of the original implicit scheme, resulting in fast convergence.

The success demonstrated by unstructured grids for steady flow problems has prompted their applications to unsteady moving boundary flow problems. For a moving boundary flow problem, the computational grids must move with the moving boundaries. The most straightforward approach is to deform the computational grid locally using a spring-analogy type algorithm to follow the motion of the moving boundaries [27]. The approach is very efficient because it does not require solution interpolation. A disadvantage of the approach is that the grid integrity can be destroyed by large motions or shear-type of boundary motions. To remedy this drawback, local remeshing can be applied whenever the grid becomes too skewed. With local remeshing, solution interpolations from the old to the new grid become necessary. The hybrid approach of combining grid deformation with grid local remeshing seems to be the state-of-the-art in handling moving boundary problems, and has been used successfully for a variety of applications [28]. Most of the unstructured simulations for moving boundary problems are performed with triangular or tetrahedral grids only [27–29]. The use of adaptive Cartesian grid with cell-cutting for inviscid moving boundary flow problems was pursued and demonstrated in both 2D [10] and 3D [18]. In this paper, we advocate a hybrid adaptive Cartesian/quad/triangular grid approach for dynamic moving boundary flow problems in 2D. In 3D, the approach becomes adaptive Cartesian/prism/tetrahedral grid-based, which of course is much more complicated to implement because the generation of prismatic grids around complex geometries can still fail. In the current study, we focus on demonstrations in 2D only. There are several reasons why an adaptive Cartesian grid is used for moving boundary problems: (1) Cartesian cells are more efficient in filling space given a certain length scale than triangular/tetrahedral cells; (2) searching operations can be performed

very efficiently with the quadtree data structure; (3) solution-based and geometry-based grid adaptations are straightforward to carry out. The grid generation process is as follows. Body-fitted quad grids are generated first near solid bodies to resolve viscous boundary layers. An adaptive Cartesian grid is then generated to cover the outer domain. The Cartesian cells which overlap the quad mesh are marked and removed from the computational domain. As a result, a gap is generated between the Cartesian and quad grids. Triangular grids are then used to fill the gap. If the bodies move, the quad grids move with the bodies, while the Cartesian grid remains stationary. Meanwhile, the unstructured triangular grids are deformed according to the motions of the bodies with a 'spring' analogy approach. If the triangular grids become too skewed because of the deformation, the triangular grids are then regenerated, and the solutions are also interpolated from the old to the new grids. The advantages of above hybrid grid method include: (1) more efficient than the fully unstructured triangular grids, especially for viscous flow simulations; (2) more accurate for viscous flow computations because of the high-quality grids in the boundary layer; (3) only a small local region needs to be remeshed resulting in less data interpolation and less errors caused by data interpolations.

The solution efficiency of the time-integration algorithm for a moving boundary flow problem becomes more important than for a steady problem because the unsteady residual should be driven close to zero at each time step. For viscous flow problems with highly clustered computational grids, an explicit time-marching method is clearly out of the question because of the time-step limit imposed by an explicit scheme. Therefore, only implicit schemes are considered in this study. A very popular and effective approach for unsteady flows is the dual time stepping algorithm of Jameson [30]. In this approach a pseudo-time is added to the unsteady residual, and many algorithms developed for accelerating the convergence rate of steady problems can be used directly for the pseudo-time. As we mentioned earlier, one has to drive the unsteady residual to zero (or at least to truncation error) at each time step when an implicit scheme is used to compute unsteady flows. Many of the successful implicit solvers for steady flows problems can be used to converge the unsteady residual, and there are successful examples in the literature. The convergence rate of this 'inner' solver determines the overall efficiency of the solution algorithm. In this study, we will extend and test the successful BLU-SGS scheme [24–26] to hybrid dynamic grids for moving boundary problems. Comparisons will also be made between explicit, point implicit and BLU-SGS solvers.

The paper is organized as follows. In the next section, the hybrid adaptive Cartesian/quad/triangular grid generation approach will be presented, together with illustration examples. After that, the finite volume, Godunov-type second-order dual time-stepping method for dynamic grids is described. Details of the BLU-SGS inner iteration solver are presented. In Section 4, several unsteady moving boundary problems are computed. Temporal and spatial refinement studies are performed to ensure the computational solutions are time-step and grid-independent. Computational results are compared with experimental data whenever possible. Finally conclusions from this study are summarized in Section 5.

2. Grid generation strategy

In this study, a hybrid adaptive Cartesian/quad/triangular grid approach is adopted to discretize a complex computational domain. Body-fitted quad grids are generated first near solid

bodies with an advancing layer method [6,8] by marching in the surface normal direction. When the aspect ratio of the quad cell reaches a pre-defined scale (for example, 0.6) or the width of remaining gap between solid bodies is only one or two times of the local grid size, the local advancing layer procedure stops. Then an adaptive Cartesian grid is generated to cover the outer domain by a modified quadtree method [31]. This adaptive Cartesian grid is generated by recursively subdividing a large root cell covering the entire computational domain. The grid resolution of the adaptive Cartesian grid automatically matches that of the quad grids near the outer layer of the quad grids. Of course, Cartesian cells close and inside the last advancing layer of the quad grids are removed from the computational domain. Usually the gap width between the quad grid and the adaptive Cartesian grid is set to several times of the local grid size (usually six times or less). Then triangular grids are used to fill the gap using an advancing front method (AFM) [32–34]. To control the grid distribution smoothly, a structured background grid [35] is employed, and some controlling point or line sources are specified in the field according to the configuration of interest. Finally, a 'spring' analogy approach is employed to smooth the initial grids generated with the AFM.

Due to the motion of the moving boundaries, the grids at time level n is different from that at time level $n+1$. The grid at time $n+1$ is usually generated by deforming the grid at time n with the grid connectivity or topology remaining the same. With our hybrid adaptive Cartesian/quad/triangular grid approach, we adopt the following strategy. The quad grids around moving bodies move (translate or rotate) with the moving bodies, while the adaptive Cartesian grid remains stationary. Meanwhile, the unstructured triangular grids are deformed according to the motions of the moving bodies with a 'spring' analogy approach [27]. Because the quad grids are not deformed, the quality of the quad grids remains the same, which is obviously a benefit for computing the viscous boundary layer. In addition, the outer adaptive Cartesian grid always remains stationary. When large motions occur, the quad grids may cross into the flow cells of the original adaptive Cartesian grid, or the triangular grids become too skewed. In this case, a remeshing step is undertaken. The holes in the adaptive Cartesian grid are regenerated-based on the new outer boundary of the quad grids. New triangular grids are generated to fill the gap between the adaptive Cartesian and quad grids. The flow solutions are also interpolated from the old to the new grids at the time when the remeshing step is taken. The interpolation algorithm takes the following steps:

- Compute the cell centroids of the new grids, and identify the cells in the old grids, which bound the centroids of the new grids.
- Compute the solution variables at the cell centroids of the new grids assuming a linear distribution of the solution in the bounding cell using the reconstructed solution gradients.

The interpolation has the property of linearity-preserving, i.e., any linear function is preserved in the interpolation, which is compatible with the second-order accuracy of the flow solver. However, the interpolation does not strictly conserve mass, momentum or energy. It is believed that these conservation errors are of the same order of truncation errors and do not affect the shock-capturing ability of the flow solver since the solution interpolation is not performed across a time step. Note that this remeshing step is always local, and occurs near the triangular grids. Therefore, this approach is expected to be very efficient.

Several examples are used here to illustrate the basic grid generation algorithm. The first example is a 2D fighter aircraft with a store. The store separates from the fighter with a given trajectory. Fig. 1(a) shows the initial hybrid grid over the configuration, while Figs. 1(b)–(d) display the close-up views near the fighter body, the nose of the fighter, and the head of the store, respectively. Note that the grids are very smooth in the entire computational domain, and the grid quality is adequate even in the narrow gap between the fighter and the store. Figs. 1(e) and (f) show the grids when the store is far away from the fighter. During the separation, the body-fitted quad grid around the store moves with the store. The control sources on the store for the background grid also move with the body, resulting a new smooth grid distribution. Note that the grid quality is still satisfactory even after very large motions.

The second example is a double airfoil configuration, which is made up by the authors to test the ability of present hybrid grid generation method in handling narrow gaps and shear-type of grid motions. The main airfoil is a RAE2822 airfoil, and the small one is a NACA0012, which is scaled to one-third of the main chord. Fig. 2(a) shows the hybrid grid over the combined configuration. Figs. 2(b)–(d) display the close-up views of the same grid, while Figs. 2(e) and (f) present the hybrid grids when the small airfoil separates away from the cavity. Once again, the present hybrid grid generator produces high-quality grids for this multi-body geometry.

It may seem to some readers that a complex flow solver is needed to handle the quad, adaptive Cartesian and triangular grid cells. In fact, the different grid cells are treated uniformly as arbitrary polygons using a face-based data structure, which is very efficient and can handle arbitrary cells just as efficiently as cells of a particular type, e.g., quad cells. The grid generator and the flow solver are two independent modules, which communicate through the use of files. Shell scripts are employed to automate the simulation of moving boundary flow problems.

