Journal of Computational Physics **163,** 271–312 (2000)
doi:10.1006/jcph.2000.6575, available online at http://www.idealibrary.com on IDEAL®

A Cell-Centered Adaptive Projection Method for the Incompressible Euler Equations[1]

Daniel F. Martin and Phillip Colella

Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory, Berkeley, California 94720
E-mail: DFMartin@lbl.gov

Received August 24, 1999; revised June 20, 2000

We present an algorithm to compute adaptive solutions for incompressible flows using block-structured local refinement in both space and time. This method uses a projection formulation based on a cell-centered approximate projection, which allows the use of a single set of cell-centered solvers. Because of refinement in time, additional steps are taken to accurately discretize the advection and projection operators at grid refinement boundaries using composite operators which span the coarse and refined grids. This ensures that the method is approximately freestream preserving and satisfies an appropriate form of the divergence constraint. © 2000 Academic Press

Key Words: adaptive mesh refinement; incompressible flow; projection methods.

1. INTRODUCTION

We present a local refinement algorithm for finite-difference solutions to the time-dependent incompressible Euler equations. Our approach is based on that of Berger and Oliger [11] and Berger and Colella [10], in which refined regions are organized into unions of a relatively small number of nested rectangular blocks. Refinement is performed in time as well as in space so that the ratio of the time step to the grid spacing is kept fixed. Calculations are organized around updating the data on the union of rectangles corresponding to a given level of refinement, combined with steps to enforce consistency among the data at different levels.

[1] This research was supported at the Lawrence Berkeley National Laboratory by the U.S. Department of Energy: Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical, Information, and Computing Sciences Division under Contract DE-AC03-76SF00098 and at the University of California at Berkeley by the U.S. Department of Energy Mathematical, Information, and Computing Sciences Division, Grants DE-FG03-94ER25205 and DE-FG03-92ER25140, and the Computational Science Graduate Fellowship Program of the U.S. Department of Energy.

This adaptive mesh refinement (AMR) algorithm has been used extensively to solve a variety of problems in hyperbolic conservation laws and more recently has been extended to incompressible flow in [3]. The incompressible flow case is essentially more difficult for local refinement methods, particularly in the presence of refinement in time. In that case, it is not entirely obvious how to represent the divergence-free constraint. One set of design choices for dealing with this problem was made in [3]; this paper investigates an alternative set of choices for this issue that are simpler, particularly in light of anticipated requirements of representing complex geometry using Cartesian grids.

Our underlying single-grid discretization method is a second-order extension of Chorin's projection method [15] of a type first introduced by Bell, Colella, and Glaz (BCG) [8]. In this approach, a time-centered estimate of the right-hand side of the advective and diffusive terms in the momentum equation is used to compute a provisional update of the velocity. This velocity is then projected onto the space of divergence-free fields. The velocities are co-located at the centers of rectangular control volumes. Advective terms are computed using time-centered velocity fields centered at the edges using a second-order Godunov method, combined with an intermediate staggered-grid projection step [9, 18]. This choice of a single-grid method leads to an adaptive grid method which is second order in space and time as $\Delta t, \Delta x \rightarrow 0$, with $\frac{\Delta t}{\Delta x}$ held fixed.

In extending this algorithm to be adaptive, we addressed the following major design choices.

Choice of Projection Discretization

We use a cell-centered approximate projection similar to that developed independently by Lai and co-workers [23, 24] and Zang *et al.* [36]. This has the advantage of using the same cell-centered elliptic solvers that are used in the intermediate staggered-grid projection step and that are also required for the viscous solvers. Starting from this single-grid algorithm, we apply the projection at two different points in the Berger–Oliger time-stepping procedure. At the end of each time step on a given level of refinement, we apply the projection to the velocity field at that level, using appropriate boundary conditions for the pressure and velocity interpolated from the next coarser level. Second, we apply a multilevel projection similar to that used in [27, 28] to the velocity data at multiple levels of refinement at times when data at two or more levels of refinement have been advanced to the same time. This projection has the effect of imposing the correct matching conditions on the velocity field at the boundaries between different levels of refinement. In the approximate projection formulation of the BCG algorithm, there are four options that have the same truncation error. Either the velocity or the update to the velocity may be projected, and the projection may solve for either the pressure or the update to the pressure over the course of the time step [7, 33]. We have found that, to obtain a robust algorithm, it was necessary to use the project-velocity/solve-for-pressure formulation in the level projection.

Freestream Preservation for Advective Transport

It is often necessary to compute the advective transport of conserved scalars using a conservative finite-difference formulation. In a single-level calculation, advective transport of additional conserved scalars is computed using intermediate edge-centered velocities that are discretely divergence-free, leading to an algorithm that is freestream-preserving. The straightforward application of the algorithm in [10] to maintain conservation in the presence of refinement in time fails to be freestream-preserving in the neighborhood of the boundary between grids at different levels of refinement. The face-centered advection velocities on the coarse and fine grids are computed independently, and there is nothing to guarantee that the composite, time-averaged velocity field is discretely divergence-free at the boundary between coarse and fine grids. Our approach to this problem is based on the volume discrepancy method in [1, 30, 35]. We carry an auxiliary advectively transported quantity that is initialized to be identically equal to a constant. The deviation of this quantity from the constant is an integrated measure of the extent of the violation of the freestream condition. We use this auxiliary quantity to form the right-hand side of a Poisson equation for a potential flow field which, when added to the advection velocity, tends to restore locally constant advected quantities to their freestream state. We find that this procedure reduces the maximum deviation from freestream conditions by one order in the mesh spacing. In addition, the deviation from freestream conditions is negligible away from the cells adjacent to boundaries between levels.

To test these ideas, we present the algorithm in the simplest possible setting: the incompressible Euler equations with an additional conserved scalar. Despite the simplicity of the equations, this is a stringent test for the ideas presented here. The extension to the Navier–Stokes equations will follow in a later paper.

2. AMR NOTATION

Following [10], we perform our adaptive mesh calculations on a hierarchy of nested, cell-centered grids (Fig. 1). At each AMR level $\ell = 0, \ldots, \ell_{\max}$, the problem domain is discretized by a uniform grid Γ^ℓ with grid spacing h_ℓ. Level 0 is the coarsest level, while each level $\ell + 1$ is a factor $n_{\text{ref}}^\ell = h_\ell / h_{\ell+1}$ finer than level ℓ; the refinement ratio n_{ref}^ℓ is an integer. Because refined grids overlie coarser ones, cells on different levels will represent the same geometric region in space. We identify cells at different levels which occupy the same geometric regions by means of the coarsening operator $\mathcal{C}_r(i, j) = (\lfloor \frac{i}{r} \rfloor, \lfloor \frac{j}{r} \rfloor)$. In that

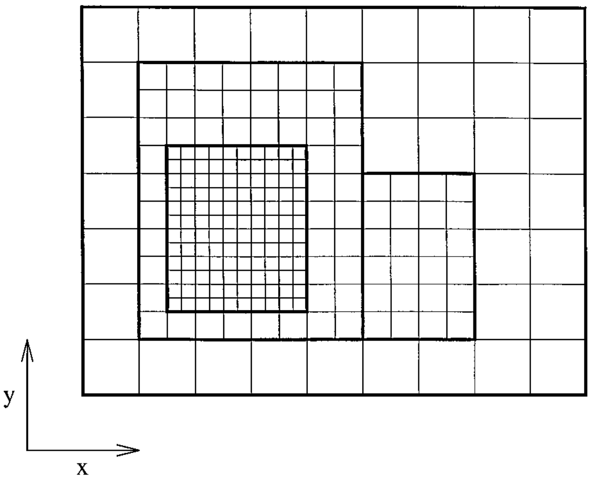


FIG. 1. Block-structured local refinement. Note that refinement is by an integer factor and is organized into rectangular patches.

case, $\{\mathcal{C}_r\}^{-1}\{(i, j)\}$ is the set of all cells in a grid r times finer that represent the same geometric region (in a finite volume sense) as the cell (i, j).

In the present work, we assume that the problem domain is a rectangle and that the refinement ratios are powers of 2. Calculations are performed on a hierarchy of meshes $\Omega^\ell \subset \Gamma^\ell$, with $\Omega^\ell \supset \mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1})$. Ω^ℓ is the union of rectangular patches (grids) with spacing h_ℓ; the block-structured nature of refinement is used in the implementation to simplify computations on the hierarchy of meshes. On the coarsest level, $\Omega^0 = \Gamma^0$. A cell on a level either is completely covered by cells at the next finer level or is not refined at all. Since we assume the solution on finer grids is more accurate, we distinguish between *valid* and *invalid* regions on each level. The valid region on a level is not covered by finer grid cells: $\Omega^\ell_{\text{valid}} = \Omega^\ell - \mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1})$. The grids on each level satisfy a *proper nesting* condition [10]: no cell at level $\ell + 1$ represents a geometric region adjacent to one represented by a valid cell at level $\ell - 1$.

Likewise, $\Omega^{\ell,*}$ denotes the cell edges of level ℓ cells, while $\Omega^{\ell,*}_{\text{valid}}$ refers to the cell edges on level ℓ not covered by level $\ell + 1$ edges. Note that the coarse–fine interface $\partial\Omega^{\ell+1,*}$ between levels ℓ and $\ell + 1$ is considered to be valid on level $\ell + 1$, but not on level ℓ. The coarsening operator also extends to edges: $\mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1,*})$ is the set of level ℓ edges overlain by level $\ell + 1$ edges.

A *composite variable* is defined on the union of valid regions of all levels. Since we organize computation on a level-by-level basis, the invalid regions of each level also contain data, usually an approximation to the valid solution. A *level variable* is defined on the entire level Ω^ℓ (not just the valid region). For a cell-centered variable ϕ, the level variable ϕ^ℓ is defined on all of Ω^ℓ; the composite variable ϕ^{comp} is defined on the union of valid regions over all levels. We also define composite and level-based *vector fields,* which are defined at normal cell edges. Like other edge-centered variables, a composite vector field $\mathbf{u}^{\text{edge,comp}}$ is valid on all edges not overlain by finer edges (Fig. 2). Likewise, we

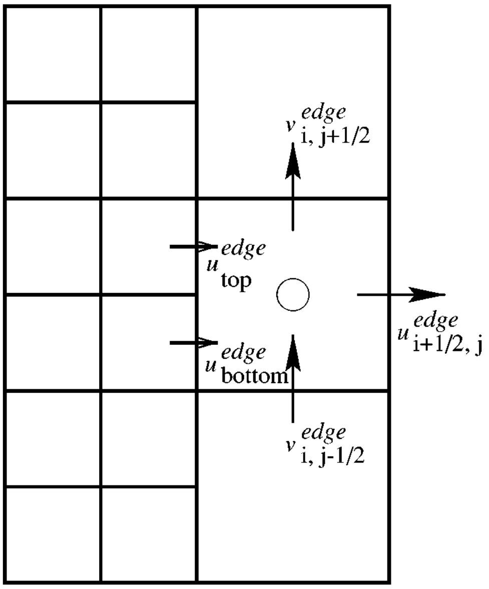


FIG. 2. Sample coarse–fine interface with an edge-centered vector field. Cell (i, j) (open circle) is to the right of the coarse–fine interface.

define composite and level operators which operate on composite and level variables, respectively.

It is also necessary to transfer information from finer grids to coarser ones. We define $\langle\phi^{\ell+1}\rangle$ to be the appropriate cell-centered or edge-centered arithmetic average of level $\ell+1$ data $\phi^{\ell+1}$ to the underlying coarser cells in level ℓ.

Divergence, Flux Registers, and Reflux Divergence

The basic multilevel divergence is a cell-centered divergence of an edge-centered vector field. If none of the edges of cell (i, j) are coarse–fine interfaces, we use a centered-difference divergence:

$$(D^{\text{comp},\ell}\mathbf{u}^{\text{edge}})_{i,j} = \frac{u^{\text{edge}}_{i+1/2,j} - u^{\text{edge}}_{i-1/2,j}}{h_\ell} + \frac{v^{\text{edge}}_{i,j+1/2} - v^{\text{edge}}_{i,j-1/2}}{h_\ell}. \tag{1}$$

On the fine side of a coarse–fine interface, the stencil is unchanged, since the coarse–fine interface with the coarser level $\ell-1$ is a valid edge in level ℓ. For cells on the coarse side of a coarse–fine interface, the coarse-grid vector on the coarse–fine interface is the average of the fine-grid vectors on that edge. For the coarse-grid cell in Fig. 2, the divergence operator is

$$(D^{\text{comp},\ell}\mathbf{u}^{\text{edge}})_{i,j} = \frac{u^{\text{edge},\ell}_{i+1/2,j} - \langle u^{\text{edge},\ell+1}\rangle_{i-1/2,j}}{h_\ell} + \frac{v^{\text{edge},\ell}_{i,j+1/2} - v^{\text{edge},\ell}_{i,j-1/2}}{h_\ell}. \tag{2}$$

The level-operator divergence D^ℓ of a level variable $\mathbf{u}^{\text{edge},\ell}$ is defined by ignoring any finer levels and computing D^ℓ everywhere in Ω^ℓ using (1). Since the composite divergence on level ℓ depends on both level ℓ and level $\ell+1$ data, it may be written as $D^{\text{comp},\ell}(\mathbf{u}^{\text{edge},\ell}, \mathbf{u}^{\text{edge},\ell+1})$; the level operator only depends on level ℓ data: $D^\ell(\mathbf{u}^{\text{edge},\ell})$.

Assume that the vector field $\mathbf{u}^{\text{edge},\ell}$ can be extended to all edges in $\Omega^{\ell,*}$, including those covered by the coarse–fine interface edge $\partial\Omega^{\ell+1,*}$. The composite divergence D^{comp} $\mathbf{u}^{\text{edge,comp}}$ on Ω^ℓ may then be expressed as the level-operator divergence D^ℓ along with a correction for the effects of the finer level $(\ell+1)$. To do this efficiently, we define a *flux register* $\delta\mathbf{u}^{\ell+1}$, defined on $\mathcal{C}_{n^\ell_{\text{ref}}}(\partial\Omega^{\ell+1,*})$, which stores the difference in the edge-centered quantity $\mathbf{u}^{\text{edge}}$ on the coarse–fine interface between levels ℓ and $\ell+1$. Notationally, $\delta\mathbf{u}^{\ell+1}$ belongs to the fine level $(\ell+1)$ because it represents information on $\partial\Omega^{\ell+1,*}$. However, it has coarse-level (ℓ) grid spacing and indexing.

We define the *reflux divergence* D^ℓ_R to be the D^ℓ stencil as applied to the edge-centered vectors on the coarse–fine interface with level $\ell+1$; the general composite operator can then be expressed as

$$(D^{\text{comp},\ell}\mathbf{u}^{\text{edge}})_{i,j} = (D^\ell\mathbf{u}^{\text{edge},\ell})_{i,j} + D^\ell_R(\delta\mathbf{u}^{\ell+1})_{i,j}, \tag{3}$$

$$\delta\mathbf{u}^{\ell+1} = \langle\mathbf{u}^{\text{edge},\ell+1}\rangle - \mathbf{u}^{\text{edge},\ell} \quad \text{on } \mathcal{C}_{n^\ell_{\text{ref}}}(\partial\Omega^{\ell+1,*}). \tag{4}$$

For the level ℓ cell (i, j), D^ℓ_R can be defined as

$$D^\ell_R(\delta\mathbf{u}^{\ell+1})_{i,j} = \frac{1}{h_\ell}\sum_p \pm(\delta\mathbf{u}^\ell)_p, \tag{5}$$

where the sum is over the set of all edges of cell (i, j) which are also coarse–fine interfaces with level $\ell + 1$, and the $\pm$ is $+$ if the edge p is on the high side of cell (i, j) and $-$ if p is on the low side. Note that D_R^ℓ only affects the set of level ℓ cells immediately adjacent to the coarse–fine interface with level $\ell + 1$.

Gradient and Coarse–Fine Interpolation

The gradient is an edge-centered, centered-difference gradient of a cell-centered variable ϕ. The field $G^{\text{comp}}\phi$ is a composite vector field, defined on all valid edges in the multilevel domain. On edges which are not coarse–fine interfaces,

$$\begin{aligned} G^{\text{comp},\ell}(\phi)^x_{i+1/2,j} &= \frac{\phi_{i+1,j} - \phi_{i,j}}{h_\ell} \\ G^{\text{comp},\ell}(\phi)^y_{i,j+1/2} &= \frac{\phi_{i,j+1} - \phi_{i,j}}{h_\ell}. \end{aligned} \tag{6}$$

To compute $G^{\text{comp}}\phi$ at a coarse–fine interface, we interpolate values for ϕ using both coarse- and fine-level values. For the example shown in Fig. 3, to compute the y-component of the gradient across the horizontal coarse–fine interface we first interpolate values into ghost cells around the fine grid (circled X's in Fig. 3); we then use these interpolated values in (6). We use a quadratic interpolation similar to that used in [27] and adopted by [3, 25, 28] to compute ϕ^I. First, a quadratic interpolation is performed parallel to the coarse–fine interface using nearby coarse cells (open circles in Fig. 3) to get values at intermediate points (solid circles). These intermediate values are used along with two fine grid cells (X's) in a second quadratic interpolation normal to the interface to get the appropriate ghost cell values. We denote this quadratic coarse–fine interpolation operator as $I(\phi^\ell, \phi^{\ell-1})$.