3. Numerical method

3.1. Finite volume method for dynamic grids

The time-dependent Reynolds-averaged Navier–Stokes equations for dynamic grids can be expressed in the integral form as

$$\frac{\partial}{\partial t}\int_V Q\,\mathrm{d}V + \oint_S (F^{\mathrm{i}}(Q) - Q\boldsymbol{v_g}\cdot\boldsymbol{n})\,\mathrm{d}S = \oint_S F^{\mathrm{v}}(Q)\,\mathrm{d}S, \tag{3.1}$$

where S is the surface surrounding the control volume V, $\boldsymbol{n}$ is the out-going unit normal of S, $\boldsymbol{v_g}$ is the velocity of S, and Q is the vector of conserved variables, F^{i} is the inviscid and F^{v} the viscous flux vectors. The eddy viscosity for turbulent flow is calculated by the standard κ–ε turbulence model with a wall function [36]. The governing equations for inviscid flow and for fixed control volumes are only subsets of Eq. (3.1). If we integrate Eq. (3.1) in a polygonal control volume V_i, we obtain

$$\frac{\partial}{\partial t}(QV_i) + \sum_f (F^{\mathrm{i}}(Q) - Qv_{gn})_f\,\mathrm{d}S_f = \sum_f F_f^{\mathrm{v}}(Q)\,\mathrm{d}S_f, \tag{3.2}$$

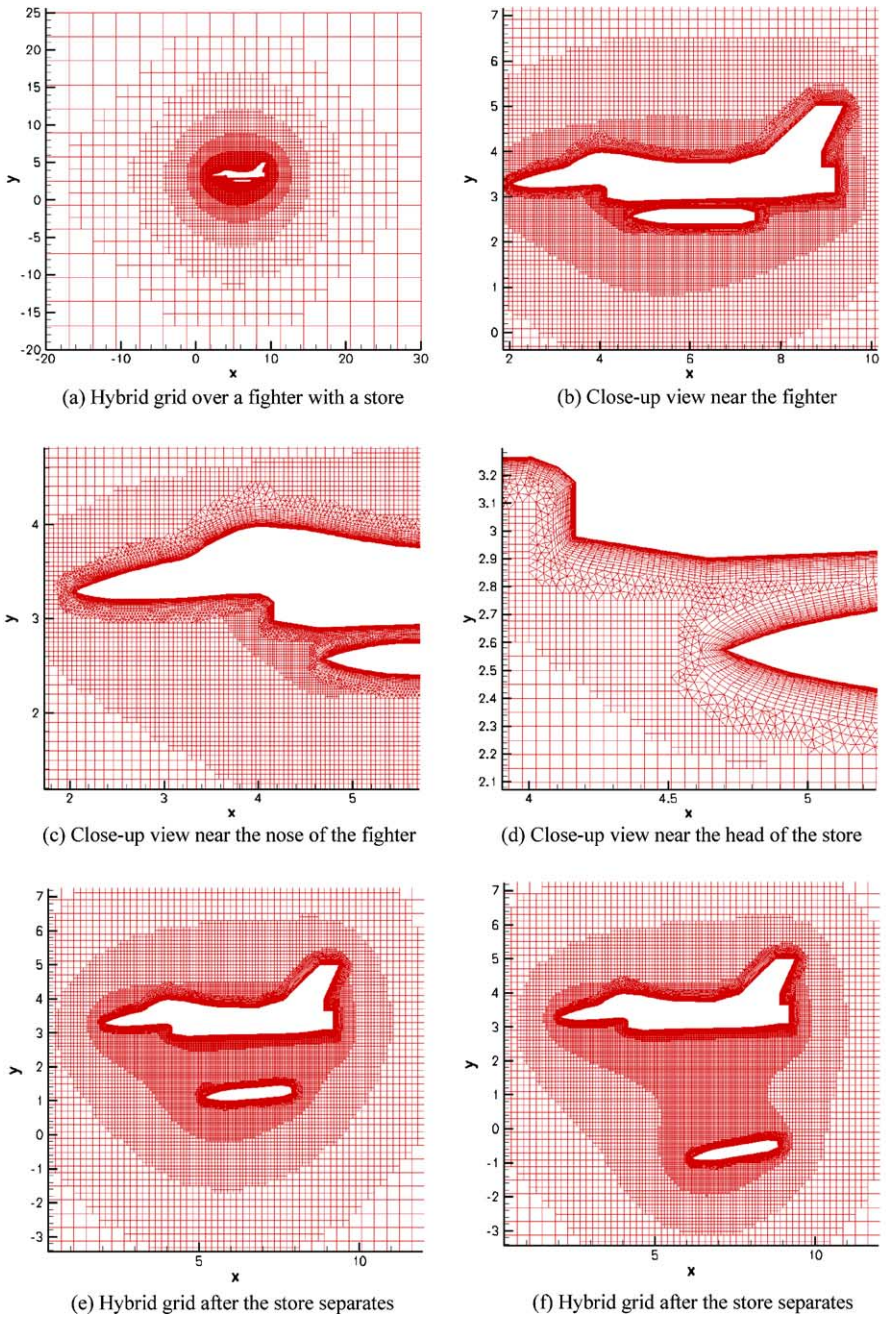


Fig. 1. Hybrid grid for a store-separation problem at different times.

where the summation index f represents all the faces surrounding control volume V_i, and $v_{gn} = \boldsymbol{v_g} \cdot \boldsymbol{n}$. The inviscid flux is calculated using Roe's approximate Riemann solver [37] with reconstructed state variables at both sides of a face. A least square linear reconstruction scheme of

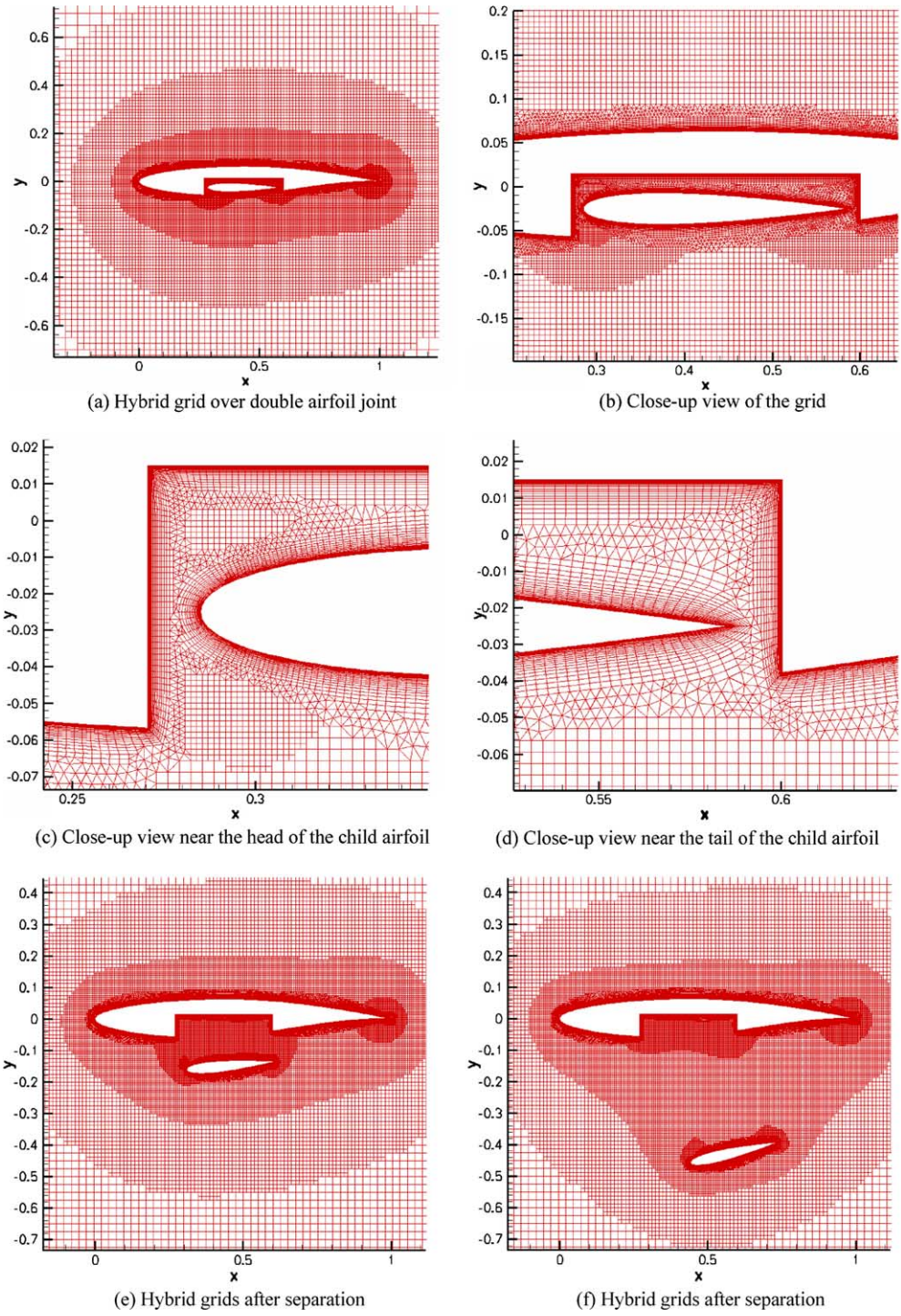


Fig. 2. Hybrid grid for a store-separation problem at different times.

the primitive variables is used. Venkatakrishnan's limiter [38] is applied to the reconstruction of the primitive variables to reduce spurious oscillations near discontinuities. This limiter has a better convergence property than other limiters because small-scale oscillations are allowed. There is one user specified parameter ε in Venkatakrishnan's limiter, which defines the small-scale oscillations. This parameter is difficult to select because it depends not only on the solution variable but also on the mesh size. In [14], Wang redefined the parameter as the following

$$\varepsilon = \varepsilon'(q^{\max} - q^{\min}),$$

where $q^{\max}$ and $q^{\min}$ are the maximum and minimum of the primitive variable over the entire computational domain, and ε' is a nondimensional parameter in the range of [0.01, 0.2]. Unless otherwise noted, the parameter ε' is chosen to be 0.15 for the inviscid simulations, and 0.1 for the viscous simulations to be shown later. A ε' value of 0.025 caused the convergence to stall after 2 orders of magnitude. For the viscous terms, an efficient second-order centered scheme is employed. Details of the flow solver are contained in [14,15].