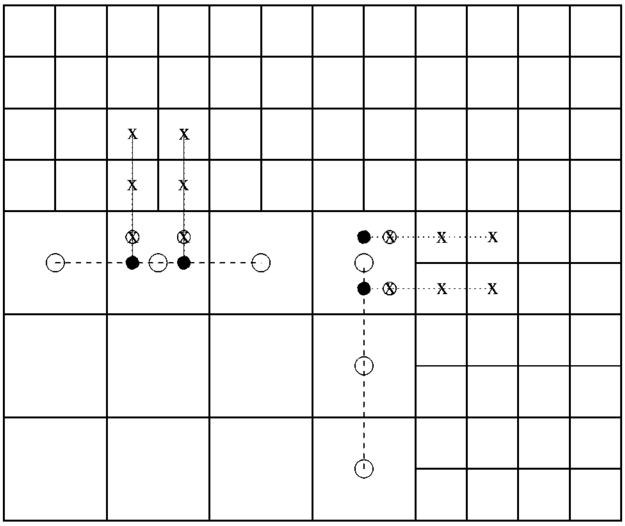


FIG. 3. Interpolation at a coarse–fine interface. Left stencil is the usual stencil. Right stencil is the modified interpolation stencil; since the upper coarse cell is covered by a fine grid, use a shifted coarse grid stencil (open circles) to get intermediate values (solid circles) and then perform final interpolation as before to get "ghost cell" values (circled X's). Note that to perform interpolation for the horizontal coarse–fine interface, we need to shift the coarse stencil left.

Hence

$$\phi^{\ell} = I(\phi^{\ell}, \phi^{\ell-1}) \quad \text{on } \partial\Omega^{\ell} \tag{7}$$

means that ghost cell values for ϕ on level ℓ along the coarse–fine interface with level $\ell - 1$ are computed using this interpolation.

As in [25], if a coarse-grid cell used in the interpolation stencil is covered by a finer grid, we then shift the stencil so that only uncovered coarse-grid cells are used in the interpolation (Fig. 3). This is done for accuracy reasons—we want the flux to be $O(h^2)$ to obtain a formally consistent method. The alternative is to use an averaging procedure which is $O(h^3)$, a process which increases the complication of the algorithm and leads to larger buffer regions between levels, reducing the efficiency of the method. If a suitable coarse-grid quadratic stencil does not exist, we drop the order of interpolation and use whichever coarse cells are available, with a corresponding local loss of accuracy.

The level-operator gradient G^{ℓ} is defined by extending G^{comp} (which is only defined on $\Omega^{\ell,*}_{\text{valid}}$) to all edges in $\Omega^{\ell,*}$. Away from $\partial\Omega^{\ell,*}$ we use the grid-interior stencil (6), while on interfaces with a coarser level $\ell - 1$, we use the interpolation operator $I(\phi^{\ell}, \phi^{\ell-1})$ to compute ghost cell values to be used in (6).

The composite gradient on level ℓ, $G^{\text{comp},\ell}$, is dependent on level ℓ and coarse-level $(\ell - 1)$ data: $G^{\text{comp},\ell}(\phi^{\ell}, \phi^{\ell-1})$. Likewise, the level-operator gradient can be written $G^{\ell}(\phi^{\ell}, \phi^{\ell-1})$.

Laplacian

The Laplacian is defined as the divergence of the gradient:

$$L^{\text{comp}}\phi^{\text{comp}} = D^{\text{comp}}G^{\text{comp}}\phi^{\text{comp}} \tag{8}$$

$$L^{\ell}\phi^{\ell} = D^{\ell}G^{\ell}\phi^{\ell}. \tag{9}$$

On the interiors of grids, (8) and (9) reduce to the normal five-point second-order discrete Laplacian. On the fine side of a coarse–fine interface, the interpolation operator I fills ghost-cell values which are used in the five-point stencil. On the coarse side of a coarse–fine interface, (8) becomes

$$L^{\text{comp},\ell}\phi = L^{\ell}\phi^{\ell} + D^{\ell}_{R}(\delta G\phi^{\ell+1}) \tag{10}$$

$$\delta G\phi^{\ell+1} = \langle G^{\ell+1}\phi\rangle - G^{\ell}\phi^{\ell}. \tag{11}$$

On grid interiors, L^{comp} has a truncation error of $O(h^2)$ owing to cancellation of error terms in the centered-difference stencil. At coarse–fine interfaces, this drops to $O(h)$ owing to division of the $O(h^3)$ interpolant by h^2 and the loss of centered-difference cancellations. However, if the discrete equation $L^{\text{comp}}\phi = \rho$ is solved using these operators, the resulting solution ϕ is second-order accurate, because this loss of accuracy occurs on a set of codimension one [22]. The dependencies of the Laplacian operators may again be expressed explicitly: $L^{\text{comp},\ell}(\phi^{\ell}, \phi^{\ell+1}, \phi^{\ell-1})$ and $L^{\ell}(\phi^{\ell}, \phi^{\ell-1})$.

3. ADAPTIVE ALGORITHM DESCRIPTION

3.1. Formulation of the Problem

The simplest example of inviscid incompressible fluid flows is the incompressible constant-density Euler equations

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p, \tag{12}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{13}$$

$$\mathbf{u} \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega, \tag{14}$$

where $\mathbf{u}(\mathbf{x}, t)$ is the fluid velocity vector $(u, v)^T$, t is the time, and $p(\mathbf{x}, t)$ is the pressure. The evolution equation for a passively transported scalar $s(\mathbf{x}, t)$ in incompressible flow is

$$\frac{\partial s}{\partial t} + \nabla \cdot (\mathbf{u}s) = 0. \tag{15}$$

We transform the constrained dynamics problem of Eqs. (12) and (13) into an initial value problem through the use of the Hodge–Helmholtz decomposition. An arbitrary vector field $\mathbf{w}$ can be uniquely decomposed into two orthogonal components, one of which is divergence-free, with the other being the gradient of a scalar:

$$\mathbf{w} = \mathbf{w}_d + \nabla\phi \tag{16}$$

$$\nabla \cdot \mathbf{w}_d = 0 \tag{17}$$

$$\Delta\phi = \nabla \cdot \mathbf{w} \quad \text{on } \Omega \tag{18}$$

$$\mathbf{w}_d \cdot \mathbf{n} = 0, \quad \frac{\partial \phi}{\partial \mathbf{n}} = \mathbf{w} \cdot \mathbf{n} \quad \text{on } \partial\Omega \tag{19}$$

$$\int_\Omega \mathbf{w}_d \cdot \nabla\phi \, d\mathbf{x} = 0. \tag{20}$$

This decomposition can be expressed in terms of an orthogonal projection $\mathbf{P}$: $\mathbf{P}(\mathbf{w}) = \mathbf{w}_d$, computed by solving (18) for $\nabla\phi$ and subtracting to obtain the divergence-free part. Formally,

$$\mathbf{P} = I - \nabla(\Delta)^{-1}\nabla\cdot \tag{21}$$

Using the projection operator, we can transform the constrained system (12), (13) into a pure evolution equation, with the constraint applied to the initial data:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}(-\mathbf{u} \cdot \nabla\mathbf{u}) \tag{22}$$

$$(\nabla \cdot \mathbf{u})(\cdot, t = 0) = 0. \tag{23}$$

Chorin [15] used this formulation as the starting point for a discretization of the incompressible flow equations. Following [8], we use for our algorithm a predictor–corrector formulation in which we first compute an intermediate velocity field and project it onto the space of vectors which satisfy the divergence constraint. Updates to the scalar s are computed using a conservative update.

On a uniform grid with grid spacing h, the velocity $\mathbf{u}(\mathbf{x}, t)$ is approximated by $\mathbf{u}_{i,j}(t) \approx \mathbf{u}(ih, jh, t)$. The scalar field $s(\mathbf{x}, t)$ is likewise approximated as $s_{i,j}(t) \approx s(ih, jh, t)$. Then,

$$\mathbf{u}(t + \Delta t) = \mathbf{P}(\mathbf{u}(t) - \Delta t(\mathbf{u} \cdot \nabla\mathbf{u})^H) \tag{24}$$

$$\nabla p^H = \frac{1}{\Delta t}(\mathbf{I} - \mathbf{P})(\mathbf{u}(t) - \Delta t(\mathbf{u} \cdot \nabla\mathbf{u})^H) \tag{25}$$

$$s(t + \Delta t) = s(t) - \Delta t\nabla \cdot (\mathbf{u}s)^H, \tag{26}$$

where the superscript H indicates centering at the intermediate time $(t + \frac{1}{2}\Delta t)$. Following [16], we compute $(\mathbf{u} \cdot \nabla\mathbf{u})^H$ and $(\nabla \cdot (\mathbf{u}s))^H$ using a second-order upwind method. Details of the advection scheme can be found in the Appendix. Equation (24) can also be expressed in terms of the pressure gradient $(\nabla p^H)_{i,j} \approx \nabla p(ih, jh, t + \frac{\Delta t}{2})$, where ∇p^H is computed using (25):

$$\mathbf{u}(t + \Delta t) = \mathbf{u}(t) - \Delta t(\mathbf{u} \cdot \nabla\mathbf{u})^H - \Delta t(\nabla p)^H. \tag{27}$$

We extend the algorithm developed by Berger and Colella [10] for hyperbolic conservation laws to the incompressible Euler equations. The algorithm in [10] refined in time as well as space and maintained conservation at coarse–fine interfaces. This algorithm can be extended to general first-order hyperbolic PDEs such as the momentum equation for incompressible flow. For context, we first outline the hyperbolic algorithm before describing its extension to incompressible flow.

3.2. Hyperbolic Algorithm

In [10], hyperbolic conservation laws of the form

$$u_t + f(u)_x + g(u)_y = 0 \tag{28}$$

are solved using local refinement in both time and space.

The multilevel hyperbolic algorithm can be generalized for any number of levels by defining it as a series of recursive single-level advances. The pseudocode in Fig. 4 advances the level ℓ discrete solution U^ℓ from time t^ℓ to $t^\ell + \Delta t^\ell$. Because this function is recursive, all finer levels (initially also at t^ℓ) are also advanced to the new time. The entire solution is advanced from time t^0 to $t^0 + \Delta t^0$ by advancing the coarsest level, which advances the composite solution through a series of recursive level advances.

In the initial update to U^ℓ, an explicit conservative finite-difference method is used. $\mathbf{F} = (F, G)^T$ is the numerical approximation to the flux function $\mathbf{f} = (f, g)^T$. Data needed for the stencil of this method which lie outside Ω^ℓ are interpolated in time and space using level $\ell - 1$ data. The recursive finer-level advances are subcycled, with $\Delta t^\ell / h^\ell = \Delta t^{\ell+1} / h^{\ell+1}$. Discussions of the benefits of subcycling can be found in [3, 10, 11]. The time steps $\{\Delta t^\ell\}$ can only be changed at the end of the level 0 time step.

Once the fine level $(\ell + 1)$ has been advanced to time $t^\ell + \Delta t^\ell$, levels ℓ and $\ell + 1$ are *synchronized*. Synchronization forces the update of the multilevel solution to be in discrete conservation form by replacing the flux across $\partial\Omega^{\ell+1,*}$ into the valid level ℓ cells with the sum of the fine-level fluxes. Since these coarse cells have already been updated in the initial update, this is achieved in the synchronization step by a reflux divergence of the difference between the coarse flux and the sum of the fine-level fluxes. Also, the solution $U^\ell(t^\ell + \Delta t^\ell)$ on invalid regions of level ℓ is replaced with the averaged finer solution $\langle U^{\ell+1}(t^\ell + \Delta t^\ell)\rangle$.

```
LevelAdvance($\ell$, $t^\ell$, $\Delta t^\ell$)
    Initial update of $U^\ell$:
        $U^\ell(t^\ell + \Delta t^\ell)_{i,j} = U^\ell(t^\ell)_{i,j} - \Delta t^\ell D^\ell(\mathbf{F}^\ell)$
    Initialize/update flux registers:
        if ($\ell < \ell_{\max}$) $\delta\mathbf{F}^{\ell+1} = -\mathbf{F}^\ell \cdot \mathbf{n}^{\ell+1}_{\mathrm{CF}}$   on $\mathcal{C}_{n^\ell_{\mathrm{ref}}}(\partial\Omega^{\ell+1,*})$
        if ($\ell > 0$) $\delta\mathbf{F}^\ell := \delta\mathbf{F}^\ell + \frac{1}{n^{\ell-1}_{\mathrm{ref}}}\left\langle \mathbf{F}^\ell \cdot \mathbf{n}^\ell_{\mathrm{CF}} \right\rangle$   on $\mathcal{C}_{n^{\ell-1}_{\mathrm{ref}}}(\partial\Omega^{\ell,*})$
    Recursive calls to LevelAdvance for next finer level:
        if ($\ell < \ell_{\max}$) then
            for $n = 0, n^\ell_{\mathrm{ref}} - 1$
                $\Delta t^{\ell+1} = \frac{1}{n^\ell_{\mathrm{ref}}}\Delta t^\ell$
                $t^{\ell+1} = t^\ell + n\Delta t^{\ell+1}$
                LevelAdvance($\ell + 1$, $t^{\ell+1}$, $\Delta t^{\ell+1}$)
            end for
            Synchronize level $\ell$ with level $\ell + 1$:
                $U^\ell(t^\ell + \Delta t^\ell) = \langle U^{\ell+1}(t^\ell + \Delta t^\ell)\rangle$   on   $\mathcal{C}_{n^\ell_{\mathrm{ref}}}(\Omega^{\ell+1})$.
                $U^\ell(t^\ell + \Delta t^\ell) := U^\ell(t^\ell + \Delta t^\ell) - \Delta t D^\ell_R(\delta\mathbf{F}^{\ell+1})$
        end if
end LevelAdvance
```

FIG. 4. Pseudocode for recursive timestep used for hyperbolic conservation laws.

3.3. Extension to the Incompressible Euler Equations

In the incompressible algorithm, we maintain the essential structure of the hyperbolic algorithm. Once again, the algorithm is structured as a series of recursive updates on a level. The basic steps when updating level ℓ are:

1. Perform single-level update on level ℓ, including application of a level ℓ projection to the velocities to ensure that they are approximately divergence-free (π^ℓ is the approximation to the pressure computed using the level projection),
2. Initialize/update flux registers with coarse–fine interface information.
3. Make recursive calls to update finer levels n_{ref} times with $\Delta t^{\ell+1} = \Delta t^\ell / n^\ell_{\mathrm{ref}}$.
4. Synchronize composite multilevel solution.

The single-level update is based on the single-grid projection algorithm outlined in the Appendix. In addition to conservation, additional issues are raised by solving the equations of incompressible flow on a multilevel hierarchy of grids.

To illustrate these issues, consider a straightforward extension of the hyperbolic algorithm to a projection method for incompressible flow, illustrated in pseudocode form in Fig. 5. (Here u^{half} and s^{half} are staggered-grid approximations to u and s at the intermediate time $(t^\ell + \frac{1}{2}\Delta t^\ell)$, computed as described in the Appendix.) In this algorithm, the velocity field is updated using a projection based on cell-centered level operators, described below. Even though the momentum equation is not in conservative form, a refluxing correction is applied at the coarse–fine interface, as was done in [3]. This results in a consistent and stable set of boundary conditions for the velocity advection operator.

```
EulerAdvance($\ell$, $t^\ell$, $\Delta t^\ell$)
    Initial update of $s^\ell$, $\mathbf{u}^\ell$:
        $s^\ell(t^\ell + \Delta t^\ell) = s^\ell(t^\ell) - \Delta t^\ell D^\ell(\mathbf{u}^{\text{half},\ell} s^{\text{half},\ell})$
        $\mathbf{u}^\ell(t^\ell + \Delta t^\ell) = \mathbf{P}^\ell(\mathbf{u}^\ell(t^\ell) - \Delta t^\ell(\mathbf{u}^{\text{half},\ell} \cdot \nabla \mathbf{u}^{\text{half},\ell}))$
        $\pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell) = (\mathbf{I} - \mathbf{P}^\ell)(\mathbf{u}^\ell(t^\ell) - \Delta t^\ell(\mathbf{u}^{\text{half},\ell} \cdot \nabla \mathbf{u}^{\text{half},\ell}))$
    Initialize/update flux registers:
        **if** $(\ell < \ell_{\max})$
            $\delta s^{\ell+1} = -s^{\text{half},\ell}\mathbf{u}^{\text{half},\ell} \cdot \mathbf{n}_{\text{CF}}^{\ell+1}$
            $\delta \mathbf{V}^{\ell+1} = -\mathbf{u}^{\text{half},\ell}\mathbf{u}^{\text{half},\ell} \cdot \mathbf{n}_{\text{CF}}^{\ell+1}$
        **if** $(\ell > 0)$
            $\delta s^\ell := \delta s^\ell + \langle s^{\text{half},\ell}\mathbf{u}^{\text{half},\ell}\rangle \cdot \mathbf{n}_{\text{CF}}^\ell$
            $\delta \mathbf{V}^\ell := \delta \mathbf{V}^\ell + \langle \mathbf{u}^{\text{half},\ell}\mathbf{u}^{\text{half},\ell}\rangle \cdot \mathbf{n}_{\text{CF}}^\ell$
    Recursive calls for finer level:
        **if** $(\ell < \ell_{\max})$ **then**
            **for** $n = 0, n_{\text{ref}}^\ell - 1$
                $\Delta t^{\ell+1} = \frac{1}{n_{\text{ref}}^\ell}\Delta t^\ell$
                $t^{\ell+1} = t^\ell + n\Delta t^{\ell+1}$
                EulerAdvance($\ell + 1$, $t^{\ell+1}$, $\Delta t^{\ell+1}$)
            **end for**
            Synchronization:
                $s^\ell(t^\ell + \Delta t^\ell) := s^\ell(t^\ell + \Delta t^\ell) - \Delta t^\ell D_R^\ell(\delta s^{\ell+1})$
                $\mathbf{u}^\ell(t^\ell + \Delta t^\ell) := \mathbf{u}^\ell(t^\ell + \Delta t^\ell) - \Delta t^\ell D_R^\ell(\delta \mathbf{V}^{\ell+1})$
        **end if**
**end** EulerAdvance
```

FIG. 5. Straightforward extension of hyperbolic algorithm to incompressible flow.