The conservation of a constant flow is a necessary condition for any viable numerical scheme. Otherwise mass, momentum or energy would be produced unphysically by the numerical simulation. If we examine Eq. (3.2), in order to preserve a uniform free stream, we must have:

$$\frac{\partial V_i}{\partial t} = \sum_f v_{gn}\,\mathrm{d}S_f. \tag{3.3}$$

This is the so-called Geometric Conservative Law [39,40] in its semi-discretized form. Assume that the grid velocity is computed at time level $n + 1/2$. Then we can use the following time discretization to achieve second-order accuracy:

$$\frac{V_i^{n+1} - V_i^n}{\Delta t} = \sum_f v_{gn}\,\mathrm{d}S_f. \tag{3.4}$$

Instead of having the grid velocity v_{gn} satisfy Eq. (3.4), we utilize the equation to calculate v_{gn}. In this case, we are sure that GCL is guaranteed. To this end, we employ a simple fact: the volume that a cell sweeps over is equal to the total of the volumes swept by its faces, i.e.,

$$V_i^{n+1} - V_i^n = \sum_f \Delta V_f, \tag{3.5}$$

where ΔV_f represents the volume swept by face f. Comparing Eqs. (3.4) and (3.5), we arrive at the following equation:

$$\Delta V_f = \Delta t v_{gn}\,\mathrm{d}S_f \quad \text{or} \quad v_{gn} = \frac{\Delta V_f}{\Delta t\,\mathrm{d}S_f}. \tag{3.6}$$

3.2. Time-integration algorithm

Once the fluxes are evaluated for each cell face using the preceding finite volume scheme, the semi-discrete form of the governing equations is then integrated in time. For convenience, we rewrite Eq. (3.2) as the following nonlinear system:

$$\frac{\partial (QV)_i}{\partial t} + R_i(Q) = 0, \tag{3.7}$$

where R_i is the residual given by

$$R_i(Q) = \sum_f (F^{\mathrm{i}}(Q) - Qv_{gn} - F^{\mathrm{v}}(Q))_f \, \mathrm{d}S_f. \tag{3.8}$$

The easiest time-integration algorithm for Eq. (3.7) is the explicit multi-stage Runge–Kutta method. However for viscous flow computational, the heavily clustered mesh imposes a too severe time step limit. We therefore employ the following family of implicit schemes

$$\frac{Q_i^{n+1}V_i^{n+1} - Q_i^n V_i^n}{\Delta t} + (1-\theta)R_i(Q^{n+1}) + \theta R_i(Q^n) = 0. \tag{3.9}$$

If $\theta = 0$, the scheme is the backward Euler method. If $\theta = 1/2$, the resulting scheme known as the Crank–Nicolson method is second-order accurate in time. Eq. (3.9) represents a nonlinear system of coupled equations, which has to be solved at each time step. It can be solved by introducing a pseudo-time variable τ,

$$\frac{\partial (QV)_i}{\partial \tau} + R_i^*(Q) = 0 \tag{3.10}$$

and 'time-marching' the solution using local pseudo-time $\Delta\tau$, until Q converges to Q^{n+1}. In (3.10), Q is the approximation of Q^{n+1} and the unsteady residual $R_i^*(Q)$ is defined as

$$R_i^*(Q) = \frac{Q_i V_i^{n+1} - Q_i^n V_i^n}{\Delta t} + (1-\theta)R_i(Q) + \theta R_i(Q^n). \tag{3.11}$$

Obviously, Eq. (3.11) can be solved by using a variety of numerical schemes including the explicit multi-stage Runge–Kutta method [1]. Note that this dual time method with an explicit inner iteration scheme should be many times faster than the explicit time-marching method because local time-stepping in the pseudo-time can be used to accelerate the convergence rate. Of course the best efficiency is expected to be achieved by an implicit inner iteration schemes. The following three inner iteration schemes are considered in the present study.

3.2.1. Explicit Runge–Kutta inner iteration scheme

Applying an explicit Runge–Kutta scheme for the pseudo-time in (3.11), we obtain:

$$\begin{cases} Q_i^{(0)} = Q_i^n, \\ Q_i^{(k)} = Q_i^{(0)} - \alpha_k \dfrac{\Delta t}{V_i^{n+1}} R_i^*(Q^{(k-1)}) \quad \text{where} \quad \alpha_k = \dfrac{1}{m-k+1} \;\; k = 1, 2, \ldots, m, \\ Q_i^{n+1} = Q_i^{(m)}. \end{cases} \tag{3.12}$$

The indices n, m, and k indicate the time level, the order of the Runge–Kutta method, and a dummy index, respectively.

3.2.2. Implicit BLU-SGS inner iteration scheme

For simplicity of presentation, we assume $\theta = 1$. The extension to the case of $\theta = 1/2$, which has second-order in time, is straightforward. Discretizing the pseudo-time in (3.10) with a backward Euler scheme, we obtain the following equation:

$$\frac{V_i^{n+1}(Q_i^{(m+1)} - Q_i^{(m)})}{\Delta\tau} + \frac{V_i^{n+1}Q_i^{(m+1)} - V_i^n Q_i^n}{\Delta t} + \sum_f [\widetilde{F}_f^{\mathrm{i}}(Q^{(m+1)}) - F_f^{\mathrm{v}}(Q^{(m+1)})]\,\mathrm{d}S_f = 0, \tag{3.13}$$

where indices n, m indicates the real time and the pseudo-time levels. Here we let $Q^{(0)} = Q^n$ and the converged solution is then Q^{n+1}, and

$$\widetilde{F}^{\mathrm{i}}(Q^{(m)}) = F^{\mathrm{i}}(Q^{(m)}) - Q^{(m)} v_{gn}. \tag{3.14}$$

Eq. (3.13) can be rewritten further in the following delta form:

$$\frac{V_i^{n+1}\,\Delta Q_i^{(m)}}{\Delta\tau} + \frac{V_i^{n+1}\,\Delta Q_i^{(m)}}{\Delta t} + \sum_f [\Delta\widetilde{F}^{i(m)} - \Delta F^{v(m)}]_f\,\mathrm{d}S_f = -R_i^*(Q^{(m)}), \tag{3.15}$$

where

$$\begin{aligned} \Delta\widetilde{F}^{\mathrm{i}(m)} &= \widetilde{F}^{\mathrm{i}}(Q^{(m+1)}) - \widetilde{F}^{\mathrm{i}}(Q^{(m)}), \\ \Delta F^{\mathrm{v}(m)} &= F^{\mathrm{v}}(Q^{(m+1)}) - F^{\mathrm{v}}(Q^{(m)}). \end{aligned} \tag{3.16}$$

If we assume that the inviscid and viscous fluxes depend only on the state variables at the two cells sharing the face, the flux differences can be approximated with the following formula:

$$\Delta\widetilde{F}^{\mathrm{i}(m)} = [\widetilde{F}^{\mathrm{i}}(Q_i^{(m+1)}, Q_j^{(m+1)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m+1)})] + [\widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m+1)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)})] \tag{3.17}$$

and

$$\Delta F^{\mathrm{v}(m)} = [F^{\mathrm{v}}(Q_i^{(m+1)}, Q_j^{(m+1)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m+1)})] + [F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m+1)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)})], \tag{3.18}$$

where the subscripts i and j denote the current cell under consideration and its neighbor cell that shares face f, respectively. Linearizing the first terms on the right-hand sides of Eqs. (3.17) and (3.18), we have

$$\widetilde{F}^{\mathrm{i}}(Q_i^{(m+1)}, Q_j^{(m+1)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m+1)}) \approx \frac{\partial\widetilde{F}^{\mathrm{i}}}{\partial Q_i}\Delta Q_i^{(m)}, \tag{3.19}$$

$$F^{\mathrm{v}}(Q_i^{(m+1)}, Q_j^{(m+1)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m+1)}) \approx \frac{\partial F^{\mathrm{v}}}{\partial Q_i}\Delta Q_i^{(m)}. \tag{3.20}$$

Substituting Eqs. (3.17)–(3.20) back to Eq. (3.15), we obtain:

$$\begin{aligned} &D\,\Delta Q_i^{(m)} + \sum_f [\widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(m)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)})]_f\,\mathrm{d}S_f \\ &\quad - \sum_f [F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(m)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)})]_f\,\mathrm{d}S_f = -R_i^*(Q^{(m)}), \end{aligned} \tag{3.21}$$

where

$$D = \left(\frac{V_i^{n+1}}{\Delta t} + \frac{V_i^{n+1}}{\Delta \tau} \right) I + \sum_f \left[\frac{\partial \widetilde{F}_f^{\mathrm{i}(m)}}{\partial Q_i} - \frac{\partial F_f^{\mathrm{v}(m)}}{\partial Q_i} \right] \mathrm{d}S_f \tag{3.22}$$

and I is the identity matrix.

In the original LU-SGS approach [41], the first-order numerical flux vectors in the left-hand side of Eq. (3.21) are chosen as

$$\widetilde{F}^{\mathrm{i}} - F^{\mathrm{v}} = \tfrac{1}{2}(\widetilde{F}_i + \widetilde{F}_j - \lambda_{ij}(Q_j - Q_i)), \tag{3.23}$$

where λ_{ij} is the spectral radius of the flux Jacobean matrix at the cell face:

$$\lambda_{ij} = |\boldsymbol{v} \cdot \boldsymbol{n}| + a + \frac{(2\mu + \mu_t)}{\rho |\boldsymbol{n} \cdot (\boldsymbol{r_j} - \boldsymbol{r_i})|}, \tag{3.24}$$

where $\boldsymbol{r_i}$ and $\boldsymbol{r_j}$ are the position vectors of cell centers of cell i and cell j, respectively, ρ is the density, a is the speed of sound, μ and μ_{t} are dynamic and turbulent viscosities, respectively. Then matrix D changes into a diagonal matrix and Eq. (3.21) degenerates into a set of scalar equations.