This algorithm suffers from two deficiencies, both of which were identified in [3].

1. While the level projection $\mathbf{P}^\ell$ ensures that the velocity field satisfies $D^\ell \mathbf{u}^\ell = 0$, there is nothing to guarantee that $(\mathbf{u}^{\ell,\text{valid}} - \langle \mathbf{u}^{\ell+1,\text{valid}}\rangle) \cdot \mathbf{n}_{\text{CF}}^{\ell+1} = 0$ to the appropriate order of accuracy along the coarse–fine interface $\partial\Omega^{\ell+1,*}$. This is equivalent to the elliptic matching condition described in [25, 26]. To maintain accuracy, solutions to elliptic equations must satisfy both Dirichlet *and* Neumann matching conditions across coarse–fine interfaces. The pressure field computed in this algorithm only satisfies a Dirichlet matching condition across coarse–fine interfaces. Also, we increment the coarse-level velocity field in the velocity-refluxing step, but we do not guarantee that this increment is divergence-free.

2. Refluxing is not freestream preserving: initially constant scalar fields may not remain constant at coarse–fine interfaces. Consider a scalar field which has a constant value s_0. In this case, the level ℓ fluxes are equal, and the single-level update produces the correct updated value s_0. For freestream preservation, the flux correction in the synchronization

step should be zero. For scalar advection we have

$$\begin{aligned}\delta s^{\ell+1} &= (\langle \mathbf{F}^{s,\ell+1}\rangle - \mathbf{F}^{s,\ell}) \cdot \mathbf{n}_{\mathrm{CF}}^{\ell+1} \\ &= (\langle \mathbf{u}^{\mathrm{half},\ell+1} s^{\ell+1}\rangle - \mathbf{u}^{\mathrm{half},\ell} s^{\ell}) \cdot \mathbf{n}_{\mathrm{CF}}^{\ell+1} \\ &= (\langle \mathbf{u}^{\mathrm{half},\ell+1}\rangle - \mathbf{u}^{\mathrm{half},\ell}) s_0 \cdot \mathbf{n}_{\mathrm{CF}}^{\ell+1}. \end{aligned} \tag{29}$$

To preserve the uniform solution, the level ℓ and averaged $(\ell+1)$ advection velocities $\mathbf{u}^{\mathrm{half}}$ would have to be equal along the coarse–fine interface. However, $\mathbf{u}^{\mathrm{half},\ell}$ and $\mathbf{u}^{\mathrm{half},\ell+1}$ are computed independently on each level, with nonlocal dependence on the data via a solution to an elliptic pressure equation, so there is no reason to expect this to be the case. As a result, errors in advection occur at coarse–fine interfaces; these errors are then advected through the flow. The essential problem is that incompressibility of the advection velocities is enforced using single-level operators rather than composite operators, so the advection velocities are not divergence-free across coarse–fine interfaces. While it would be possible to implement a composite multilevel projection for the advection velocities at all subcycled time steps (as done by Hornung and Trangenstein [20] in the context of porous media flow), this would substantially increase the computational expense of the method and would reduce the benefits of refinement in time.

In the following, we discuss the cell-centered discretization and its extension to multiple levels used to impose an appropriate form of the divergence constraint on multilevel velocity data. We also describe the volume discrepancy method used to maintain freestream preservation, at least approximately.

3.4. Composite Projection

The multilevel divergence, gradient, and Laplacian operators defined in Section 2 correspond to an idempotent projection based on a staggered-grid velocity field, similar to those used in the composite staggered-grid projection in [28]. However, velocity and pressure in this work are cell-centered; so a cell-centered projection discretization is required.

In earlier versions of the projection method [8, 9, 15], $\mathbf{P} = I - G(DG)^{-1}D$, where D and G are difference approximations to the divergence and gradient, with $D = -G^T$. This discretization of the projection is idempotent ($\mathbf{P}^2 = \mathbf{P}$); however, decoupling of the stencils causes badly behaved linear algebra, which greatly increases the difficulty of implementing local refinement [21]. It was first proposed in [2] to deal with this problem by the introduction of a stable but nonidempotent discretization of the projection with well-behaved numerical linear algebra. The starting point for the discretization of $\mathbf{P}$ used in this work is the approximate projection developed by Lai [24], which uses only cell-centered solvers:

$$\mathbf{P} = I - G^{\mathrm{CC}}(L^{-1})D^{\mathrm{CC}}. \tag{30}$$

Here D^{CC} and G^{CC} are cell-centered centered-difference approximations to the divergence and gradient operators, and L is the standard five-point discrete Laplacian operator.

The cell-centered divergence and gradient operators are constructed from staggered-grid operators through the use of appropriate cell-to-edge and edge-to-cell averaging. If $\mathbf{u}^{\mathrm{edge}}$ is a staggered-grid vector field, defined on $\Omega^{\ell,*}$, then $Av^{\mathrm{E}\to\mathrm{C}}(\mathbf{u}^{\mathrm{edge}})$ is

$$(Av^{\mathrm{E}\to\mathrm{C}}\mathbf{u}^{\mathrm{edge}})_{i,j} = \left(\frac{u^{\mathrm{edge}}_{i+1/2,j} + u^{\mathrm{edge}}_{i-1/2,j}}{2}, \frac{v^{\mathrm{edge}}_{i,j+1/2} + v^{edge}_{i,j-1/2}}{2} \right)^T. \tag{31}$$

If $\mathbf{u}$ is a cell-centered vector field defined on Ω^ℓ, then

$$\begin{aligned}(Av^{\mathrm{C}\to\mathrm{E}}u)_{i+1/2,j} &= \frac{u_{i+1,j}+u_{i,j}}{2} \quad \text{on } \Omega^{\ell,*} - \partial\Omega^{\ell,*}\\ (Av^{\mathrm{C}\to\mathrm{E}}v)_{i,j+1/2} &= \frac{v_{i,j+1}+v_{i,j}}{2}.\end{aligned} \tag{32}$$

On $\partial\Omega^{\ell,*}$, $(Av^{\mathrm{C}\to\mathrm{E}}\mathbf{u})$ is obtained by second-order extrapolation, including solid wall physical boundaries, but not including periodic boundaries—the latter are computed using the interior formulas (32) and ghost cells.

We can use these operators to define cell-centered divergence and gradient operators on Ω^ℓ:

$$D^{\mathrm{CC},\ell}\mathbf{u}^{\mathrm{CC}} = D^\ell Av^{\mathrm{C}\to\mathrm{E}}\mathbf{u}^{\mathrm{CC},\ell}, \tag{33}$$

$$G^{\mathrm{CC},\ell}\phi^\ell = Av^{\mathrm{E}\to\mathrm{C}}G^\ell\phi^\ell. \tag{34}$$

The composite operators $D^{\mathrm{CC,comp}}$ and $G^{\mathrm{CC,comp}}$ are defined similarly:

$$D^{\mathrm{CC,comp}} = D^{\mathrm{comp}}Av^{\mathrm{C}\to\mathrm{E}}\mathbf{u}^{\mathrm{CC,comp}}, \tag{35}$$

$$G^{\mathrm{CC,comp}}\phi^{\mathrm{comp}} = Av^{\mathrm{E}\to\mathrm{C}}G^{\mathrm{comp}}\phi^{\mathrm{comp}}. \tag{36}$$

Typically, the gradient field being computed in the projection is discontinuous at coarse–fine interfaces, since it is correcting the mismatch in the normal component of the velocity across the interface. For the correction field shown in Fig. 6, computing staggered-grid gradients using G^{comp} and then averaging to produce a cell-centered gradient washes out the structure of the gradient field near the interface because the positive and negative gradients cancel. For this reason, the derivative normal to the interface is computed using one-sided differencing from the coarse side of the coarse–fine interface; this one-sided gradient is used

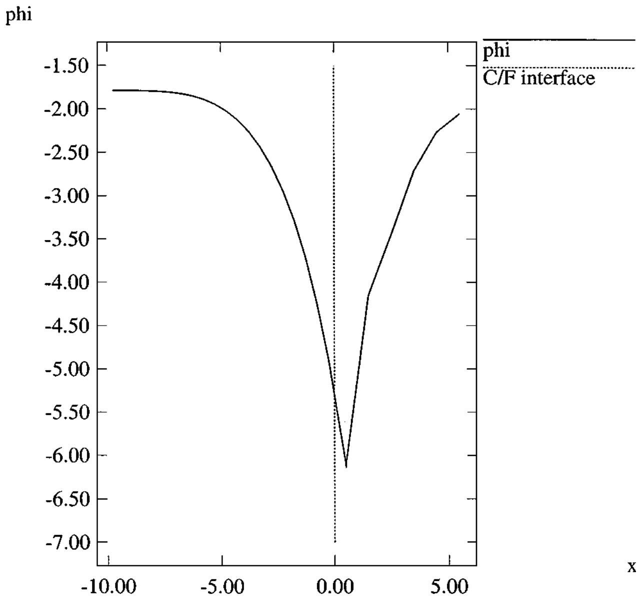


FIG. 6. Typical synchronization correction, ϕ. Fine grid is to the left of the coarse–fine interface.

for the coarse cell immediately adjacent to the coarse–fine interface, which is sufficient to preserve the structure of the correction. The composite cell-centered gradient can be written as $G^{\mathrm{CC,comp},\ell}(\phi^{\ell}, \phi^{\ell-1})$, while the level-operator cell-centered gradient can be expressed $G^{\mathrm{CC},\ell}(\phi^{\ell}, \phi^{\ell-1})$. Likewise, the composite and level-operator cell-centered divergence operators can be written $D^{\mathrm{CC,comp},\ell}(\mathbf{u}^{\mathrm{CC},\ell}, \mathbf{u}^{\mathrm{CC},\ell+1})$ and $D^{\mathrm{CC,comp},\ell}(\mathbf{u}^{\mathrm{CC},\ell})$, respectively.

Discretization of the Composite Projection

Our composite projection is based on the composite cell-centered operators $G^{\mathrm{CC,comp}}$, $D^{\mathrm{CC,comp}}$, and L^{comp}. Since the composite Laplacian is already a cell-centered operator, it is used without modification. Similar operators were used in the nonsubcycled algorithm in [27, 28]. This projection is applied to a multilevel velocity field for all levels finer than and including ℓ_{base}:

$$\mathbf{P}^{\mathrm{comp}} = (I - G^{\mathrm{CC,comp}}(L^{\mathrm{comp}})^{-1}D^{\mathrm{CC,comp}}) \quad \text{on} \bigcup_{\ell \geq \ell_{\mathrm{base}}} \Omega^{\ell}_{\mathrm{valid}}. \tag{37}$$

We first solve for a correction field ϕ,

$$\begin{aligned} L^{\mathrm{comp}}\phi^{\mathrm{comp}} &= D^{\mathrm{CC,comp}}\mathbf{u} \quad \text{for } \ell \geq \ell_{\mathrm{base}} \\ \phi^{\ell_{\mathrm{base}}} &= I(\phi^{\ell_{\mathrm{base}}}, \phi^{\ell_{\mathrm{base}}-1}), \end{aligned} \tag{38}$$

with appropriate physical boundary conditions; we then apply the correction

$$\begin{aligned} \mathbf{u} &:= \mathbf{u} - G^{\mathrm{CC,comp}}\phi^{\mathrm{comp}} \quad \text{for } \ell \geq \ell_{\mathrm{base}} \\ \phi^{\ell_{\mathrm{base}}} &= I(\phi^{\ell_{\mathrm{base}}}, \phi^{\ell_{\mathrm{base}}-1}). \end{aligned} \tag{39}$$

Note that coarse–fine boundary conditions have been given for the case where $\ell_{\mathrm{base}} > 0$.

The discrete composite projection operator we use is

$$\mathbf{P}^{\mathrm{comp}} = I - Av^{\mathrm{E}\to\mathrm{C}}G^{\mathrm{comp}}(L^{\mathrm{comp}})^{-1}D^{\mathrm{comp}}Av^{\mathrm{C}\to\mathrm{E}}. \tag{40}$$

For a uniform single grid with periodic boundary conditions, Lai [24] demonstrated using Fourier analysis that this projection discretization is stable, in that $\|\mathbf{P}\| \leq 1$, and that repeated application of the projection drives the divergence to zero,

$$D^{\mathrm{CC}}(\mathbf{P}^{N}\mathbf{u}) \to 0,$$

as $N \to \infty$, where $\mathbf{P}^{N}(\mathbf{u})$ represents the repeated application of the projection N times to the velocity field $\mathbf{u}$. To demonstrate the effectiveness and stability of this composite projection, we repeatedly applied the composite projection to a sample problem and evaluated the results. This was performed on a three-vortex test case. Each vortex has the initial condition

$$u_{\theta}(r) = \begin{cases} \Gamma\left(\frac{8}{3R^5}r^4 - \frac{5}{R^4}r^3 + \frac{10}{3R^2}r\right) & \text{if } r < R, \\ \Gamma\left(\frac{1}{r}\right) & \text{if } r \geq R, \end{cases} \tag{41}$$

where u_{θ} is the azimuthal velocity component around the vortex center, r is the radial distance from the vortex center (x_0, y_0), R is the radius of the vortex patch, and Γ is the vortex strength. For this problem, there are three vortices with equal strength and

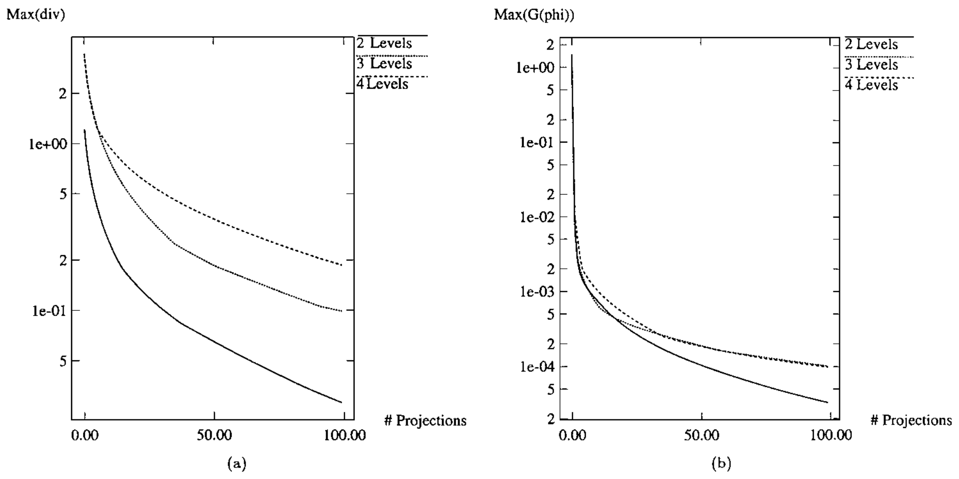


FIG. 7. (a) Max(divergence) and (b) max($G^{\text{comp}}(e)$) vs number of repeated projection applications.

radius $\Gamma = 0.50$ and $R = 0.75$ which are centered at (0.68, 0.5), (0.455, 0.65588457), and (0.455, 0.34411543).

Figure 7a shows $|D^{\text{comp}}\mathbf{u}|_\infty$ vs N. The composite divergence does go to zero as the projection is applied repeatedly. Adding more levels of refinement affects the rate at which the divergence decreases, but does not appear to affect the general behavior. (While the maximum divergence may seem high, it is not unreasonably so—Max($D(\mathbf{u})$) is an expression of the local truncation error and will tend to exaggerate isolated values along the coarse–fine interface, which will have only a minor effect on the solution accuracy.) A better indicator is the amount that each application of the projection changes the solution; $(\mathbf{P}^n - \mathbf{P}^{n+1})$ is equal to $(\mathbf{I} - \mathbf{P})\mathbf{P}^n$, which is $G^{\text{CC,comp}}e^n$. We expect each successive projection to have a smaller effect on the solution, as the velocity field converges toward one which is discretely divergence-free. Figure 7b shows $|G^{\text{CC,comp}}(e)|_\infty$, the maximum that the solution is changed in a given application of the projection (note that the maximum velocity for this problem is about 10). This value decreases monotonically as the projection is repeatedly applied. The magnitude of the correction is much larger in the first application of the projection because the physical boundary condition (solid walls, in this case) is being enforced; the velocity field is initialized as if it were in infinite space and then the initial projection enforces the no-flow boundary conditions. After a few iterations, $|G^{\text{CC,comp}}(e)|_\infty$ has decreased by three orders of magnitude, as the error in the velocity field decreases.

3.5. Freestream Preservation

The starting point for our approach to freestream preservation is the introduction of an auxiliary advected scalar field Λ whose purpose is to provide a time-integrated measure of the extent to which freestream preservation has been violated:

$$\frac{\partial \Lambda}{\partial t} + \nabla \cdot (\mathbf{u}\Lambda) = 0, \tag{42}$$

$$\Lambda(\mathbf{x}, t = 0) = 1. \tag{43}$$

Since Λ should remain one everywhere, $\Lambda \neq 1$ is an indicator of freestream preservation errors. We use Λ to compute a correction velocity field $\mathbf{u}_p$, which is added to future advection velocities and works to drive Λ back to one. Conserved scalars s satisfying (15) (including Λ) are then evolved using the equation

$$s^\ell(t^\ell + \Delta t^\ell) = s^\ell(t^\ell) - \Delta t D^\ell(\mathbf{u}^{\mathrm{AD}} s^\ell), \tag{44}$$

$$\mathbf{u}^{\mathrm{AD}} = \mathbf{u}^{\mathrm{half},\ell} + \mathbf{u}_{\mathrm{p}}^\ell. \tag{45}$$

The field $\mathbf{u}_{\mathrm{p}}$ is a potential flow field computed by solving an elliptic equation,

$$\mathbf{u}_{\mathrm{p}} = G e_\Lambda \tag{46}$$

$$L e_\Lambda = \zeta(\Lambda - 1), \zeta > 0. \tag{47}$$

Like the projection operator, the physical boundary conditions for e_Λ are $\partial e_\Lambda / \partial \mathbf{n} = \mathbf{u}_{\mathrm{p}} \cdot \mathbf{n}$. The field $\mathbf{u}_{\mathrm{p}}$ is computed during the synchronization step for all levels $\ell \geq \ell_{\mathrm{base}}$. Since Λ is updated using a conservative scheme, (47) is solvable.

If $\Lambda \lesseqgtr 1$, then $\nabla \cdot \mathbf{u}_{\mathrm{p}} \lesseqgtr 0$. This leads to the evolution equation for Λ

$$\Lambda^\ell(t + \Delta t^\ell) = \Lambda^\ell - \Delta t^\ell D^\ell(\mathbf{u}^{\mathrm{AD}} \Lambda) \tag{48}$$

$$= \Lambda^\ell(t) - \Delta t^\ell (\mathbf{u}^{\mathrm{AD}} \cdot G\Lambda)^{\mathrm{cen}} - \Delta t^\ell \overline{\Lambda}^\ell D^\ell \mathbf{u}_{\mathrm{p}}, \tag{49}$$

where $\overline{\Lambda}^\ell$ and $(\mathbf{u}^{\mathrm{AD}} \cdot G\Lambda)^{\mathrm{cen}}$ are obtained by using a discrete Leibniz rule. So, the qualitative behavior of this correction is to drive Λ back to one.

For dimensional consistency, ζ must have units of $1/T$. Since (47) is solved during a synchronization operation, we use $1/\Delta t^{\mathrm{sync}}$. We also include a scaling factor, η, to adjust the strength of the correction. The equation we solve is

$$\begin{aligned} L^{\mathrm{comp}} e_\Lambda &= \frac{\Lambda - 1}{\Delta t^{\mathrm{sync}}} \eta \quad \text{for } \ell \geq \ell_{\mathrm{base}} \\ e_\Lambda^{\ell_{\mathrm{base}}} &= I\left(e_\Lambda^{\ell_{\mathrm{base}}}, e_\Lambda^{\ell_{\mathrm{base}}-1}\right), \end{aligned} \tag{50}$$

$$\mathbf{u}_{\mathrm{p}} = G^{\mathrm{comp}} e_\Lambda. \tag{51}$$

The parameter η in this formulation is the number of ℓ_{base} time steps it will take for Λ to be returned to one. We have found that $\sigma\eta < 1/2$ is a sufficient condition for stability, where σ is the CFL number $(U\Delta t/\Delta x)$. For the problems examined in this work, we have generally used $\sigma = 0.5$ and $\eta = 0.9$.

To demonstrate the behavior of the freestream preservation correction, we computed solutions from an initial condition which is a pair of counterrotating vortices. Each vortex has an initial condition described by (41). For this problem, there are two counterrotating vortices, one with $\Gamma = 0.35$, $R = 0.15$, and centered at $(x_0, y_0) = (0.3, 0.65)$, and the second with $\Gamma = -0.35$, $R = 0.15$, and $(x_0, y_0) = (0.3, 0.35)$. The domain is the unit square, with solid-wall boundary conditions. The result of this initial condition is that the vortex pair translates to the right. For this problem, we use a 32×32 base grid with one level of refinement with $n_{\mathrm{ref}} = 4$. Grids are placed dynamically and follow the vortices; regridding is done every two coarse-grid time steps. Figure 8 shows the vorticity distribution at time $t = 0.36$.

Figure 9 shows the distribution of the Λ field after 2 and 100 time steps if no correction is applied, while Fig. 10 shows Λ when the freestream preservation correction is applied. To

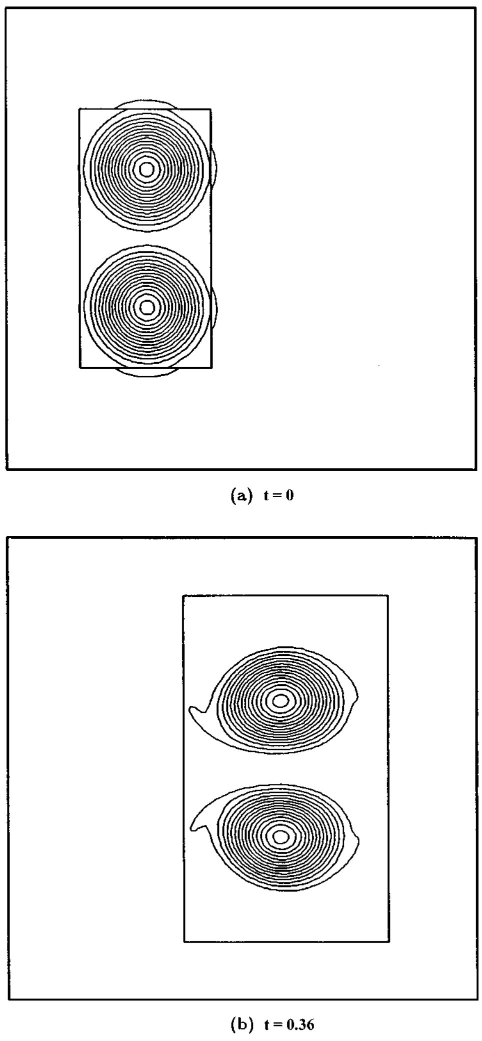
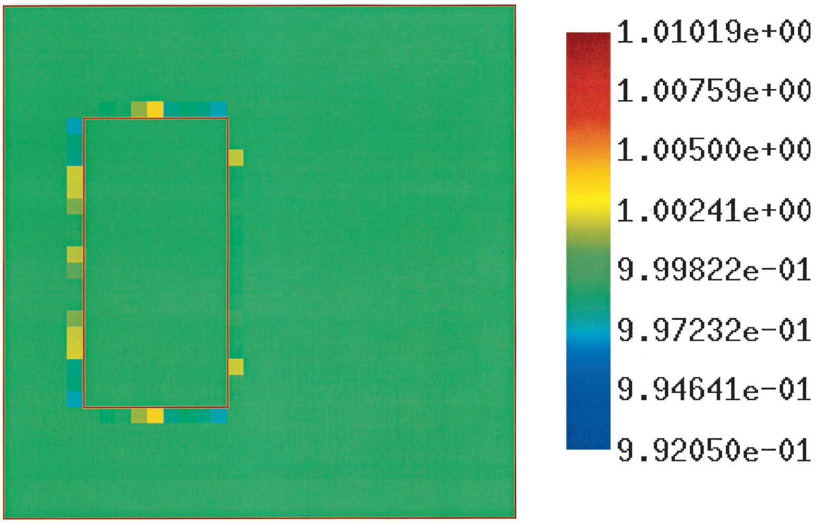


FIG. 8. Two-vortex solution (a) at initial time and (b) after 100 time steps.

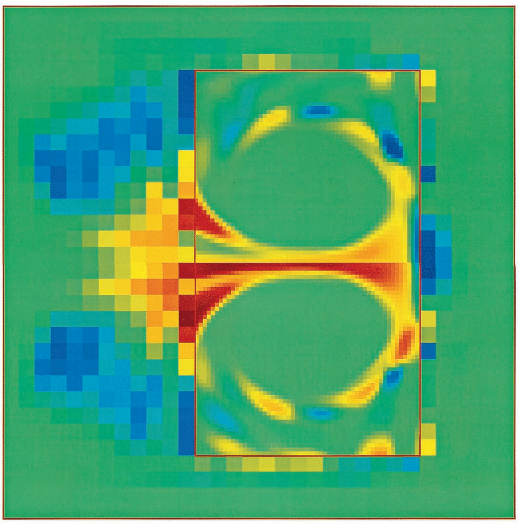
isolate the effect of the freestream preservation correction, the synchronization projection is applied in both cases. Note that the scales are different for the two figures.

Recall that $(\Lambda - 1)$ is the error in Λ caused by failures of freestream preservation. Without the correction (Fig. 9), freestream preservation errors are generated at coarse–fine interfaces as the solution evolves. Furthermore, as the grids move with the vortices, each new coarse–fine interface results in the creation of a new set of errors, which is left behind once the refined grids have moved. These errors are then advected throughout the flow, quickly becoming nonlocal. In contrast, even with moving grids, Figure 10 shows that the correction tends to confine errors to the cells immediately adjacent to the coarse–fine interfaces and limits them to approximately the error generated in one time step. Since the correction is a lagged one, this is what is expected. Although this one-cell-wide error is left behind when the coarse–fine interface moves, it is quickly removed by the action of the correction.

To further examine the advection errors for this case, we ran a series of cases with 32×32, 64×64, and 128×128 base grids, each with one level of refinement. To judge

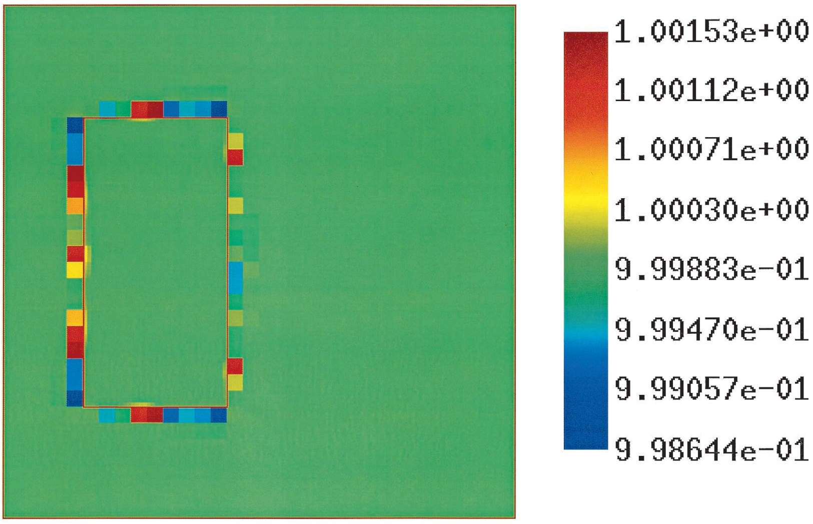


(a)

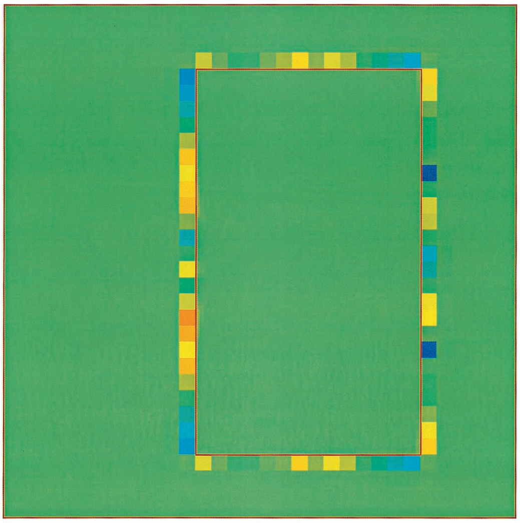


(b)

FIG. 9. Λ (without volume-discrepancy correction) after (a) 2 and (b) 100 time steps.

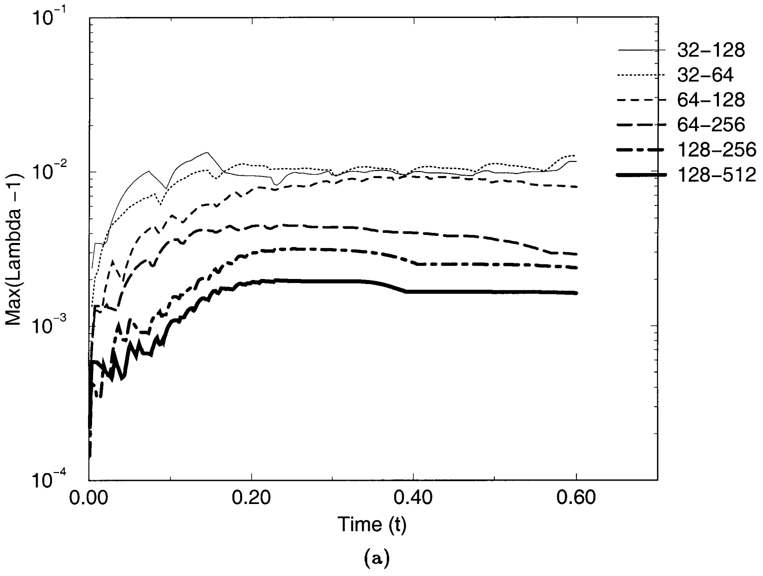


(a)



(b)

FIG. 10. Λ (with volume-discrepancy correction) after (a) 2 and (b) 100 time steps.



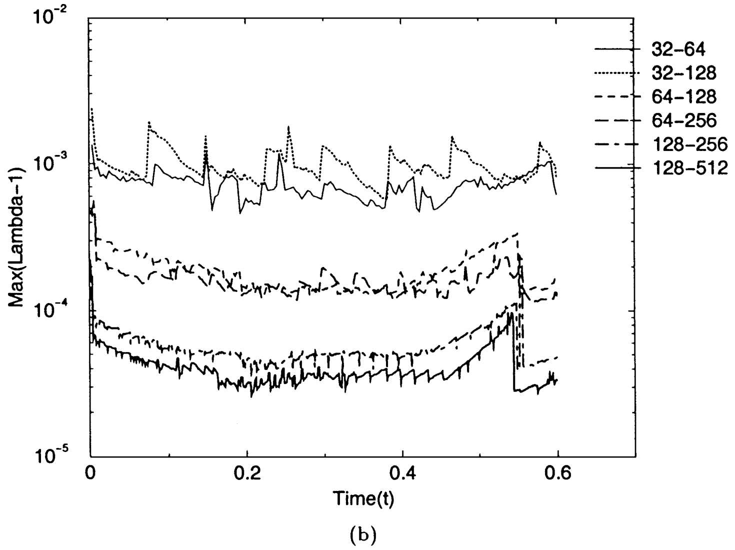


FIG. 11. Max($\Lambda - 1$) vs time for the traveling vortex pair case: (a) without freestream preservation correction and (b) with correction. Once again, note that (a) and (b) have different scales.

the effects of refinement ratio on the advection errors, each case was run with both $n_{\mathrm{ref}} = 2$ and $n_{\mathrm{ref}} = 4$. $|\Lambda - 1|_\infty$ is plotted vs time for these cases in Fig. 11. Without the correction (Figure 11a), the advection errors are to first approximation a function of the coarse-grid spacing; the effect of the refinement ratio is only secondary. Also, without the freestream preservation correction, the errors converge at roughly $O(h_c)$, where h_c is the coarse-grid spacing.

Comparison of the scales of Figs. 11a and 11b demonstrates that the volume discrepancy correction drastically reduces the maximum error. Because of regridding, the max(Λ) field in the corrected case is much more oscillatory than the uncorrected case; as the grids move, the correction field must adjust to the new grid configuration. Since we observe that the correction restricts the error to something less than the error made in one time step, and we expect this error to be no worse than $O(h_c)\Delta t^c$, where Δt^c is the coarse-grid time step (which is $O(h_c)$), this restores second-order accuracy to this aspect of the method.

We have also found that the freestream preservation correction does not corrupt scalar fields which are nonconstant in the vicinity of coarse–fine interfaces. To demonstrate this, we take the case of a single vortex in a unit square domain with solid walls, Eq. (41) with $\Gamma = 1.0$, $R = 0.3$, and $(x_0, y_0) = (0.5, 0.5)$. We initialize an advected scalar s to have a Gaussian distribution:

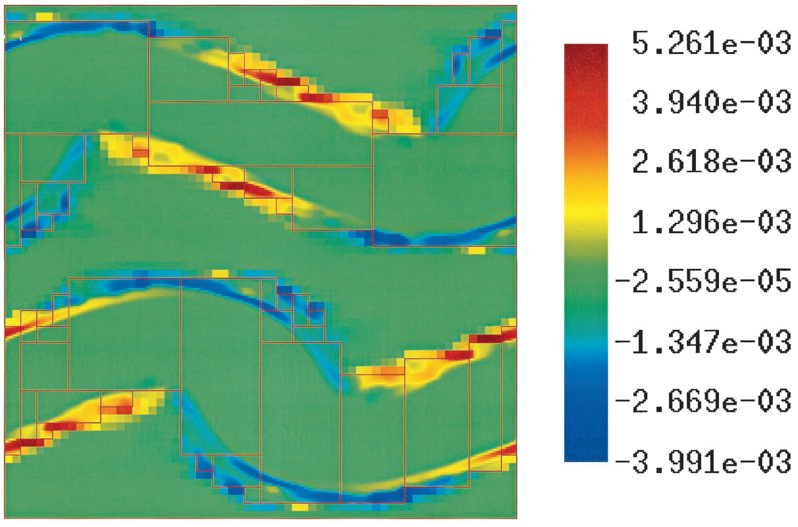
$$\begin{aligned} s(x, y, 0) &= e^{-r_{\text{blob}}^2/R}, \\ r_{\text{blob}}^2 &= (x - 0.35)^2 + (y - 0.35)^2. \end{aligned} \tag{52}$$

For this test case, we refined the lower-left quadrant of the grid and computed the solution to time $t = 6.4$. As in the previous case, a 1024×1024 uniform-grid solution was used as the "exact" solution. Errors in s both with and without the freestream preservation correction are tabulated in Table I.