In order to improve the convergence rate of the original LU-SGS, BLU-SGS keeps the diagonal block of the implicit system, and then use forward and backward sweeps to include the implicit contributions from the off-diagonal blocks. The sweep procedure is as the following:

Forward sweep:

$$\begin{aligned} & D\Delta Q_i^{(*)} + \sum_{f \in L(i)} [\widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{*}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f \\ & - \sum_{f \in L(i)} [F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(*)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f \\ & + \sum_{f \in U(i)} [\widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(m-1)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f \\ & - \sum_{f \in U(i)} [F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(m-1)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f = -R_i^*(Q^{(m)}). \end{aligned} \tag{3.25}$$

Backward sweep:

$$\begin{aligned} & D\Delta Q_i^{(m)} + \sum_{f \in L(i)} [\widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(*)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f \\ & - \sum_{f \in L(i)} [F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(*)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f \\ & + \sum_{f \in U(i)} [\widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(m)}) - \widetilde{F}^{\mathrm{i}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f \\ & - \sum_{f \in U(i)} [F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)} + \Delta Q_j^{(m)}) - F^{\mathrm{v}}(Q_i^{(m)}, Q_j^{(m)})]_f \, \mathrm{d}S_f = -R_i^*(Q^{(*)}), \end{aligned} \tag{3.26}$$

where $L(i)$ and $U(i)$ represent the lower and upper neighbor cells of cell i according to the cell ordering. The initial state variables for the inner sweep are set to be the same as those at the last time step, i.e., $\Delta Q^{(0)} = 0$. Eqs. (3.25) and (3.26) are then solved efficiently with an exact LU decomposition method.

3.2.3. Point implicit inner iteration scheme

In the point implicit, only the main diagonal block is retained, i.e.,

$$D\Delta Q_i^{(m)} = -R_i^*(Q^{(m)}). \tag{3.27}$$

In all of the above inner iteration schemes, it is desirable to drive the unsteady residual $R_i^*(Q)$ close to zero. It is rarely necessary to drive $R_i^*(Q)$ to machine zero because of the inherent truncation error. Through extensive numerical tests, it was found that one only needs to reduce the unsteady residual by 2–3 orders. We therefore use both a convergence tolerance and the maximum number of inner iterations to control the inner iteration scheme.

3.3. Boundary conditions

In order to treat the boundary cells as transparently as possible, a ghost cell is generated for each boundary cell. Then the solution variables at the ghost cell are computed from the boundary cell according to the physical boundary condition. For a steady inviscid flow, the velocity components at the ghost cell for a solid wall boundary are computed as:

$$u_{\text{ghost}} = u - 2n_x v_n \quad v_{\text{ghost}} = v - 2n_y v_n, \tag{3.28}$$

where v_n is the normal velocity given by

$$v_n = un_x + vn_y. \tag{3.29}$$

Meanwhile, the density and pressure of the ghost cell are set to be the same as those of the boundary cell.

For unsteady moving boundary problems, the condition must be adjusted since the boundary face is moving. Then the normal velocity should be modified as

$$v_n = un_x + vn_y - v_{gn}. \tag{3.30}$$

Similarly for an unsteady viscous surface boundary, the velocity components at the ghost cell are computed using the following equation:

$$u_{\text{ghost}} = -u + 2n_x v_{gn} \quad v_{\text{ghost}} = -v + 2n_y v_{gn}. \tag{3.31}$$

In the far field, a characteristic analysis based on Riemann invariants is used to determine the values of the flow variables on the outer ghost cells. This analysis correctly accounts for wave propagations in the far field, which is important for rapid convergence to steady-state and serves as a 'nonreflecting' boundary condition for unsteady applications.

4. Numerical results

Several moving boundary flow problems are simulated to demonstrate the developed grid generation and solution methodology. All the CPU times were obtained on a single processor 2.8 GHz Pentium IV computer running the Linux operating system. From many numerical tests, it was found that the unsteady residual must be reduced by 2–3 orders using the inner iteration solver. More specifically, the inner iteration tolerance is set at 0.01 (i.e., 2 order reduction in the

unsteady residual) for the inviscid flow simulations, while the tolerance is set at 0.001 for turbulent flow simulations. The maximum number of inner iterations for the BLU-SGS solver is set at 30. For the BLU-SGS solver, a maximum of three forward–backward sweeps are employed. The test cases are presented next.

4.1. Moving cylinder with Mach 4

This case was selected as a validation case to test the dynamic grid implementation. A steady-state problem of supersonic flow around a cylinder (with respect to a reference frame fixed on the cylinder) was simulated as a moving boundary problem of a cylinder traveling at Mach 4 through stationary air. After the initial transients, the flow field around the moving cylinder should settle down, and should become 'steady' with respect to the cylinder. A major feature of this flow problem is a bow shock in front of the moving cylinder. For comparison purposes, this problem was also run in the steady mode. For the moving body simulation, a sequence of hybrid computational grids at different times are generated, and displayed in Fig. 3. These grids have approximately a total of 11 K cells, with about 2.3 K quad cells, 3.7 K triangular cells and 5.0 K Cartesian cells. The pressure contours at the corresponding times are also shown in Fig. 3. Note that a bow shock is generated from the wall when the cylinder starts to move. Later the shock moves ahead of the moving cylinder. Finally the bow shock remains at a fixed location relative to the cylinder. In Fig. 4, an enlarged view of the computational grid is plotted to show the details. The pressure distributions along the cylinder surface from both the steady state and moving body simulations are compared in Fig. 5. It is obvious that the agreement is very good, indicating that the implementation of the dynamic grid solver is successful.

4.2. Inviscid flow over an oscillating NACA0012 airfoil

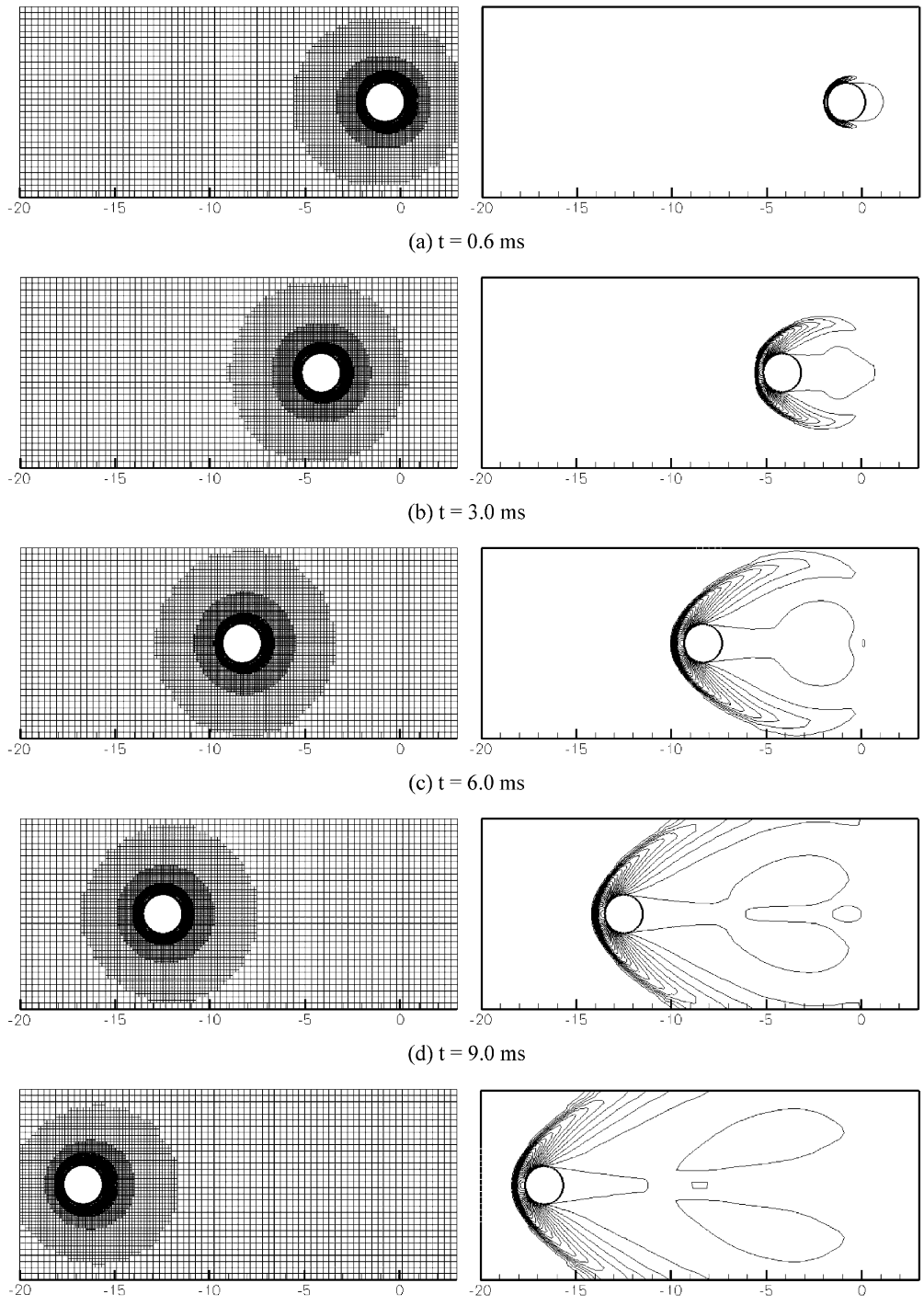
This unsteady test case has been used extensively in the literature for validation and demonstration studies because of the availability of experimental data [42]. The geometry is the well-known NACA0012 airfoil, which undergoes harmonic pitching motion about the quarter chord with the following time-dependent angle of attack

$$\alpha = \alpha_m + \Delta a \sin \omega t, \tag{4.1}$$

where α_m is the mean angle of attack; and $\Delta\alpha$, the oscillation amplitude. The reduced frequency, which is an important similarity parameter for this unsteady problem, is defined as

$$\kappa = \frac{\omega c}{2U_\infty}, \tag{4.2}$$

where U_∞ is the free stream velocity and c, the chord length of the airfoil. Since the flow is attached, it is assumed inviscid. The following flow parameters are chosen: $\alpha_m = 0.016°$, $\Delta\alpha = 2.51°$, $\kappa = 0.0814$, and $M_\infty = 0.755$. The unsteady calculation was started from a steady-state solution at the same Mach number with the mean angle of attack. Several different views of the initial computational grid are shown in Fig. 6. The hybrid grid has a total of 8373 cells, with 1384 quad cells, 2877 triangular cells, and 4122 adaptive Cartesian cells. Even though the flow is assumed



(a) t = 0.6 ms

(b) t = 3.0 ms

(c) t = 6.0 ms

(d) t = 9.0 ms

(e) t = 12 ms

Fig. 3. Computational grids and computed pressure contours at different times.