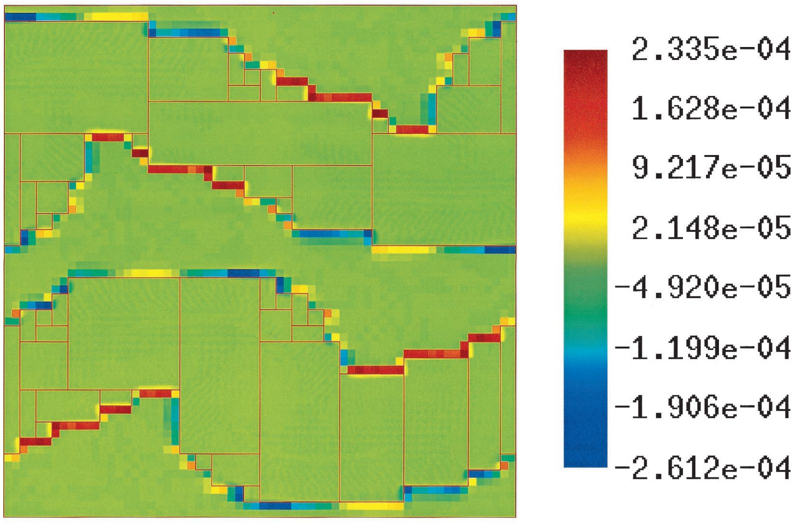
If a scalar field s is non-constant, the errors in freestream preservation are small in comparison to the variations in the solution, so the effect due to the freestream preservation correction is small; the freestream preservation violations which are so apparent in Λ are not significant for s, since they are overwhelmed by variations in the solution. While the freestream preservation correction does not improve the accuracy of advected scalars in this case, neither does it corrupt the solution. In contrast, a constant scalar near a coarse–fine interface mirrors the behavior of Λ; the use of the freestream preservation correction in this case drastically reduces the solution errors and increases the convergence rates, which

TABLE I
L_2 and L_∞ Norms of Errors in Advected Scalar s for the Single-Vortex Test Problem at Time 6.4

	Base Grid Size h			
	1/32	1/64	1/128	1/256
L_2: Uniform Grid	1.248e-2	4.247e-3	1.606e-3	5.458e-4
$n_{\text{ref}} = 2$: with correction	1.133e-2	3.800e-3	1.389e-3	—
without correction	1.122e-2	3.781e-3	1.386e-3	—
$n_{\text{ref}} = 4$: with correction	1.119e-2	3.730e-3	—	—
without correction	1.104e-2	3.705e-3	—	—
L_∞: Uniform Grid	1.143e-1	4.965e-2	2.391e-2	9.432e-3
$n_{\text{ref}} = 2$: with correction	9.888e-2	4.178e-2	2.057e-2	—
without correction	9.720e-2	4.181e-2	2.059e-2	—
$n_{\text{ref}} = 4$: with correction	9.712e-2	4.046e-2	—	—
without correction	9.505e-2	4.050e-2	—	—



(a)



(b)

FIG. 18. Error in freestream preservation indicator Λ at time $t = 0.75$: (a) without freestream preservation correction; (b) with correction. Note the different scales in the figures. Base grid size is 64×64, with one level of refinement, $n_{\text{ref}} = 4$.

demonstrates the importance of the freestream preservation correction when solutions are constant-valued at coarse–fine interfaces.

3.6. Multilevel Algorithm

In this section, we describe the complete recursive algorithm used to advance the level ℓ solution from time t^ℓ to time $t^\ell + \Delta t^\ell$. Implicit in this recursive algorithm is the subcycled advance of all finer levels to time $t^\ell + \Delta t^\ell$, including all necessary synchronization operations. A pseudocode description appears in Fig. 12.

```
EulerLevelAdvance($\ell$, $t^\ell$, $\Delta t^\ell$)
    Compute advection velocities $\mathbf{u}^{\mathrm{AD},\ell}$
    Compute advective fluxes $\mathbf{F}^{s,\ell}$, $\mathbf{F}^{\Lambda,\ell}$
    Compute advective updates:
        $s^\ell_{i,j}(t^\ell + \Delta t^\ell) = s^\ell_{i,j}(t^\ell) - \Delta t^\ell D^\ell(\mathbf{F}^{s,\ell})_{i,j}$
        $\Lambda^\ell_{i,j}(t^\ell + \Delta t^\ell) = \Lambda^\ell_{i,j}(t^\ell) - \Delta t^\ell D^\ell(\mathbf{F}^{\Lambda,\ell})_{i,j}$
    Predict $\mathbf{u}^{\mathrm{half}}$
    $\mathbf{u}^{*,\ell}_{i,j} = \mathbf{u}^\ell_{i,j}(t^\ell) - \Delta t[(\mathbf{u}\cdot\nabla)\mathbf{u}]^{n+1/2}_{i,j}$
    Update advective and velocity flux registers:
        if ($\ell < \ell_{\max}$) then
            $\delta s^{\ell+1} = -\mathbf{F}^{s,\ell}\cdot\mathbf{n}^{\ell+1}_{\mathrm{CF}}$   on $\mathcal{C}_{n^\ell_{\mathrm{ref}}}(\partial\Omega^{\ell+1,*})$
            $\delta\Lambda^{\ell+1} = -\mathbf{F}^{\Lambda,\ell}\cdot\mathbf{n}^{\ell+1}_{\mathrm{CF}}$   on $\mathcal{C}_{n^\ell_{\mathrm{ref}}}(\partial\Omega^{\ell+1,*})$
            $\delta\mathbf{V}^{\ell+1} = -\left(\mathbf{u}^{\mathrm{AD},\ell}\cdot\mathbf{n}^{\ell+1}_{\mathrm{CF}}\right)\mathbf{u}^{\mathrm{half},\ell}$   on $\mathcal{C}_{n^\ell_{\mathrm{ref}}}(\partial\Omega^{\ell+1,*})$
        end if
        if ($\ell > 0$) then
            $\delta s^\ell = \delta s^\ell + \frac{1}{n^{\ell-1}_{\mathrm{ref}}}\left\langle\mathbf{F}^{s,\ell}\cdot\mathbf{n}^\ell_{\mathrm{CF}}\right\rangle$   on $\mathcal{C}_{n^{\ell-1}_{\mathrm{ref}}}(\partial\Omega^{\ell,*})$
            $\delta\Lambda^\ell = \delta\Lambda^\ell + \frac{1}{n^{\ell-1}_{\mathrm{ref}}}\left\langle\mathbf{F}^{\Lambda,\ell}\cdot\mathbf{n}^\ell_{\mathrm{CF}}\right\rangle$   on $\mathcal{C}_{n^{\ell-1}_{\mathrm{ref}}}(\partial\Omega^{\ell,*})$
            $\delta\mathbf{V}^\ell = \delta\mathbf{V}^\ell + \frac{1}{n^{\ell-1}_{\mathrm{ref}}}\left\langle\left(\mathbf{u}^{\mathrm{AD},\ell}\cdot\mathbf{n}^\ell_{\mathrm{CF}}\right)\mathbf{u}^{\mathrm{half},\ell}\right\rangle$   on $\mathcal{C}_{n^{\ell-1}_{\mathrm{ref}}}(\partial\Omega^{\ell,*})$
        end if
    Project $\mathbf{u}^{*,\ell} \to \mathbf{u}^\ell(t^\ell + \Delta t^\ell)$:
        Solve $L^\ell\pi^\ell = \frac{1}{\Delta t^\ell} D^{\mathrm{CC},\ell}\mathbf{u}^{*,\ell}$
        $\mathbf{u}^\ell(t^\ell + \Delta t^\ell) = \mathbf{u}^{*,\ell} - \Delta t^\ell G^{\mathrm{CC},\ell}\pi^\ell$
    if ($\ell < \ell_{\max}$)
        $\Delta t^{\ell+1} = \frac{1}{n^\ell_{\mathrm{ref}}}\Delta t^\ell$
        for $n = 0, n^\ell_{\mathrm{ref}} - 1$
            EulerLevelAdvance($\ell + 1$, $t^\ell + n\Delta t^{\ell+1}$, $\Delta t^{\ell+1}$)
        end for
        if ($(t^\ell + \Delta t^\ell) < (t^{\ell-1} + \Delta t^{\ell-1})$)   Synchronize($\ell$, $t^\ell + \Delta t^\ell$, $t^\ell$)
    end if
end EulerLevelAdvance
```

FIG. 12. Recursive level time step for the incompressible Euler equations.

Variables

We start the level ℓ advance with the solution at time t^ℓ, which includes the velocity field $\mathbf{u}^\ell(\mathbf{x}, t^\ell) = (u^\ell, v^\ell)^T$, an advected scalar $s^\ell(\mathbf{x}, t^\ell)$, the freestream preservation scalar $\Lambda^\ell(\mathbf{x}, t^\ell)$, and the staggered-grid freestream preservation correction $\mathbf{u}_{\mathrm{p}}$ from the most recent synchronization step, which has been extended to the invalid regions on level ℓ with $\langle\mathbf{u}^{\ell+1}_{\mathrm{p}}\rangle$.

We also need flux registers to contain coarse–fine matching information. The flux register $\delta\mathbf{V}^\ell$ contains the normal and tangential (to the coarse–fine interface) momentum fluxes across the coarse–fine interface between level ℓ and the coarser level $\ell - 1$. The registers δs^ℓ and $\delta\Lambda^\ell$ contain the fluxes of the advected scalars s and Λ.

Level Advance

The basic update on a level is structured in a way similar to that of the single-grid algorithm described in the Appendix. Because most of the algorithm is the same, only the differences due to the adaptive algorithm are highlighted here.

1. Compute advection velocities. First, a set of staggered-grid advection velocities $\mathbf{u}^{\text{AD},\ell}$ is computed. Before the tracing and upwinding steps described in Appendix A.1 are performed, we fill a ring of ghost cells around each grid which is wide enough to complete the tracing stencil for all interior cells with appropriate solution values for $\mathbf{u}(t^\ell)$. If a level ℓ ghost cell lies in the interior of another level ℓ grid, solution values are copied from the other grid. If the ghost cell lies over a coarser grid's valid region, the coarse-grid solution $\mathbf{u}^{\ell-1}$ is interpolated in time and space, using conservative linear interpolation. Once ghost cells have been filled, computation of the staggered-grid $\mathbf{u}^{\text{half},\ell}$ can be carried out as detailed in the Appendix.

Then, $\mathbf{u}^{\text{half},\ell}$ is projected using a staggered-grid projection to ensure that the advection velocities are divergence-free:

$$\begin{aligned} L^\ell \phi^\ell &= D^\ell \mathbf{u}^{\text{half},\ell} \\ \phi^\ell &= I\left(\phi^\ell, \frac{\Delta t^\ell}{2}\pi^{\ell-1}\right). \end{aligned} \tag{53}$$

The coarse–fine boundary condition on ϕ is designed to ensure matching between ϕ^ℓ and the coarse-level pressure field. Then, the velocity field is corrected:

$$\begin{aligned} \mathbf{u}^{\text{half},\ell} &= \mathbf{u}^{\text{half},\ell} - G^\ell \phi^\ell \\ \phi^\ell &= I\left(\phi^\ell, \frac{\Delta t^\ell}{2}\pi^{\ell-1}\right). \end{aligned} \tag{54}$$

Finally, $\mathbf{u}_\text{p}$ from the most recent synchronization is added:

$$\mathbf{u}^{\text{AD},\ell} = \mathbf{u}^{\text{half},\ell} + \mathbf{u}_\text{p}. \tag{55}$$

2. Update scalars. Once advection velocities $\mathbf{u}^{\text{AD},\ell}$ have been computed, the scalars s^ℓ and Λ^ℓ can be updated. As in the previous step, a ring of ghost cells around each grid is filled by either copying values from other level ℓ grids or by performing a conservative linear interpolation in time and space of coarse-level data. Then, the advective fluxes $\mathbf{F}^{s,\ell}$ and $\mathbf{F}^{\Lambda,\ell}$, as well as the updated scalars $s(t^\ell + \Delta t^\ell)$ and $\Lambda^\ell(t^\ell + \Delta t^\ell)$, can be computed on a grid-by-grid basis using the conservative scalar advection algorithm detailed in the Appendix. The update equation used is

$$\{s, \Lambda\}^\ell(t^\ell + \Delta t^\ell) = \{s, \Lambda\}^\ell(t^\ell) - \Delta t^\ell D^\ell(\mathbf{u}^{\text{AD}}\{s, \Lambda\}^{\text{half},\ell}). \tag{56}$$

3. Predict $\mathbf{u}^{\text{half}}$ and $\mathbf{u}^{*,\ell}$. Using the advection velocities $\mathbf{u}^{\text{AD},\ell}$, the transverse components of the staggered-grid velocity field $\mathbf{u}^{\text{half},\ell}$ are computed as in Section A.3, using the same coarse–fine boundary conditions with the level $\ell - 1$ solution as in the original tracing step. The intermediate velocity field $\mathbf{u}^{*,\ell}$ is then computed,

$$\mathbf{u}^{*,\ell} = \mathbf{u}^\ell(t^\ell) - \Delta t^\ell[Av^{\text{E}\to\text{C}}(\mathbf{u}^{\text{AD},\ell}) \cdot (G^\ell \mathbf{u}^{\text{half},\ell})], \tag{57}$$

using the same discretization as Eq. (A.22) in the Appendix. Note the distinction in (57) between $\mathbf{u}^{\mathrm{AD}}$, the *advecting* velocity, and $\mathbf{u}^{\mathrm{half}}$, the *advected* velocity.

4. Initialize/update momentum and advective flux registers. Once the updates have been completed, the appropriate flux registers are updated to contain the mismatches between coarse and fine advective and momentum fluxes, as shown in Fig. 12.

5. Project $\mathbf{u}^{*,\ell} \rightarrow \mathbf{u}^{\ell}(t^{\ell} + \Delta t^{\ell})$. To complete the single-level portion of the level update, the intermediate velocity field $\mathbf{u}^{*,\ell}$ is projected using a level projection. First, solve

$$\begin{aligned} L^{\ell}\pi^{\ell} &= \frac{1}{\Delta t} D^{\mathrm{CC},\ell}\mathbf{u}^{*,\ell} \\ \pi^{\ell} &= I(\pi^{\ell}, \pi^{\ell-1}), \end{aligned} \tag{58}$$

where $\pi^{\ell-1}$ in the coarse–fine boundary condition is the most recent $\pi^{\ell-1}$, which is treated as piecewise constant in time throughout the subcycled level ℓ time steps. The correction is then applied to the velocity field:

$$\begin{aligned} \mathbf{u}^{\ell}(t^{\ell} + \Delta t^{\ell}) &= \mathbf{u}^{*,\ell} - \Delta t G^{\ell}\pi^{\ell} \\ \pi^{\ell} &= I(\pi^{\ell}, \pi^{\ell-1}). \end{aligned} \tag{59}$$

6. Recursively update finer levels. If a finer level $\ell + 1$ exists, it is then updated n_{ref}^{ℓ} times with a time step of $\Delta t^{\ell+1} = (1/n_{\mathrm{ref}}^{\ell})\Delta t^{\ell}$. This brings all levels finer than level ℓ to time $t^{\ell} + \Delta t^{\ell}$.

7. Synchronize. If a finer level $\ell + 1$ exists, we now synchronize level ℓ with all finer levels. We denote the time at which this synchronization takes place as $t^{\mathrm{sync}} = t^{\ell} + \Delta t^{\ell}$. The coarsest level which has reached t^{sync} is denoted as ℓ_{base}; all levels finer than and including ℓ_{base} are synchronized at once. In practice, we check to see if the current level has reached the new time of the coarser level, $(t^{\ell-1} + \Delta t^{\ell-1})$. If so, we drop down to the coarser level. We also denote $\Delta t^{\ell_{\mathrm{base}}}$ as Δt^{sync}, the time interval over which the synchronization is taking place. A pseudocode description of the synchronization step appears in Fig. 13. First, the finer level solutions are averaged down to underlying coarse grids and a refluxing operation is performed to correct coarse-level fluxes. Then, a multilevel sychronization projection is applied, which solves for the synchronization correction e_{sync}. The appropriate physical boundary conditions for e_{sync} are the homogeneous form of the boundary conditions applied to the level pressure π in the level projection. For solid walls, this is a homogeneous Neumann boundary condition. Finally, the freestream preservation correction $\mathbf{u}_{\mathrm{p}}$ is computed.

3.7. Initialization

After a new grid configuration is defined, either at the initial time or after a regridding operation, the solution must be initialized, as described in pseudocode form in Fig. 14. Before the initial time step, the velocity field must be projected to ensure that it satisfies the composite divergence constraint, so the composite projection defined in Section 3.4 is applied to the entire composite velocity field.

After a new grid configuration is defined for a level ℓ during a regridding operation, solution values for $\mathbf{u}^{\ell}$, s^{ℓ}, and Λ^{ℓ} are either copied from the old level ℓ grids where

```
Synchronize($\ell_{\text{base}}, t^{\text{sync}}, \Delta t^{\text{sync}}$)
    Average finer solution onto coarser levels:
        for $\ell = \ell_{\max} - 1, \ell_{\text{base}}, -1$
            $\mathbf{u}^\ell(t^{\text{sync}}) = \langle \mathbf{u}^{\ell+1}(t^{\text{sync}}) \rangle$   on $\mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1})$
            $s^\ell(t^{\text{sync}}) = \langle s^{\ell+1}(t^{\text{sync}}) \rangle$   on $\mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1})$
            $\Lambda^\ell(t^{\text{sync}}) = \langle \Lambda^{\ell+1}(t^{\text{sync}}) \rangle$   on $\mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1})$
        end for
    Reflux for conservation:
        for $\ell = \ell_{\max} - 1, \ell_{\text{base}}, -1$
            $\mathbf{u}^\ell(t^{\text{sync}}) := \mathbf{u}^\ell(t^{\text{sync}}) - \Delta t^\ell D^\ell_{\text{R}}(\delta \mathbf{V}^{\ell+1})$
            $s^\ell(t^{\text{sync}}) := s^\ell(t^{\text{sync}}) - \Delta t^\ell D^\ell_{\text{R}}(\delta s^{\ell+1})$
            $\Lambda^\ell(t^{\text{sync}}) := \Lambda^\ell(t^{\text{sync}}) - \Delta t^\ell D^\ell_{\text{R}}(\delta \Lambda^{\ell+1})$
        end for
    Apply Synchronization Projection:
        Solve $L^{\text{comp}} e_s = \frac{1}{\Delta t^{\text{sync}}} D^{\text{CC,comp}} \mathbf{u}(t^{\text{sync}})$   for $\ell \geq \ell_{\text{base}}$
            $e_s^{\ell_{\text{base}}} = I\left(e_s^{\ell_{\text{base}}}, e_s^{\ell_{\text{base}}-1}\right)$
        $\mathbf{u}(t^{\text{sync}}) := \mathbf{u}(t^{\text{sync}}) - \Delta t^{\text{sync}} G^{\text{CC,comp}} e_s$   for $\ell \geq \ell_{\text{base}}$
    Freestream Preservation Solve:
        Solve $L^{\text{comp}} e_\Lambda = \frac{(\Lambda(t^{\text{sync}}) - 1)}{\Delta t^{\text{sync}}} \eta$   for $\ell \geq \ell_{\text{base}}$
            $e_\Lambda^{\ell_{\text{base}}} = I\left(e_\Lambda^{\ell_{\text{base}}}, e_\Lambda^{\ell_{\text{base}}-1}\right)$
        $\mathbf{u}_{\text{p}} = G^{\text{comp}} e_\Lambda$
end Synchronize
```

FIG. 13. Synchronization for incompressible Euler equations.

```
EulerInit($\ell_{\text{base}}, t^{\text{init}}$)
    Project initial data
        Solve $L^{\text{comp}} e = D^{\text{CC,comp}} \mathbf{u}$   for $\ell \geq \ell_{\text{base}}$
            $e^{\ell_{\text{base}}} = I(e^{\ell_{\text{base}}}, 0)$
        $\mathbf{u} := \mathbf{u} - G^{\text{CC,comp}} e$
    Compute initial $\mathbf{u}_p$:
        Solve $L^{\text{comp}} e_\Lambda = \frac{(\Lambda - 1)}{\Delta t^{\ell_{\text{base}}}} \eta$   for $\ell \geq \ell_{\text{base}}$
            $e_\Lambda^{\ell_{\text{base}}} = I\left(e_\Lambda^{\ell_{\text{base}}}, e_\Lambda^{\ell_{\text{base}}-1}\right)$
        $\mathbf{u}_{\text{p}} = G^{\text{comp}} e_\Lambda$
end EulerInit
```

FIG. 14. Initialization algorithm for the incompressible Euler equations.

available or interpolated from the finest level available using conservative interpolation in time and space. This new velocity is then projected using a composite projection, with a coarse–fine boundary condition for ℓ_{base} of $e^{\ell_{\text{base}}} = I(e^{\ell_{\text{base}}}, 0)$. For initialization purposes, ℓ_{base} is defined as the finest *unchanged* level in the grid hierarchy.

Also, after a regridding operation, there are existing freestream preservation errors which must be corrected in future time steps. The advection correction $\mathbf{u}_{\text{p}}$ is computed using the current Λ field to correct for these errors.

3.8. Comparison with Earlier Work

There have been a number of earlier works which apply block-structured AMR to incompressible flow. Thompson and Ferziger [34] constructed an algorithm for steady flow. The algorithms of Howell and Bell [21] and Minion [27, 28] are also based on cell-centered discretizations of the projection, but they do not refine in time.

The main earlier work relevant to this paper is that of Almgren *et al.* [3], which is also second order in space and time, refines in time as well as space, and is freestream-preserving. The discretization of the projection employed in [3] is based on that of Almgren *et al.* [2], which uses a discrete Galerkin formulation. This discretization has the advantage of proven stability ($\|P\| \leq 1$) and accuracy. However, it is node-centered: pressures are centered at the corners of cells. Since a set of cell-centered solvers must be used to compute advection velocities and for viscous solves, this means that two sets of solvers must be developed and maintained, which can substantially increase the effort involved in extension to new problems and more complicated geometries. One of the ultimate goals of this work is to combine the AMR algorithms for incompressible flow with the Cartesian grid embedded boundary methods for representing complex boundary geometries [4, 5, 22, 29]. In this approach, conservative cell-centered discretizations can all be implemented using a common software framework [17]. The introduction of node-centered discretizations and solvers required for the approximate projection in [2, 3] would considerably enlarge and complicate the software support required.

Also, note that in this algorithm the projection is applied to an approximation of the updated velocity field at the new time ($\mathbf{u}^*$). Other implementations of the projection, including those in [3, 27], project the right-hand-side of the velocity update; for example,

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \mathbf{P}\big(-(\mathbf{u}\cdot\nabla\mathbf{u})^{n+1/2} - \nabla p^{n-1/2}\big) \tag{60}$$

$$\nabla p^{n+1/2} = \nabla p^{n-1/2} + (\mathbf{I} - \mathbf{P})\big((\mathbf{u}\cdot\nabla\mathbf{u})^{n+1/2} + \nabla p^{n-1/2}\big). \tag{61}$$

While the two discretizations are equivalent for a projection which is idempotent, for approximate projections (in which $L \neq DG$ and so $\mathbf{P} \neq \mathbf{P}^2$), these temporal discretizations are no longer equivalent. We were unable to construct a stable multilevel projection algorithm for (60) and (61) using a cell-centered approximate projection; however, our algorithm appears to be well behaved. This is consistent with the findings of Rider [32] and Almgren *et al.* [7] who have observed that projecting an approximation to the velocity field is more robust than projecting the update to the velocity field. However, the non-subcycled algorithm in [27] successfully used the formulation in (24)–(25) with a cell-centered spatial discretization similar to the one used in this work, which tends to indicate that our difficulties in this case are due to refinement in time.

In [3], freestream preservation is maintained exactly by computing a correction to the advection velocity field, performing a correction advection step, and then interpolating the corrections to finer levels. This method has been used successfully in a variety of applications. However, it introduces an interpolated coarse-grid approximation of the advection operator at all the finer levels, which could conceivably lead to an unacceptable loss of accuracy. In the present approach, we avoid such errors by the use of the volume discrepancy approach, at the cost of satisfying the freestream condition only approximately.

The algorithm in this work also differs from both the hyperbolic algorithm in [10] and the incompressible flow algorithm in [3] in the structure of the synchronization. In places where the synchronization operations require the solution of elliptic equations, we solve at once for all levels which have reached the same time (t^{sync}) rather than solving a series of two-level pairs. As a result, the composite projection and freestream correction solves are multilevel solutions for all levels finer than and including the coarsest level which has reached t^{sync}. There is some evidence [26] that a multilevel elliptic solve is more accurate than solving a series of coarse–fine pairs.

3.9. Solvers

The algorithm used in this work uses two elliptic solvers. The first is a single-level solver, which solves an elliptic equation using single-level operators (D^ℓ, G^ℓ, L^ℓ, etc.) on the union of rectangular grids which comprise an AMR level. If the level does not cover the entire domain, boundary conditions are taken from the next coarser level using the coarse–fine interpolation operator I. We use a straightforward multigrid approach with Gauss–Siedel with red–black ordering to relax on each multigrid level. For a general union of grids, there will be a point beyond which the grids cannot be coarsened without changing the shape of the grids; at that point, we cease coarsening and use a conjugate gradient solver as a bottom solver. Multilevel solutions of elliptic equations are found using the AMR-multigrid algorithm described in [6, 34] and also used in [13, 25, 26, 31]. For a typical two-level problem, with $n_{\text{ref}} = 2$ or 4, it takes our multilevel solver eight V-cycles to reduce the composite residual by 10 orders of magnitude.

4. RESULTS

We will demonstrate that this method is second-order accurate, that local refinement is effective at increasing the accuracy of the solution, and that the freestream-preservation correction is effective in more complicated situations. Also, we will demonstrate that this algorithm can result in significant savings in computational costs through the use of local refinement.

We use two test problems. First, we use the three-vortex problem described in Section 3.4, which is a good demonstration of the effectiveness of local refinement, since the area of interest (the vortices) is a small part of the entire domain.

We also use the doubly periodic shear layer in a unit square domain from [8], with the initial conditions

$$\begin{aligned} u(x, y, t = 0) &= \begin{cases} \tanh\left(\rho_s\left(y - \frac{1}{4}\right)\right) & \text{if } y \le \frac{1}{2}, \\ \tanh\left(\rho_s\left(\frac{3}{4} - y\right)\right) & \text{if } y > \frac{1}{2} \end{cases} \\ v(x, y, t = 0) &= \delta_s \sin(2\pi x), \end{aligned} \tag{62}$$

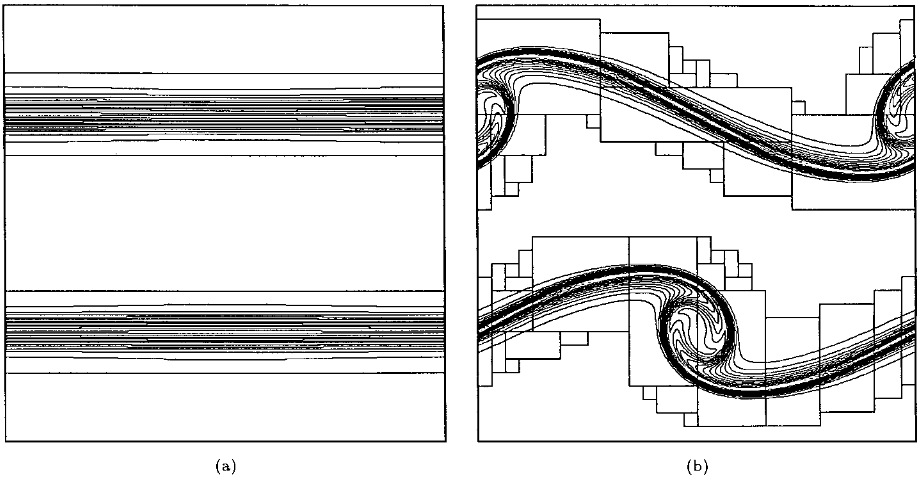


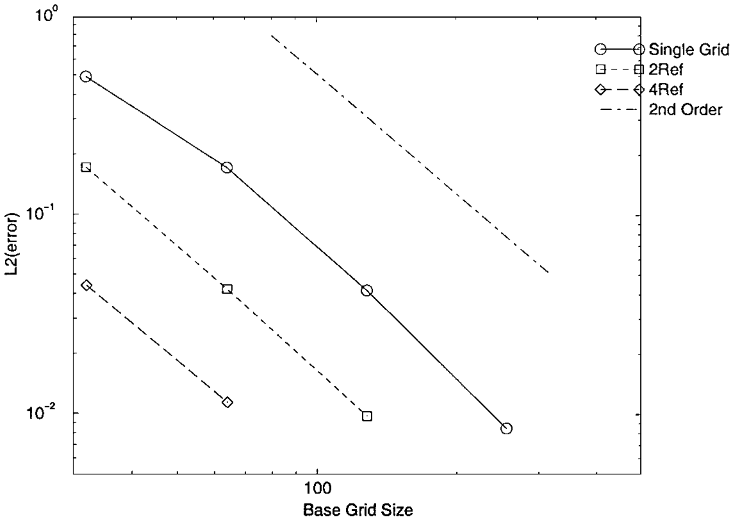
FIG. 15. Evolution of doubly periodic shear layer. Vorticity at times (a) $t = 0$ and (b) $t = 0.75$. The base grid is 64×64, and the refined grids represent a factor of 4 refinement. All contour plots use the same scale: $\mathrm{Max}(\omega) = 40.20$, $\mathrm{Min}(\omega) = -40.20$.

with $\rho_s = 42.0$ and $\delta_s = 0.05$. As this solution evolves, the shear layers roll up into two large vortices. Figure 15 shows the evolution of the vorticity and the adaptive nature of the refinement for a 64×64 base grid with a single level of refinement with $n_{\text{ref}} = 4$. This problem is not well suited for adaptivity, since a large portion of the domain must be refined; however, we believe it to be a more stringent test of the accuracy of the algorithm than the three-vortex problem, owing to the larger and more complicated coarse–fine interfaces generated.

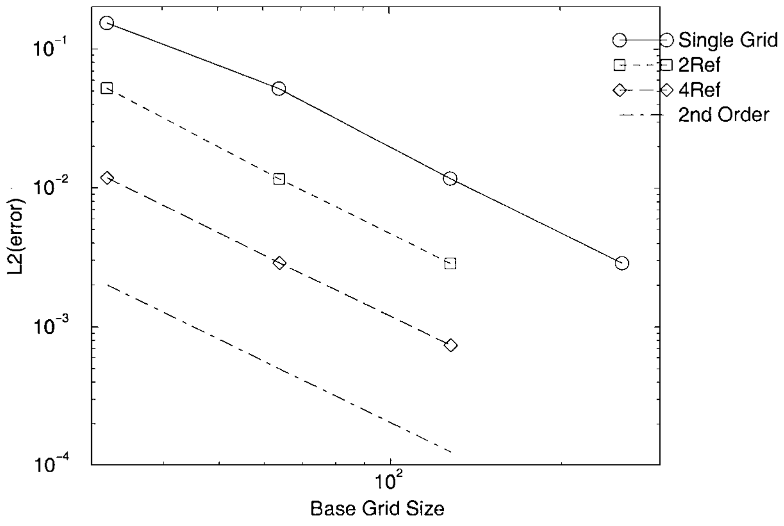
These runs were done with fixed time steps so that the results could be directly compared, with a CFL number of 0.5. Since no analytic solution exists for these problems, uniform-grid 1024×1024 solutions were computed and used as the "exact" solution. For the AMR cases, grids were placed dynamically, based on vorticity. Cells with a vorticity higher than $\epsilon|\omega|_\infty$ were tagged for refinement, with $\epsilon = 0.25$ for the problems examined here. Refined grids were then generated from the tagged cells using the clustering algorithm of Berger and Rigoutsos [12]. Regridding intervals varied to ensure that regridding was done at the same times in all cases. For example, the 32×32 base-grid cases used a regridding interval of three level 0 time steps, the 64×64 base-grid cases used a regridding interval of six time steps, and so on. To judge the effects of adaptivity, a series of adaptive calculations were run with one level of refinement with n^0_{ref} equal to 2 and 4. For comparison, a series of uniform grid computations was also carried out.

4.1. Accuracy and Convergence

If the adaptive algorithm is effective, we expect that the accuracy of the adaptive computations should approach that of the uniform-grid computation of equivalent resolution; we expect that a computation with a 64×64 base grid and one level of $n_{\text{ref}} = 4$ refinement should attain the same accuracy as a 256×256 uniform-grid computation. Figure 16 shows the errors in x-velocity for the three-vortex solution at time $t = 0.128$ and for the shear layer problem at time $t = 0.75$ (y-velocities are similar). In both cases, the adaptive solutions attain the accuracy of the single-grid solution with equivalent resolution, indicating



(a)



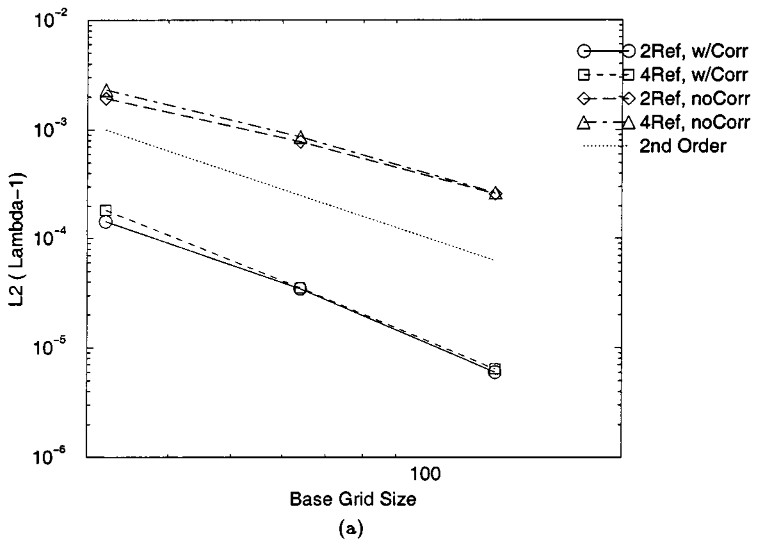
(b)

FIG. 16. Errors in x-velocity vs base grid size for (a) the three-vortex solution at $t = 0.128$ and (b) the shear layer problem at $t = 0.75$.

that local refinement is effective at increasing the accuracy of the solution. Also, the solution errors converge at second-order rates in L_2, except in the coarsest uniform-grid cases, where the solutions are under-resolved and so are not expected to be in the asymptotic regime. (Brown and Minion [14] found similar behavior in their study of the shear-layer problem.)