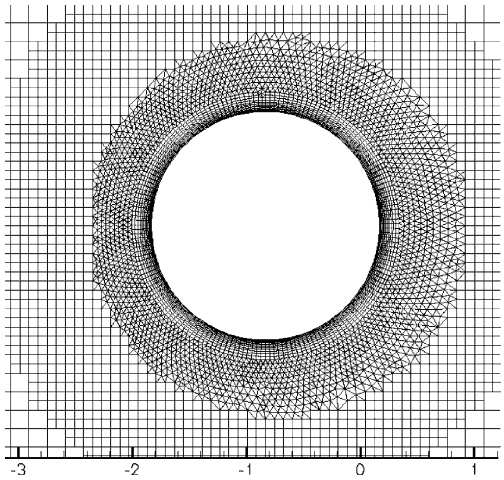


Fig. 4. Close-up view of the hybrid computational grid near the cylinder.

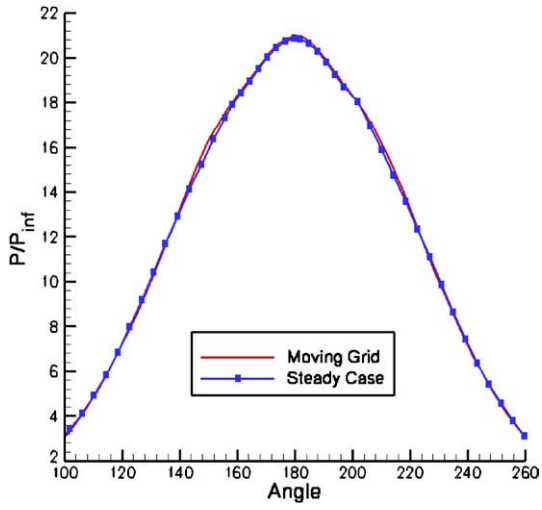


Fig. 5. Comparison of surface pressure distribution computed from steady and moving body simulations.

inviscid flow, the use of the quad cells near the moving body ensures that the computational grid has high-quality near the wall and throughout the computational domain.

The computed pressure contours at several typical times are displayed in Fig. 7. Note that a shock wave appears and disappears near the upper and lower surface of the airfoil with the motion. The time history of the lift coefficient vs. the time-dependent angle of attack using 16, 32 and 64 time steps per cycle is displayed in Fig. 8. It is clear from this figure that the computational solution is acceptable with only 16 time steps per cycle. The solutions are almost on top of each other when 32 and 64 steps are taken during each cycle. Also shown is a comparison with the experimental data of Landon [42]. Note that the difference between the computed and the

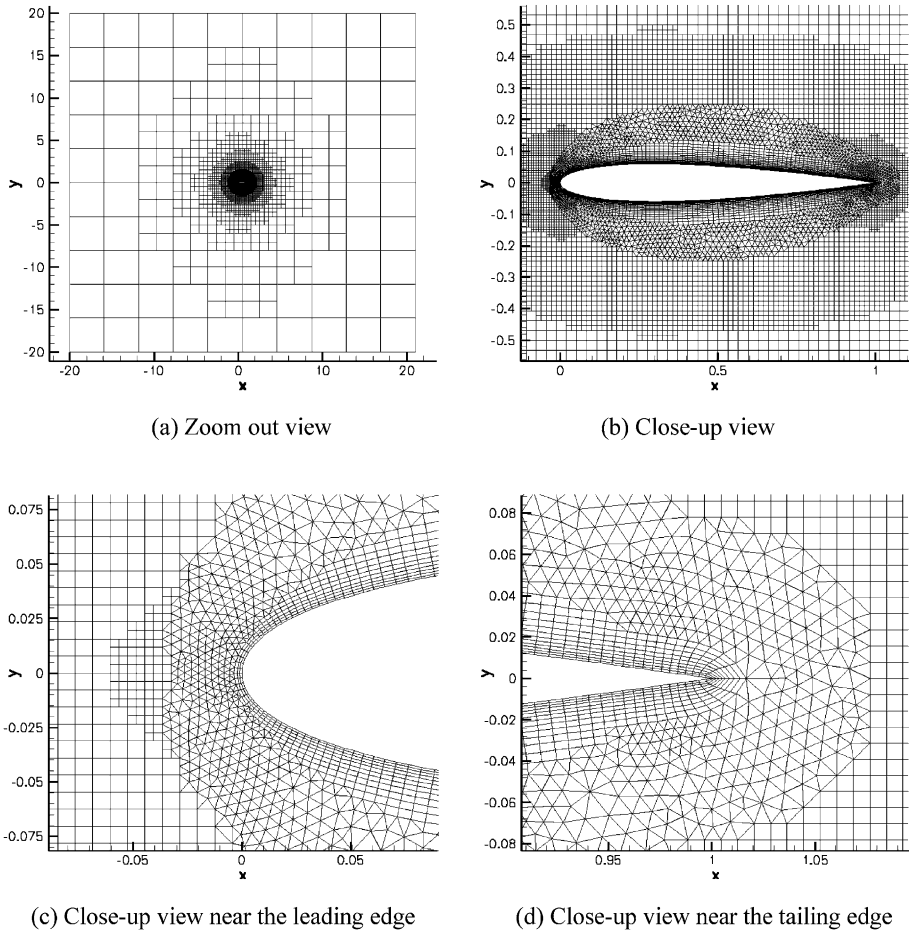


Fig. 6. Hybrid adaptive Cartesian/quad/triangular grids around the NACA0012 airfoil.

experimental lift histories is quite small. In addition, the present computational results agree very well with those in [21,28], although they are not shown here. A grid refinement study was also performed to make sure that the computed solution is grid-independent. In Fig. 9, the lift coefficient histories computed on two different grids are shown. The coarse grid has about 8.5 K cells, while the fine grid has about 16.8 K cells. Note that the solutions on the coarse and fine grids are on top of each other. The difference between the maximum and minimum C_L on the two grids is less than 1%.

To demonstrate the convergence property of the BLU-SGS inner iteration solver, we compare the convergence histories of BLU-SGS with those of the point implicit method and explicit Runge–Kutta method in Fig. 10. Fig. 10(a) shows the convergence in terms of the number of inner-iterations at the first time step (starting from the initial steady-state solution with 16 time steps per cycle), while Fig. 10(b) shows the convergence history in terms of the CPU time at the same time step. Note that the BLU-SGS solver indeed converges much faster (more than an order of magnitude) than the other two methods. Fig. 11(a) and (b) shows the convergence histories at

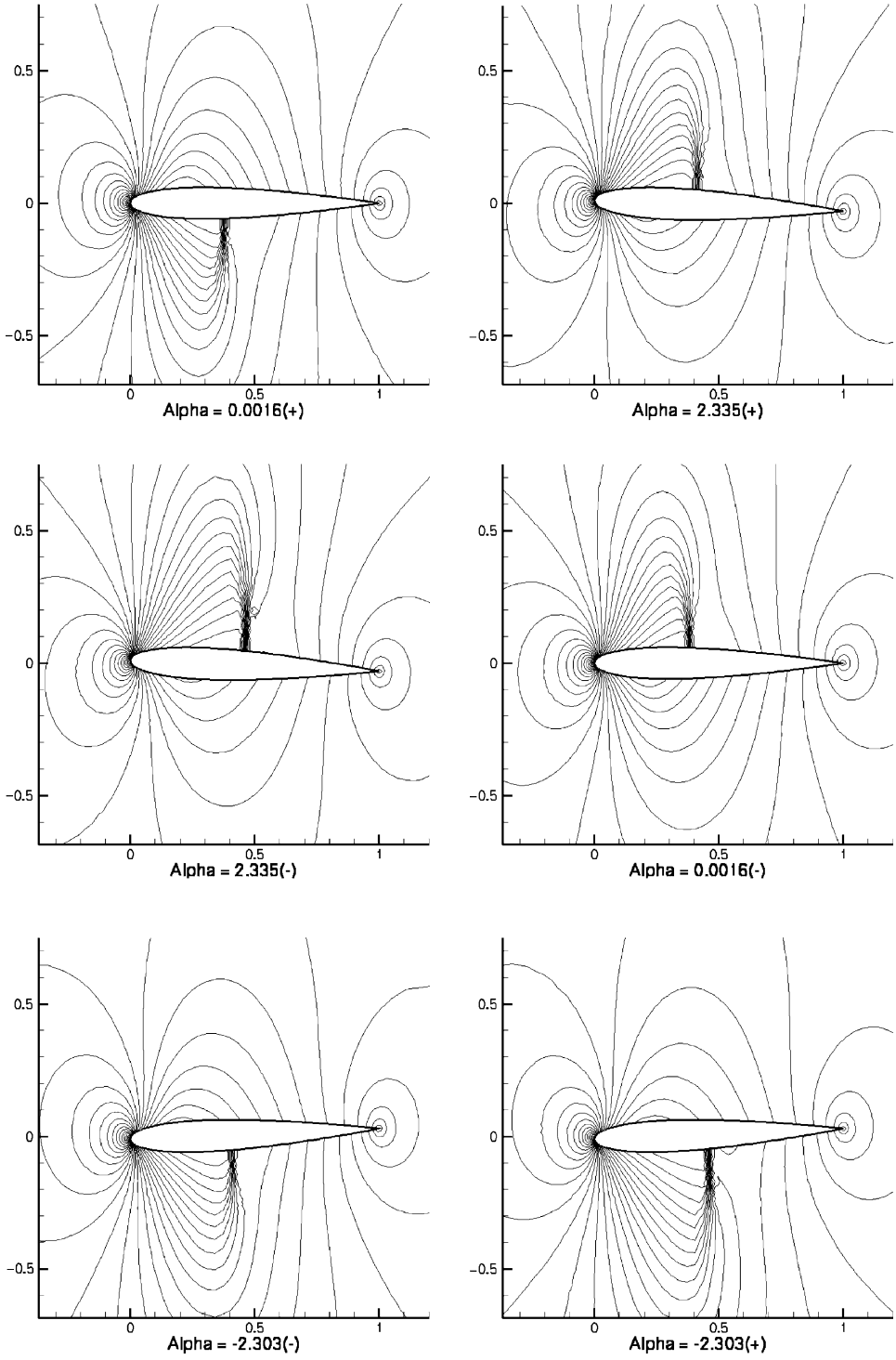


Fig. 7. Pressure contours at six different angles of attack.

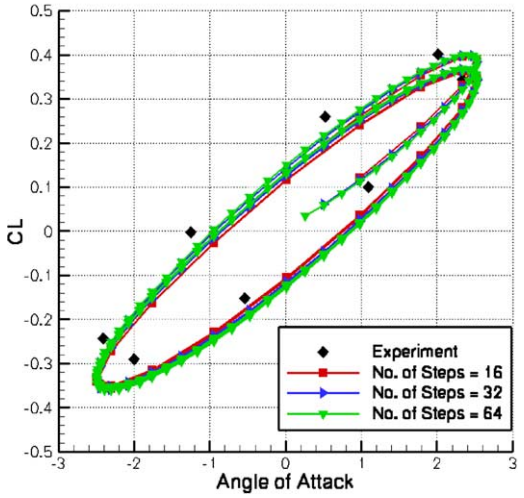


Fig. 8. Lift coefficient vs. angle of attack with different number of time steps per cycle.

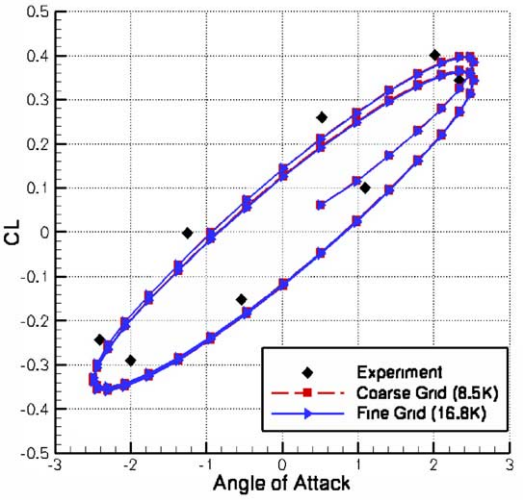


Fig. 9. Lift coefficient vs. angle of attack computed on two different grids.

the last time step (end of the fourth cycle). Once again, the same fast convergence rate is demonstrated by the BLU-SGS solver.

4.3. Turbulent flow over oscillating NACA0012 airfoil

The last test case we considered was a viscous turbulent flow problem. The geometry is the same, but the flow parameters are different than those in the last case. The new flow parameters are: $\alpha_m = 4.86°$, $\Delta\alpha = 2.44°$, $\kappa = 0.0810$, $M_\infty = 0.6$, $Re = 4.8 \times 10^6$. To capture the viscous

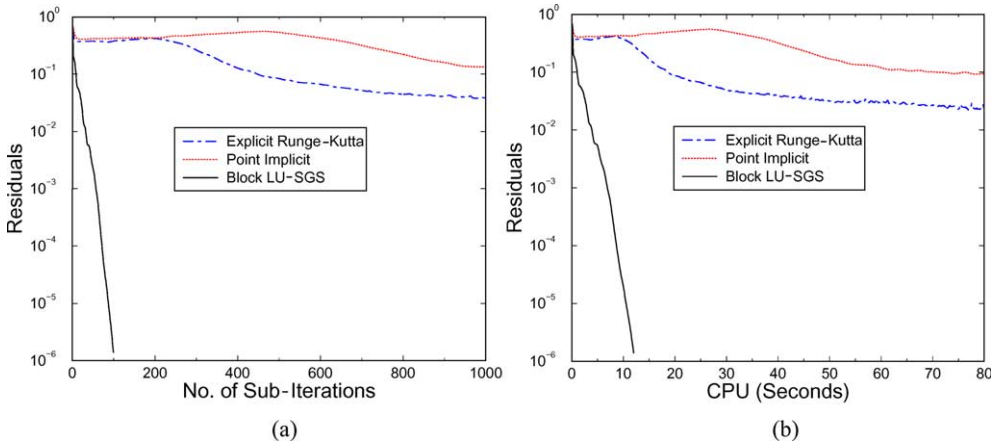


Fig. 10. Convergence histories in terms of number of sub-iterations and CPU time at the first time step.

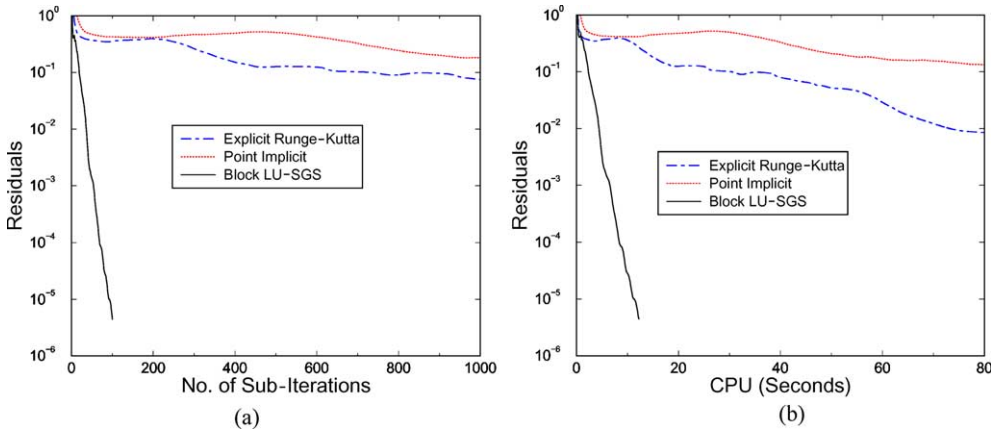


Fig. 11. Convergence histories in terms of number of sub-iterations and CPU time at the last time step.

boundary layer accurately, the quad grid is clustered near the airfoil surface to achieve an average $y+$ of around 25 due to the use of the k-ε model with a wall function. For the solution of the initial steady-state flow field, the hybrid grid has a total of 14,999 grid cells, with 7937 quad, 3254 triangular and 3808 Cartesian cells. The unsteady moving body simulation started from the steady-state solution. Then 50 time steps are used for each cycle to capture the unsteady features. Fig. 12(a) displays the computed Mach number contours at several different angles of attack. For comparison purposes, an inviscid simulation was also carried out with the same flow conditions. The computed Mach contours from this inviscid simulation are shown in Fig. 12(b) for the same angles of attack. Note that the flow fields are quite different near the airfoil, especially at the trailing edge. The turbulent boundary layer and the wake are visible in the viscous solutions. The plot of the lift coefficient vs. the angle of attack is shown in Fig. 13. Results from both the inviscid and viscous simulations are plotted in the figure. It is obvious that the viscous simulation agrees

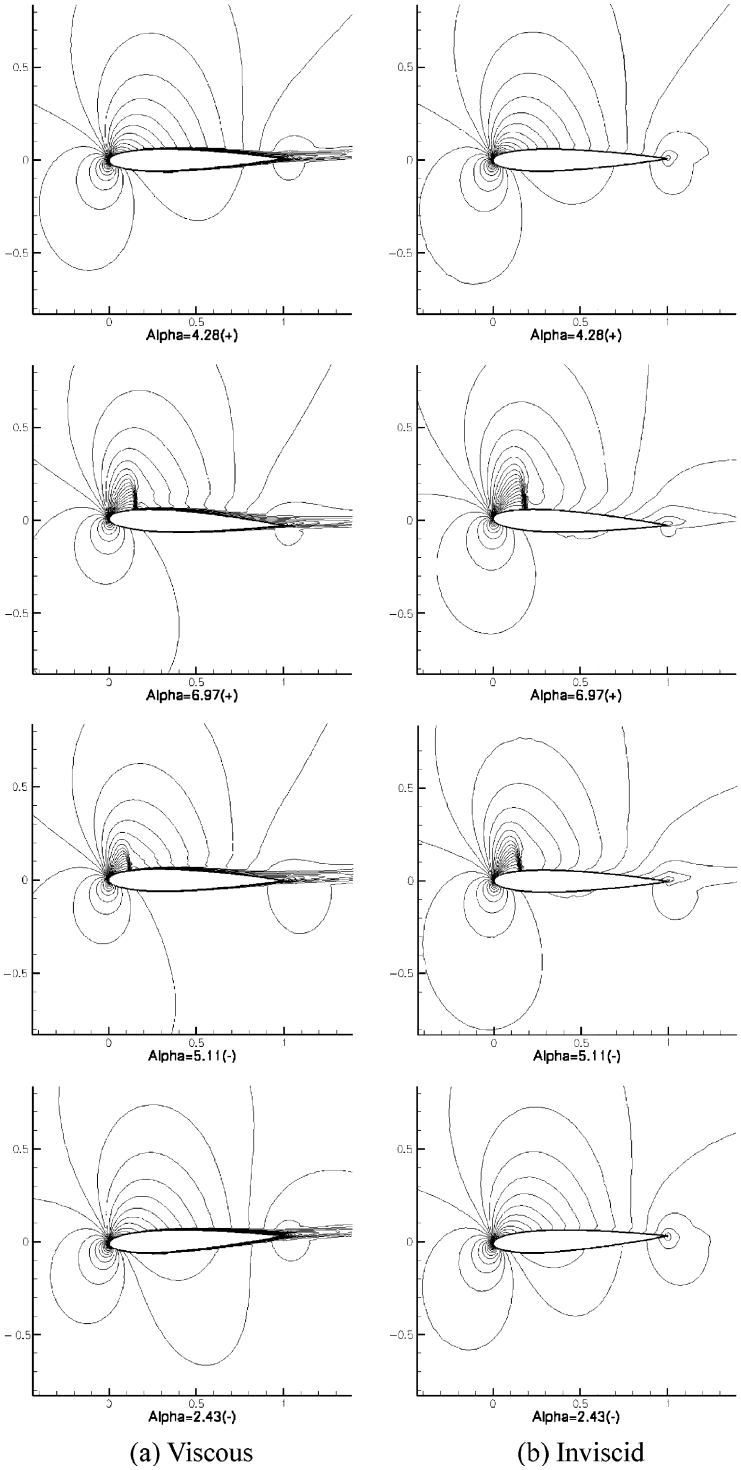


Fig. 12. Computed Mach number contours at different times.