4.2. Freestream Preservation

To evaluate the performance of the freestream preservation correction for the shear-layer problem, L_2 and L_∞ norms of Λ are shown in Fig. 17 for time $t = 0.75$. The effects of



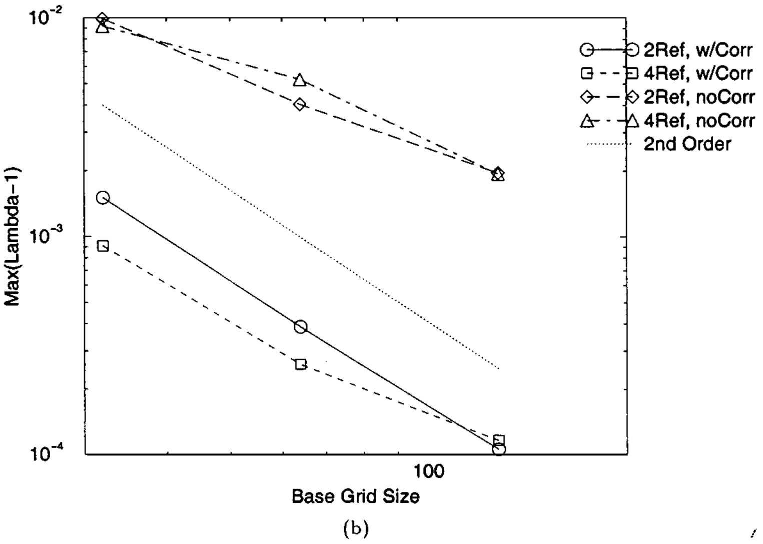


FIG. 17. Convergence of $\Lambda - 1$ for shear layer problem in (a) L_2 and (b) L_∞ norms. Adaptive cases were run with 32×32, 64×64, and 128×128 base grids, with one level of refinement, $n_{\text{ref}} = 2$ and 4. In all cases, the higher pair of lines is without the freestream preservation correction; lower sets of lines are with correction.

the correction are much more dramatic in this case than for the two-vortex case presented in Section 3.5, probably due to the more complicated refined-grid configurations generated for this flow. Without the volume-discrepancy correction, errors in Λ due to failures of freestream preservation are about an order of magnitude higher in L_2 and L_∞ norms. Also, the errors in Λ display markedly slower convergence without the correction. Figure 18 shows $(\Lambda - 1)$ at time $t = 0.75$ both with and without the correction. As in the two-vortex case, without correction, errors in Λ are generated at coarse–fine interfaces and then advected throughout the flow, contaminating the entire solution. With the correction, however, the errors are primarily confined to cells immediately adjacent to coarse–fine interfaces. (Note the difference in scales in Fig. 18 between the two cases.)

4.3. Synchronization Projection

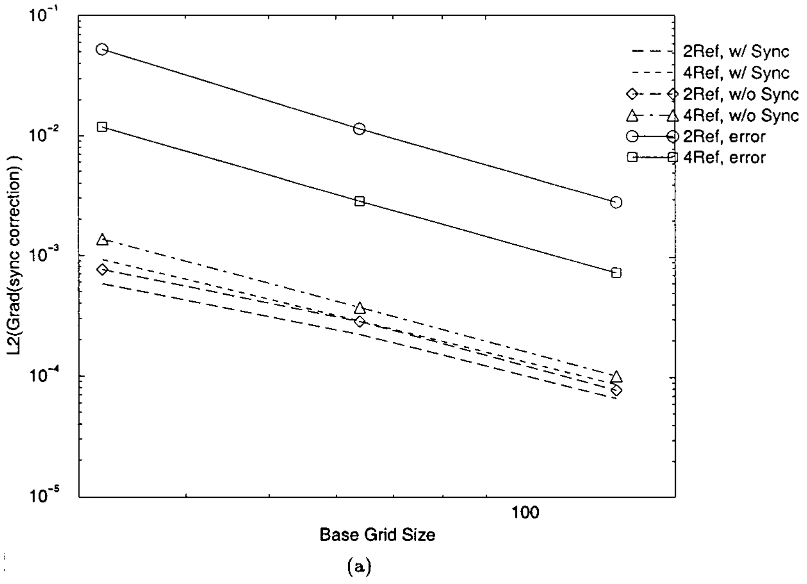
To determine the effect of the synchronization projection for this algorithm, these cases were also run without the synchronization projection, where the no-synchronization algorithm is the algorithm presented in Fig. 5. Comparison showed no noticeable accuracy difference as a result of the synchronization projection. This is in marked contrast to the results presented in [3], in which the synchronization projection was shown to be necessary to maintain the accuracy of the method. This behavior was found to be consistent across all problems on which we tested this algorithm—we were unable to find a test case for which the synchronization projection was necessary for accuracy or stability. We believe this to be a result of our projecting an approximation to the velocity field (as in (24) and (25)) instead of using the approximation to $\mathbf{u}_t$ ((60) and (61)), which is projected in [3]. For the $\mathbf{P}(\mathbf{u}_t)$ formulation, any non-divergence-free components of the velocity field which are not eliminated by the projection persist and accumulate, corrupting the solution, while for the $\mathbf{P}(\mathbf{u})$ formulation, these components are reduced or eliminated by subsequent applications of the projection [32, 33]. So, the projection formulation used in this work is more forgiving of errors made at coarse–fine interfaces, because during every time step, the velocity field is resubjected to at least a coarse approximation of the composite divergence constraint [7].

The obvious question is then whether the synchronization projection is really necessary for this algorithm. Performance data indicate that our code spends about 15% of its execution time performing the synchronization projection. Although the synchronization projection does not seem to have any effect on the accuracy of the method, it does have an impact on how well the solution satisfies the divergence constraint (13). Table II tabulates the L_2 and L_∞ norms of $D^{\text{comp}}(\mathbf{u})$. It is notable that while the no-synchronization results show higher divergences and slower convergence in all norms, the difference between the synchronization and no-synchronization cases is much less dramatic than in the results presented in [3]. The exception to this is in the L_∞ norm with $n_{\text{ref}} = 4$, where it appears that convergence stalls and actually begins to diverge. This is due to a single point (a convex corner in the refined-grid configuration) where the divergence remains large in the 64×64 base grid computation. The L_2 norms are not as affected.

TABLE II
L_2 and L_∞ Norms of Composite Divergence with and without Synchronization Projection for the Shear Layer Problem at Time = 0.75

	Base Grid Size h						
	1/32	Rate	1/64	Rate	1/128	Rate	1/256
L_2: Uniform Grid	1.545e-1	1.65	4.925e-2	1.65	1.569e-2	2.14	3.558e-3
$n_{\text{ref}} = 2$: with sync	2.761e-2	1.72	8.364e-3	2.05	2.026e-3	—	—
without sync	5.868e-2	1.14	2.661e-2	1.45	9.715e-3	—	—
$n_{\text{ref}} = 4$: with sync	1.237e-2	1.52	4.310e-3	1.60	1.423e-3	—	—
without sync	6.931e-2	1.18	3.053e-2	1.30	1.244e-2	—	—
L_∞: Uniform Grid	8.874e-1	1.15	3.990e-1	0.803	2.286e-1	1.25	9.617e-2
$n_{\text{ref}} = 2$: with sync	2.619e-1	0.827	1.477e-1	1.59	4.899e-2	—	—
without sync	3.998e-1	0.636	2.572e-1	0.956	1.326e-1	—	—
$n_{\text{ref}} = 4$: with sync	1.708e-1	1.33	6.805e-2	0.129	6.221e-2	—	—
without sync	4.529e-1	0.780	2.637e-1	−0.169	2.965e-1	—	—

Another way to judge the effect of the synchronization projection is to examine the magnitude of the correction to the flow field made as a result of the synchronization projection. Figure 19 shows the L_2 and L_∞ norms of $(1/\Delta t^{\text{sync}})G^{\text{comp}}e_s$ in the x-direction, which is the correction applied to the velocity field as a result of the synchronization projection. For the no-synchronization cases, the algorithm was run with the synchronization projection turned off, and then the composite projection was applied to the resulting velocity field—in these cases, $(1/\Delta t^{\text{comp}})G^{\text{comp}}e_s$ represents the correction needed to return the velocity field to compliance with the composite divergence constraint. These corrections are noteworthy mostly for their magnitude; for comparison, the solution errors are also included in Fig. 19.



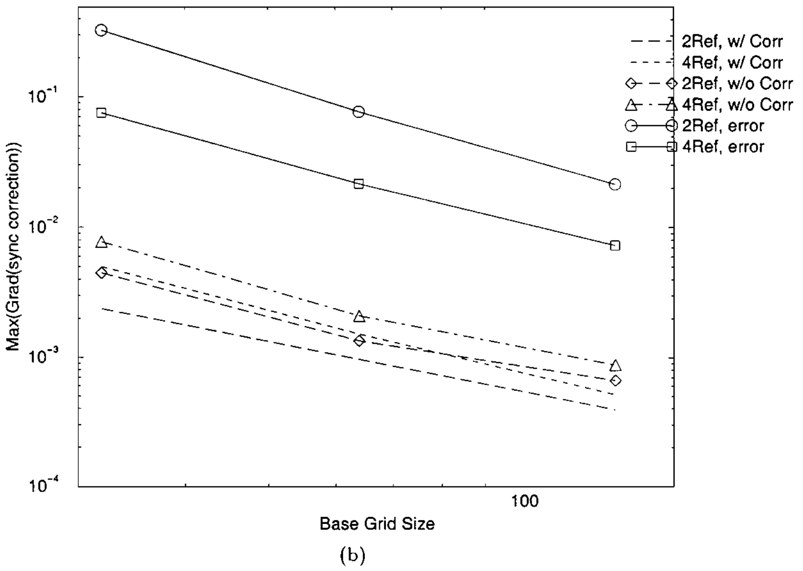
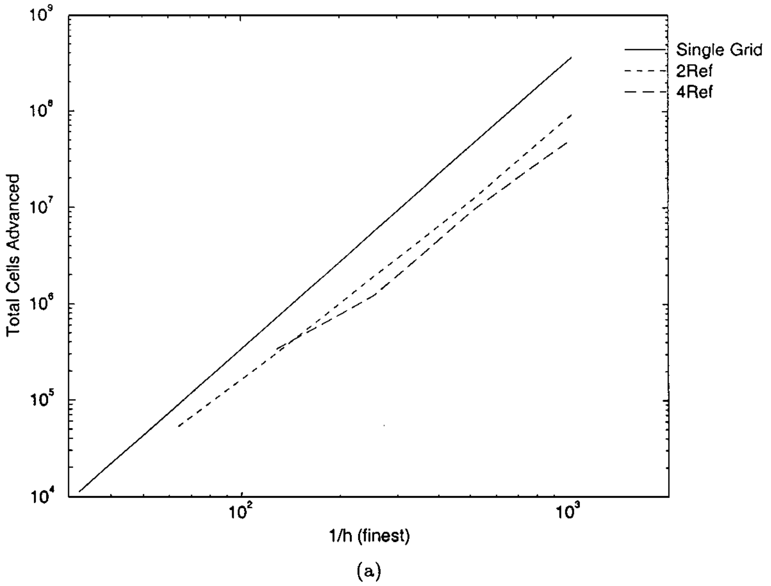


FIG. 19. (a) L_2 and (b) L_∞ norms of x-direction correction due to synchronization projection, compared with solution error. The synchronization corrections are marked with broken lines, while the solution errors are solid lines. Adaptive cases were run with 32×32, 64×64, and 128×128 base grids, with one level of refinement.

As can be seen, both with and without synchronization, the corrections are about an order of magnitude smaller than the solution error. Also, the difference between the cases with and without synchronization is quite small. We have observed similar behavior in all other test problems we have examined. As mentioned before, we believe that because we are projecting the velocity field, the level-projected velocity field does not stray far from the composite divergence constraint, since the level projections provide a reasonable approximation to the composite constraint.



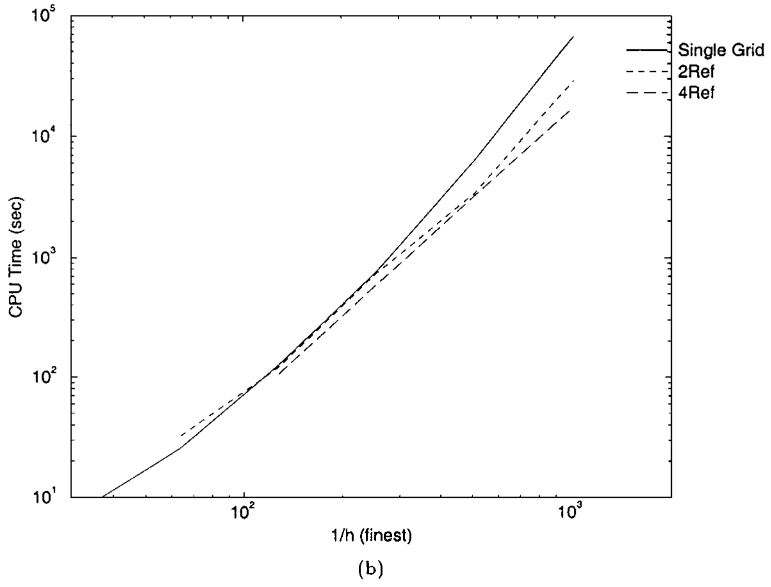


FIG. 20. Performance of the adaptive algorithm for the three-vortex problem. (a) Total cells advanced. (b) CPU time.

4.4. Adaptive Algorithm Performance

Almgren *et al.* [3] demonstrated that using a locally adaptive algorithm can produce substantial savings in the computational time and memory necessary to achieve a desired solution accuracy.

It should be noted that performance data for locally adaptive methods such as this one are highly problem dependent; results will depend on the fraction of the domain which is refined, and on the grid configurations chosen. It is felt by the authors that the example used here is representative of "typical" problems where adaptivity will be beneficial.

Figure 20 shows the total number of cells advanced and the total CPU time used for the three-vortex problem with a single uniform grid, one with $n_{\text{ref}} = 2$ levels of refinement and one with $n_{\text{ref}} = 4$ levels of refinement. Note that these results are indexed by the finest resolution of the solution, rather than that of the base grid, to represent the cost of achieving a given accuracy. Also worth noting is that as the size of the problem increases, the single-grid code becomes less efficient (CPU time per cell rises) owing to decreased efficiency as a result of worsened cache performance. While the benefits of adaptivity are marginal for smaller cases, they become more pronounced at higher resolutions.

To illustrate the effects of adaptivity on a single problem, Fig. 21 shows the CPU time and cell counts for three cases, each with an equivalent resolution of $h_{\text{finest}} = \frac{1}{1024}$. Results are normalized by the single-grid values. In this case, both the $n_{\text{ref}} = 2$ and $n_{\text{ref}} = 4$ cases show significant savings, both in memory and in CPU time. The difference between the two lines in the figure represents the overhead due to adaptivity, which comes primarily from the synchronization and intergrid transfer operations. It is expected that with some optimization,

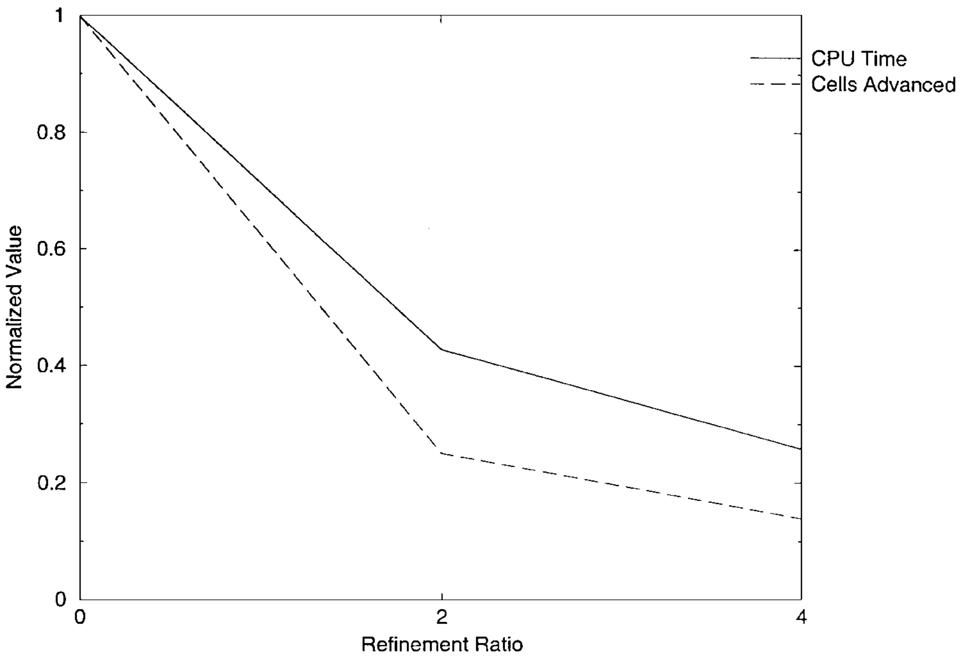


FIG. 21. Normalized timings and cell counts for adaptive code for equivalent 1024 × 1024 resolution. Cell counts and timings are normalized by the cost of the single-grid calculation.

this overhead could be substantially reduced, which would enable the execution time to better follow the cell counts.

5. CONCLUSIONS

We have presented an algorithm for computing solutions to the equations of incompressible flow on locally refined grids using a multilevel projection algorithm based on a cell-centered approximate projection. Use of a cell-centered discretization for the projection enables the use of only one set of cell-centered solvers to implement this algorithm, which will simplify the extension of this work to more complex situations and geometries. The algorithm refines in time as well as space and uses an approximate volume-discrepancy approach to correct for errors in freestream preservation at interfaces between coarse and fine grids.

We have demonstrated that the algorithm is second-order accurate in space and time and displays the same convergence properties as the uniform-grid algorithm upon which it is based. Also, we demonstrate that, with appropriate choice of local refinement, multilevel solutions computed with this algorithm can attain the accuracy of the equivalent uniform fine grid at less computational cost.

The approach to freestream preservation was shown to drastically reduce deviations from freestream conditions in advected quantities, both by reducing their magnitude and by confining freestream errors primarily to the set of coarse cells immediately adjacent to coarse–fine interfaces.

Although we include a composite synchronization projection to enforce the divergence constraint, we have been unable to find a case in which the synchronization projection is necessary for accuracy or stability. We believe that to be due to the projection formulation employed in this work, which appears to be more robust than the technique of projecting the velocity update. The natural conclusion is that the synchronization projection is unnecessary for this particular algorithm. However, it may still be necessary for problems with more complicated models, such as viscous flow, a topic which will be explored in future extensions of this algorithm.