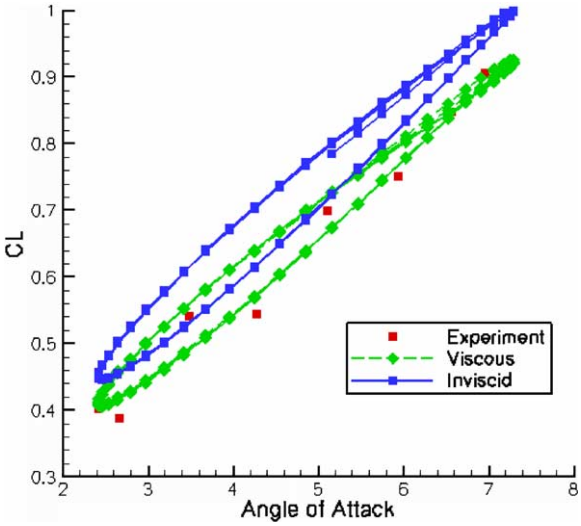


Fig. 13. Lift coefficient vs. angle of attack for both inviscid and viscous simulations.

much better with the experimental data [42] than the inviscid calculation at all angles of attack. Fig. 14 shows the time history of the lift coefficient. This figure demonstrates that the present dual time-stepping approach has a fast convergence rate, and the solution reaches a periodic steady-state within the first two cycles. In Fig. 15, the pressure coefficient distributions along the airfoil surface are displayed. It is clear that the viscous results are slightly better on the lower surface than the inviscid results, and are much better near the trailing edge. Generally speaking, the

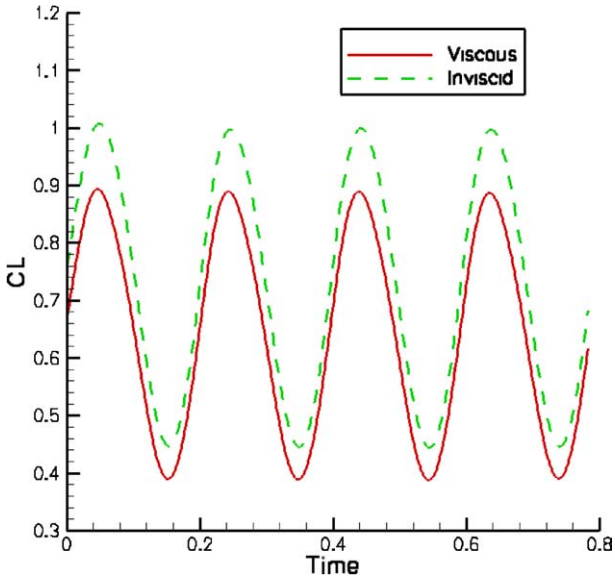


Fig. 14. Computed lift coefficient histories assuming inviscid and viscous flows.

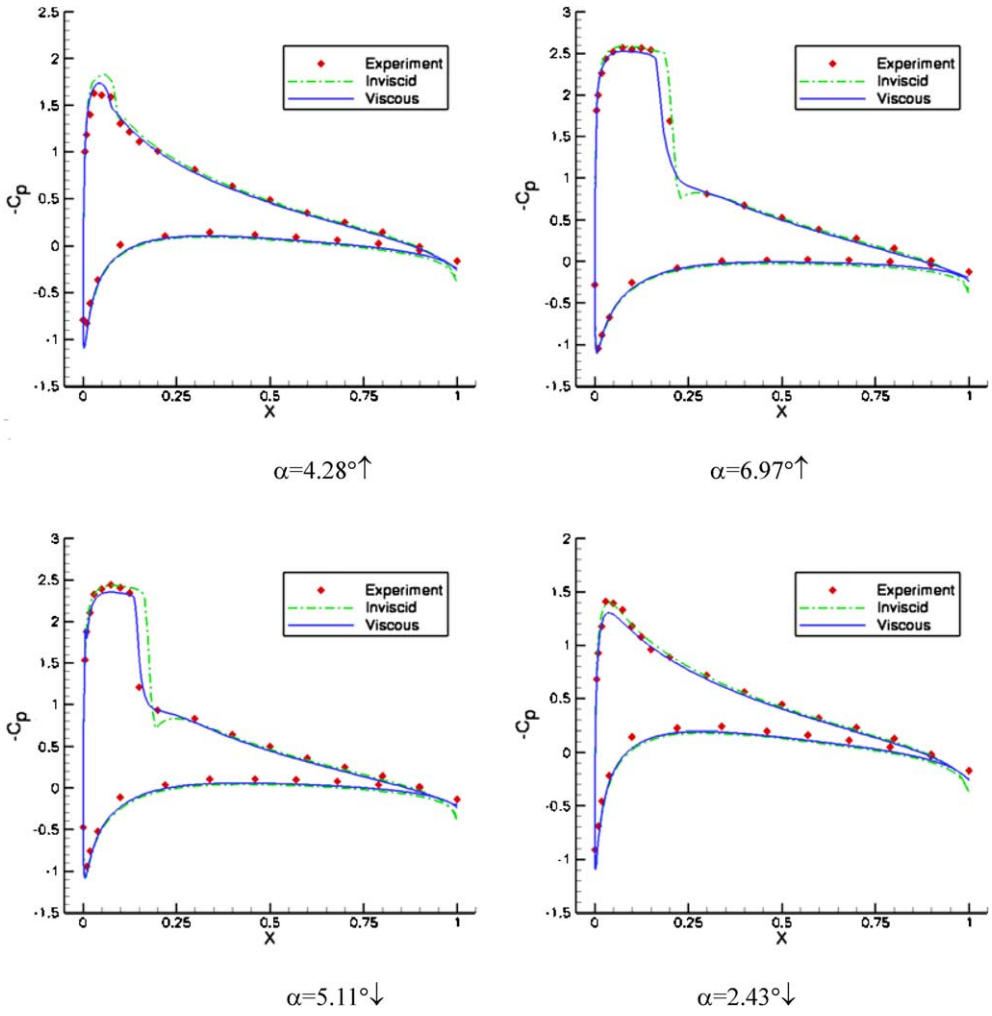


Fig. 15. Pressure coefficient distributions at four typical angles of attack.

viscous simulation produces better C_p distribution than its inviscid counterpart. To show the adequacy of the inner iteration convergence tolerance, Fig. 16 displays the computed density contours with two different convergence tolerances 1.e−3 and 1.e−6 at the last time step. These contours are indistinguishable from each other. Finally the convergence histories are plotted for the viscous simulation in Fig. 17. As expected, the BLU-SGS solver delivers orders of magnitude faster convergence rate than the point implicit or the explicit Runge–Kutta solvers.

In order to verify that the solution is grid-independent, a grid refinement study was also carried out. Two finer grids with about 30 and 55 K cells are generated, and used in the same simulation. Fig. 18(a) shows the plots of lift coefficient vs. the angle of attack computed from the viscous simulations using these three different grids. The minimum and maximum C_L on the three grids are (0.4114, 0.9342), (0.4081, 0.9275), (0.4051, 0.9230), respectively. The differences in C_L between the coarsest and finest meshes are less than 2%. Fig. 18(b) displays the corresponding plots from

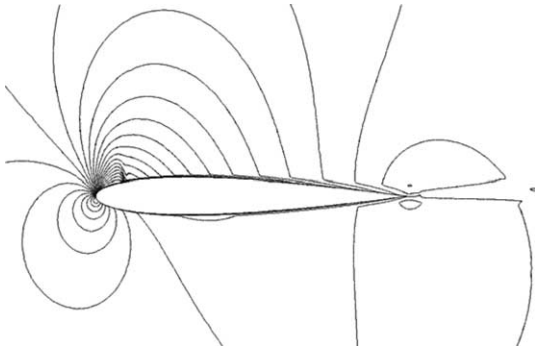


Fig. 16. Density contours computed using two different inner iteration convergence tolerances 1.e−3 and 1.e−6 at the last time step.

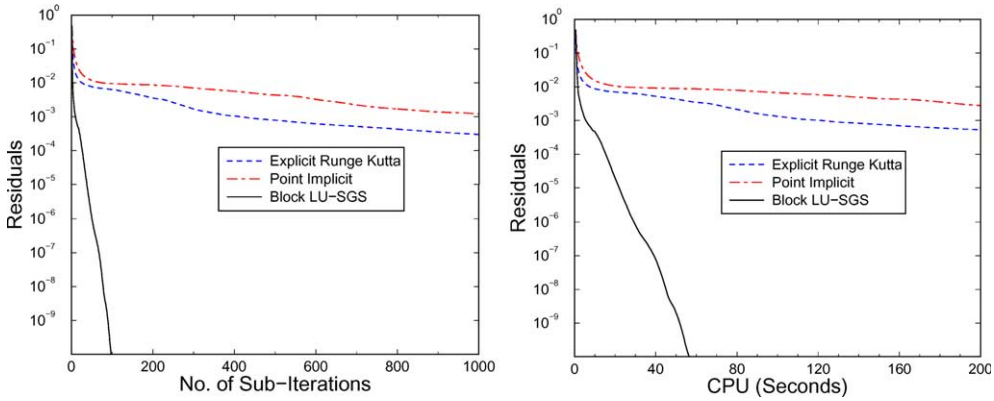
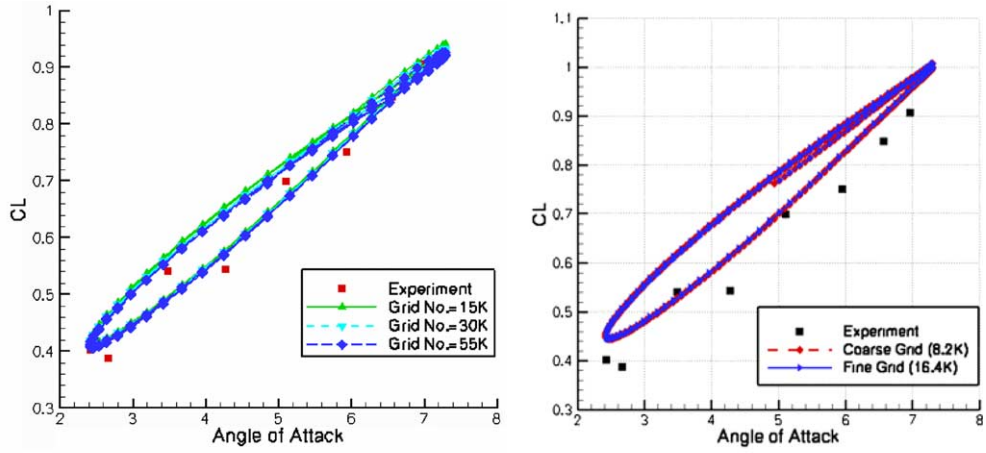


Fig. 17. Convergence histories in terms of number of iterations and CPU time assuming viscous flow.



(a) Viscous (b) Inviscid

Fig. 18. Lift coefficient vs. angle of attack computed on different grids.

the inviscid simulations using two different grids with 8.2 and 16.4 K cells, respectively. Once again, grid-independence is demonstrated for both the inviscid and viscous cases.

The total CPU time used in the viscous turbulent simulation using the BLU-SGS solver with the baseline 15 K grid is estimated to give the readers some rough idea on the required computing resources. The steady-state solution is obtained in about 600 CPU s with a 4 order reduction in the density residual. Each cycle takes 50 time steps. Therefore a total of 200 time steps are required in 4 cycles. Each time step costs about 15 s to complete. Hence a total of 3000 s are needed in the unsteady computation. In the simulation, 200 different computational grids were generated either through spring analogy (180 steps) or local remeshing (20 steps), and the total grid generation and solution interpolation time is on the order of 400 s. Therefore, the total CPU time to carry out a turbulent unsteady simulation is about 4000 s with the 15 K grid. Again these CPU times were measured on a 2.8 GHz Pentium IV machine running Linux.

5. Conclusion

A fast block LU-SGS implicit method has been extended to solve two-dimensional compressible unsteady flows on hybrid dynamic grids. The hybrid grid approach has been shown to be accurate, efficient and capable of handling moving boundaries. The use of the hybrid adaptive Cartesian/quad/triangular grids enables high resolution of viscous boundary layers, and allows high-quality meshes to be generated throughout the computational domain. In addition, only the triangular grids are regenerated when remeshing occurs, making the approach accurate and efficient. Both inviscid and viscous calculations have been presented for flows over moving bodies, including a cylinder, and an oscillating NACA0012 airfoil. The computational results have shown good agreement with experimental data. The block LU-SGS solver demonstrated much higher convergence rate than the point implicit or explicit solvers.

Acknowledgements

The authors acknowledge the startup funding from the Department of Mechanical Engineering, College of Engineering of Michigan State University (MSU), and an AFOSR grant F49620-03-1-0202 monitored by William Hilbun. The first author would like to thank the Department of Education of China and the Delia Koo Grant in MSU for sponsoring his visit to MSU. In addition, we thank Professor Bram van Leer for bringing Ref. [25] to their attention.

References

[1] Jameson A, Baker TJ, Weatherill NP. Calculation of inviscid transonic flow over a complete aircraft. AIAA Paper 86-0103, 1986.

[2] Peraire J, Vahdati M, Morgan K, Zienkiewicz OC. Adaptive remeshing for compressible flow computations. J Comput Phys 1987;72:449–66.

[3] Lohner R, Parikh P. Generation of three-dimensional unstructured grids by the advancing front method. Int J Numer Meth Fluids 1988;8:1135–49.

[4] Anderson WK. A grid generation and flow solution method for the Euler equations on unstructured grids. J Comput Phys 1994;110:23–38.
[5] Venkatakrishnan V. A perspective on unstructured grid flow solvers. AIAA Paper 95-0667, January 1995.
[6] Pirzadeh S. Three-dimensional unstructured viscous grids by the advancing-layers method. AIAA J 1996;34(1):43–9.
[7] Weatherill NP. Unstructured grids: procedures and applications. In: Thompson JF, Soni BK, editors. Handbook of grid generation. CRC Press; 1998 [Chapter 26].
[8] Kallinderis Y, Khawaja A, McMorris H. Hybrid prismatic/tetrahedral grid generation for complex geometries. AIAA J 1996;34:291–8.
[9] Coirier WJ, Jorgenson PCE. A mixed volume grid approach for the Euler and Navier–Stokes equations. AIAA Paper 96-0762, January 1996.
[10] Bayyuk SA, Powell KG, van Leer B. A simulation technique for 2D unsteady inviscid flows around arbitrarily moving and deforming bodies of arbitrarily geometry. AIAA Paper 93-3391-CP, 1993.
[11] Coirier WJ, Powell KG. Solution-adaptive Cartesian cell approach for viscous and inviscid flows. AIAA J 1996;34:938–45.
[12] Aftosmis MJ, Berger MJ, Melton JE. Robust and efficient Cartesian mesh generation for component-based geometry. AIAA Paper No 97-0196, 1997.
[13] Karman SL. SPLITFLOW: a 3D unstructured Cartesian/prismatic grid CFD code for complete geometries. AIAA-95-0343, 1995.
[14] Wang ZJ. A fast nested multi-grid viscous flow solver for adaptive Cartesian/quad grids. Int J Numer Meth Fluids 2000;33(5):657–80.
[15] Wang ZJ, Chen RF. Anisotropic solution-adaptive viscous cartesian grid method for turbulent flow simulation. AIAA J 2002;40:1969–78.
[16] Zhang LP, Zhang HX, Gao SC. A Cartesian/unstructured hybrid grid solver and its applications to 2D/3D complex inviscid flow fields. In: Proceedings of the 7th International Symposium on CFD, Beijing, China, September 1997. p. 347–52.
[17] Zhang LP, Yang YJ, Zhang HX. Numerical simulations of 3D inviscid/viscous flow fields on Cartesian/unstructured/prismatic hybrid grids. In: Proceedings of the 4th Asian CFD Conference, Mianyang, Sichuan, China, September 2000.
[18] Murman S, Aftosmis M, Berger M. Implicit approaches for moving boundaries in a 3-d Cartesian method. AIAA Paper 2003-1119, 2003.
[19] Van Leer B. Towards the ultimate conservative difference scheme. J Comput Phys 1979;32:101.
[20] Vassberg JC. A fast, implicit unstructured-mesh Euler method. AIAA Paper 92-2693, 1992.
[21] Venkatakrishnan V, Mavriplis DJ. Implicit method for the computation of unsteady flows on unstructured grids. AIAA Paper 95-1705, 1995.
[22] Crumpton PI, Giles MB. Implicit time accurate solutions on unstructured dynamic grids. AIAA Paper 95-1671, 1995.
[23] Luo H, Baum JD, Lohner R. A fast, matrix-free implicit method for compressible flows on unstructured grid. J Comput Phys 1998;146:664–90.
[24] Chen RF, Wang ZJ. Fast, block lower–upper symmetric Gauss Seidel scheme for arbitrary grids. AIAA J 2000;38(12):2238–45.
[25] Van Leer B, Mulder WA. Relaxation methods for hyperbolic conservation laws. In: Proceedings of the INRIA Workshop on Numerical Methods for the Euler Equations of Fluid Dynamics, Rocquencourt, France, December 1983.
[26] Jameson A, Caughey DA. How many steps are required to solve the Euler equations of steady compressible flow: in search of a fast solution algorithm. In: 15th AIAA Computational Fluid Dynamics Conference, Anaheim, CA, June 2001.
[27] Batina JT. Unsteady Euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis. AIAA J 1991;29(3):327–33.
[28] Luo H, Baum JD, Lohner R. An accurate, fast, matrix-free implicit method for computing unsteady flows on unstructured grid. Comput Fluids 2001;30:137–59.

[29] Singh KP, Newman JC, Baysal O. Dynamic unstructured method for flows past multiple objects in relative motion. AIAA J 1995;33(4):641–9.
[30] Jameson A. Time-dependent calculations using multi-grid, with applications to unsteady flows past airfoils and wings, AIAA Paper No 91-1596.
[31] Merry MA, Shephard MS. Automatic three-dimensional mesh generation by the modified-octree technique. Int J Numer Meth Eng 1984;20:1965–90.
[32] Lohner R. Some useful data structures for the generation of unstructured grids. Comm Appl Numer Meth 1988;4:123–35.
[33] Parikh P, Pirzadeh S, Lohner R. A package for 3D unstructured grid generation, finite element flow solution and flow field visualization. NACA CR-182090, 1990.
[34] Zhang LP, Guo C, Zhang HX, Gao SC. An unstructured grid generator and its applications for three-dimensional complex geometries. Chinese J Comput Phys 1999;16(5):552–8.
[35] Pirzadeh S. Structured background grids for generation of unstructured grids by advancing front method. AIAA J 1993;31(2):257–64.
[36] Launder BE, Spalding DB. The numerical computation of turbulent flows. Comput Meth Appl Mech Eng 1974;3:269–89.
[37] Roe PL. Approximate Riemann solvers, parameter vectors, and difference schemes. J Comput Phys 1981;43:357–72.
[38] Venkatakrishnan V. Convergence to steady-state solutions of the Euler equations on unstructured grids with limiters. J Comput Phys 1995;118(1):120–30.
[39] Thomas PD, Lombard CK. Geometric conservation law and it's application to flow computations on moving grids. AIAA J 1979;17(10):1030–7.
[40] Wang ZJ, Yang HQ. Unstready flow simulation using a zonal multi-grid approach with moving boundaries. AIAA Paper 94-0057, 1994.
[41] Yoon S, Jameson A. Lower–upper symmetric-Gauss–Seidel method for the Euler and Navier–Stokes equations. AIAA J 1988;26:1025–6.
[42] Landon H. Compendium of unsteady aerodynamic measurements. AGARD Report No 702, 1983.