APPENDIX: SINGLE-GRID ALGORITHM

The single-grid version of the algorithm advances the solution $\mathbf{u}$ and s from time t^n to time t^{n+1}. At time t^n we have the current solution $\mathbf{u}^n$ and s^n. Our discretization of (12) is

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t[(\mathbf{u} \cdot \nabla)\mathbf{u}]^{n+1/2} - \Delta t \nabla \pi^{n+1/2}, \tag{A.1}$$

where $\pi^{n+1/2}$ is our approximation of the pressure at time $t^n + \frac{1}{2}\Delta t$. The scalar advection equation (15) is discretized as

$$s^{n+1} = s^n - \Delta t[\nabla \cdot (\mathbf{u}s)]^{n+1/2}, \tag{A.2}$$

which is our evolution equation for s.

We use a predictor–corrector scheme based on that in [8] as extended in [9]. We first predict an approximation to the new velocity field, $\mathbf{u}^*$, which does not, in general, satisfy the divergence constraint (13). We then correct the velocity field by projecting $\mathbf{u}^*$ onto the space of vector fields which approximately satisfy the divergence constraint.

A.1. Computing Advection Velocities

First, we compute approximate edge-centered advection velocities $\mathbf{u}^{\text{edge}}$ by averaging the cell-centered $\mathbf{u}^n$ to edges:

$$\mathbf{u}^{\text{edge}} = Av^{\text{C}\to\text{E}}\mathbf{u}^n. \tag{A.3}$$

Next, we use a Taylor expansion to extrapolate normal velocities to cell edges at time $t^n + \Delta t/2$, using (12) to replace the time derivative. For the $(i+\frac{1}{2}, j)$ edges, this is

$$\begin{aligned} u_{i,j}^{\text{norm}} &= \frac{1}{2}\left(u_{i+1/2,j}^{\text{edge}} + u_{i-1/2,j}^{\text{edge}}\right) \\ \tilde{u}_{i+1/2,j}^{L,n+1/2} &= u_{i,j}^n + \min\left[\frac{1}{2}\left(1 - u_{i,j}^{\text{norm}}\frac{\Delta t}{h}\right), \frac{1}{2}\right](u_x)_{i,j} - \frac{\Delta t}{2h}(\overline{u}_y)_{i,j}, \end{aligned} \tag{A.4}$$

where u_x is the undivided centered-difference in the normal direction,

$$(u_x)_{i,j} = \frac{1}{2}\left(u_{i+1,j}^n - u_{i-1,j}^n\right), \tag{A.5}$$

and $\overline{u}_y$ is the undivided upwinded transverse difference,

$$\begin{aligned} v_{i,j}^{\tan} &= \frac{1}{2}\left(v_{i,j+1/2}^{\text{edge}} + v_{i,j-1/2}^{\text{edge}}\right) \\ (\overline{u}_y)_{i,j} &= \begin{cases} u_{i,j}^n - u_{i,j-1}^n & \text{if } v_{i,j}^{\tan} > 0, \\ u_{i,j+1}^n - u_{i,j}^n & \text{if } v_{i,j}^{\tan} < 0. \end{cases} \end{aligned} \tag{A.6}$$

Computing the right state is similar and we get

$$\tilde{u}_{i+1/2,j}^{R,n+1/2} = u_{i+1,j}^n + \max\left[\frac{1}{2}\left(-1 - u_{i,j}^{\text{norm}}\frac{\Delta t}{h}\right), -\frac{1}{2}\right](u_x)_{i+1,j} - \frac{\Delta t}{2h}(\overline{u}_y)_{i+1,j}. \tag{A.7}$$

Then, we choose the upwind state:

$$u_{i+1/2,j}^{n+1/2} = \begin{cases} \tilde{u}_{i+1/2,j}^{L,n+1/2} & \text{if } u_{i+1/2,j}^{\text{edge}} > 0, \\ \tilde{u}_{i+1/2,j}^{R,n+1/2} & \text{if } u_{i+1/2,j}^{\text{edge}} < 0, \\ \frac{1}{2}\left(\tilde{u}_{i+1/2,j}^{L,n+1/2} + \tilde{u}_{i+1/2,j}^{R,n+1/2}\right) & \text{if } u_{i+1/2,j}^{\text{edge}} = 0. \end{cases} \tag{A.8}$$

The pressure term is not included in the extrapolation because these velocities are projected with an edge-centered projection. Also, unlike previous implementations of similar algorithms [3, 8, 24] we do not employ slope limiters when computing u_x and u_y. Hilditch [19] found slope limiters, developed to prevent oscillations in compressible flows with sharp discontinuities, to be unnecessary for smooth, low Mach number flows.

Extrapolation of normal y-direction velocities is similar. Once we have computed an edge-centered normal velocity field, we apply an edge-centered projection to ensure that the advection velocities are divergence-free. First solve

$$L\phi = D\left(\mathbf{u}^{n+1/2}\right), \tag{A.9}$$

and then correct the velocity field,

$$\mathbf{u}^{\text{half}} = \mathbf{u}^{n+1/2} - G\phi. \tag{A.10}$$

This set of edge-centered advection velocities at time $t^{n+1/2}$ is used to compute the advective terms in (A.1). Note that we have only computed velocities normal to the faces, which would be $(u^{\text{half}}_{i+1/2,j}, v^{\text{half}}_{i,j+1/2})^T$.

A.2. Scalar Advection

First, we predict edge-centered upwinded values for $s^{n+1/2}$ in the same way as for the velocity predictor. As before, we compute values for $\tilde{s}^{L,n+1/2}$ and $\tilde{s}^{R,n+1/2}$, and then we choose the upwind value based on the local sign of $\mathbf{u}^{\text{half}}$:

$$\begin{aligned} u^{\text{norm}}_{i,j} &= \frac{1}{2}\left(u^{\text{half}}_{i+1/2,j} + u^{\text{half}}_{i-1/2,j}\right) \\ \tilde{s}^{L,n+1/2}_{i+1/2,j} &= s^n_{i,j} + \min\left[\frac{1}{2}\left(1 - u^{\text{norm}}_{i,j}\frac{\Delta t}{h}\right), \frac{1}{2}\right](s_x)_{i,j} - \frac{\Delta t}{2h} v^{\tan}_{i,j}(\overline{s}_y)_{i,j}. \end{aligned}$$

As before,

$$\begin{aligned} v^{\tan}_{i,j} &= \frac{1}{2}\left(v^{\text{half}}_{i,j+1/2} + v^{\text{half}}_{i,j-1/2}\right) \\ (s_x)_{i,j} &= \frac{1}{2}\left(s^n_{i+1,j} - s^n_{i-1,j}\right) \\ (\overline{s}_y)_{i,j} &= \begin{cases} s^n_{i,j} - s^n_{i,j-1} & \text{if } v^{\tan}_{i,j} > 0, \\ s^n_{i,j+1} - s^n_{i,j} & \text{if } v^{\tan}_{i,j} < 0. \end{cases} \end{aligned} \tag{A.11}$$

For the right state we have

$$\tilde{s}^{R,n+1/2}_{i+1/2,j} = s^n_{i+1,j} + \max\left[\frac{1}{2}\left(-1 - u^{\text{norm}}_{i,j}\frac{\Delta t}{h}\right), -\frac{1}{2}\right](s_x)_{i+1,j} - \frac{\Delta t}{2h} v^{\tan}_{i+1,j}(\overline{s}_y)_{i+1,j}.$$

Then, choose the upwind state:

$$s^{n+1/2}_{i+1/2,j} = \begin{cases} \tilde{s}^{L,n+1/2}_{i+1/2,j} & \text{if } u^{\text{edge}}_{i+1/2,j} > 0, \\ \tilde{s}^{R,n+1/2}_{i+1/2,j} & \text{if } u^{\text{edge}}_{i+1/2,j} < 0, \\ \frac{1}{2}\left(\tilde{s}^{L,n+1/2}_{i+1/2,j} + \tilde{s}^{R,n+1/2}_{i+1/2,j}\right) & \text{if } u^{\text{edge}}_{i+1/2,j} = 0. \end{cases} \tag{A.12}$$

Computation of $s^{n+1/2}_{i,j+1/2}$ on the y-edges is similar. Then, we compute the fluxes:

$$\begin{aligned} F^{s,x}_{i+1/2,j} &= u^{\text{half}}_{i+1/2,j} s^{n+1/2}_{i+1/2,j} \\ F^{s,y}_{i,j+1/2} &= v^{\text{half}}_{i,j+1/2} s^{n+1/2}_{i,j+1/2}. \end{aligned} \tag{A.13}$$

Finally, the updated state s^{n+1} can be computed using the discrete analog to (A.2):

$$s^{n+1}_{i,j} = s^n_{i,j} - \Delta t\left(\frac{F^{s,x}_{i+1/2,j} - F^{s,x}_{i-1/2,j}}{h} + \frac{F^{s,y}_{i,j+1/2} - F^{s,y}_{i,j-1/2}}{h}\right). \tag{A.14}$$

A.3. Velocity Predictor

Using $\mathbf{u}^{\text{half}}$, we compute an approximation of the advection term $[(\mathbf{u}\cdot\nabla)\mathbf{u}]^{n+1/2}$. Although the advection velocities in this algorithm are discretely divergence-free, which would allow conservative differencing to be used to compute the advection terms, we instead use convective differencing. In the multilevel case, the advection velocities are not generally discretely divergence-free, owing to the effects of the freestream preservation correction.

First, we re-predict edge-centered velocities as in Section A.1, this time using $\mathbf{u}^{\text{half}}$ rather than $Av^{\text{C}\to\text{E}}(\mathbf{u}^n)$, which was used in Section A.1. We re-use the already-computed normal velocities $\mathbf{u}^{\text{half}}$ as predicted velocities. So, we only compute the tangential edge-centered predicted velocities. To compute $v^{\text{half}}_{i+1/2,j}$, for example, we extrapolate in the same way as for $u^{\text{half}}_{i+1/2,j}$, in this case including $G\phi$ to represent the effects of the edge-centered projection. We obtain

$$\begin{aligned} u^{\text{norm}}_{i,j} &= \frac{1}{2}\left(u^{\text{half}}_{i+1/2,j} + u^{\text{half}}_{i-1/2,j}\right) \\ \tilde{v}^{L,n+1/2}_{i+1/2,j} &= v^n_{i,j} + \min\left[\frac{1}{2}\left(1 - u^{\text{norm}}_{i,j}\frac{\Delta t}{h}\right), \frac{1}{2}\right](v_x)_{i,j} - \frac{\Delta t}{2h} v^{\text{tan}}_{i,j}(\overline{v}_y)_{i,j}, \end{aligned} \tag{A.15}$$

where

$$(v_x)_{i,j} = \frac{1}{2}\left(v^n_{i+1,j} - v^n_{i-1,j}\right) \tag{A.16}$$

and

$$\begin{aligned} v^{\text{tan}}_{i,j} &= \frac{1}{2}\left(v^{\text{half}}_{i,j+1/2} + v^{\text{half}}_{i,j-1/2}\right) \\ (\overline{v}_y)_{i,j} &= \begin{cases} v^n_{i,j} - v^n_{i,j-1} & \text{if } v^{\text{tan}}_{i,j} > 0, \\ v^n_{i,j+1} - v^n_{i,j} & \text{if } v^{\text{tan}}_{i,j} < 0. \end{cases} \end{aligned} \tag{A.17}$$

For the “right” state, we have

$$\tilde{v}^{R,n+1/2}_{i+1/2,j} = v^n_{i+1,j} + \max\left[\frac{1}{2}\left(-1 - u^{\text{norm}}_{i,j}\frac{\Delta t}{h}\right), -\frac{1}{2}\right](v_x)_{i+1,j} - \frac{\Delta t}{2h} v^{\text{tan}}_{i+1,j}(\overline{v}_y)_{i+1,j}. \tag{A.18}$$

Then we choose the upwind state:

$$v^{n+1/2}_{i+1/2,j} = \begin{cases} \tilde{v}^{L,n+1/2}_{i+1/2,j} & \text{if } u^{\text{half}}_{i+1/2,j} > 0, \\ \tilde{v}^{R,n+1/2}_{i+1/2,j} & \text{if } u^{\text{half}}_{i+1/2,j} < 0, \\ \frac{1}{2}\left(\tilde{v}^{L,n+1/2}_{i+1/2,j} + \tilde{v}^{R,n+1/2}_{i+1/2,j}\right) & \text{if } u^{\text{half}}_{i+1/2,j} = 0. \end{cases} \tag{A.19}$$

Finally, we include the pressure gradient:

$$\begin{aligned} v^{\text{half}}_{i+1/2,j} &= v^{\text{half}}_{i+1/2,j} - (G\phi)_{i+1/2,j} \\ &= v^{\text{half}}_{i+1/2,j} - \frac{\phi_{i+1,j+1} + \phi_{i-1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j-1}}{4h}. \end{aligned} \tag{A.20}$$

To compute the advective terms, first compute a cell-centered advection velocity:

$$\mathbf{u}^{\mathrm{AD-CC}} = Av^{\mathrm{E}\rightarrow\mathrm{C}}\mathbf{u}^{\mathrm{AD}}.$$

Then,

$$\begin{aligned} [(\mathbf{u}\cdot\nabla)u]_{i,j}^{n+1/2} &= u_{i,j}^{\mathrm{AD-CC}}\frac{\left(u_{i+1/2,j}^{\mathrm{half}} - u_{i-1/2,j}^{\mathrm{half}}\right)}{h} + v_{i,j}^{\mathrm{AD-CC}}\frac{\left(u_{i,j+1/2}^{\mathrm{half}} - u_{i,j-1/2}^{\mathrm{half}}\right)}{h} \\ [(\mathbf{u}\cdot\nabla)v]_{i,j}^{n+1/2} &= u_{i,j}^{\mathrm{AD-CC}}\frac{\left(v_{i+1/2,j}^{\mathrm{half}} - v_{i-1/2,j}^{\mathrm{half}}\right)}{h} + v_{i,j}^{\mathrm{AD-CC}}\frac{\left(v_{i,j+1/2}^{\mathrm{half}} - v_{i,j-1/2}^{\mathrm{half}}\right)}{h}. \end{aligned} \tag{A.21}$$

Finally, we compute $\mathbf{u}^*$:

$$\begin{aligned} u_{i,j}^* &= u_{i,j}^n - \Delta t[(\mathbf{u}\cdot\nabla)u]_{i,j}^{n+1/2} \\ v_{i,j}^* &= v_{i,j}^n - \Delta t[(\mathbf{u}\cdot\nabla)v]_{i,j}^{n+1/2}. \end{aligned} \tag{A.22}$$

A.4. Projection

First, solve

$$L\pi^{n+1/2} = \frac{1}{\Delta t}D^{\mathrm{CC}}(\mathbf{u}^*). \tag{A.23}$$

Then, correct the velocity field onto the space of vector fields which satisfy the divergence constraint:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t G^{\mathrm{CC}}\pi^{n+1/2}. \tag{A.24}$$

ACKNOWLEDGMENTS

The authors thank Ann Almgren for her invaluable advice and suggestions about the algorithm. We also thank Matthew Bettencourt and Richard Propp for their help in developing and testing the elliptic solvers used in this work.

REFERENCES

1. G. Acs, S. Doleschall, and E. Farkas, General purpose compositional model, *SPEJ Soc. Pet. Eng. J.* **25**, 543 (1985).
2. A. Almgren, J. Bell, and W. Szymczak, A numerical method for the incompressible Navier–Stokes equations based on an approximate projection, *SIAM J. Sci. Comput.* **17**, 358 (1996).
3. A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome, A conservative adaptive projection method for the variable density incompressible Navier–Stokes equations, *J. Comput. Phys.* **142**, 1 (1998).
4. A. S. Almgren, J. B. Bell, P. Colella, and T. Marthaler, A cell-centered Cartesian grid projection method for the incompressible euler equations in complex geometries, in *Proceedings of the AIAA 12th Computational Fluid Dynamics Conference, San Diego, CA, June 19–22, 1995.*
5. A. S. Almgren, J. B. Bell, P. Colella, and T. Marthaler, A Cartesian grid projection method for the incompressible Euler equations in complex geometries, *SIAM J. Sci. Comput.* **18**(5), 1289 (1997).
6. A. Almgren, T. Buttke, and P. Colella, A fast adaptive vortex method in three dimensions, *J. Comput. Phys.* **113**, 177 (1994).

7. A. S. Almgren, J. B. Bell, and W. Y. Crutchfield, *Approximate Projection Methods: Part 1. Inviscid Analysis*, Technical Report LBNL-43374 (Lawrence Berkeley National Laboratory, 1999).
8. J. B. Bell, P. Colella, and H. M. Glaz, An efficient second-order projection method for the incompressible Navier–Stokes equations, *J. Comput. Phys.* **85**, 257 (1989).
9. J. B. Bell, P. Colella, and L. H. Howell, An efficient second-order projection method for viscous incompressible flow, in *Proceedings, AIAA 10th Computational Fluid Dynamics, Honolulu, June 1991*, p. 360.
10. M. J. Berger and P. Colella, Local adaptive mesh refinement for shock hydrodynamics, *J. Comput. Phys.* **82**(1), 64 (1989).
11. M. J. Berger and J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, *J. Comput. Phys.* **53**, 484 (1984).
12. M. J. Berger and I. Rigoutsos, An algorithm for point clustering and grid generation, *IEEE Trans. Systems, Man Cybernet.* **21**(5), 1278 (1991).
13. M. T. Bettencourt, *A Block Structured Adaptive Steady-State Solver for the Drift Diffusion Equations*, Ph.D. thesis (Univ. Calif. Berkeley, 1998).
14. D. L. Brown and M. L. Minion, Performance of under-resolved two-dimensional incompressible flow simulations, *J. Comput. Phys.* **122**(1), 165 (1995).
15. A. J. Chorin, Numerical solutions of the Navier–Stokes equations, *Math. Comput.* **22**, 745 (1968).
16. P. Colella, Multidimensional upwind methods for hyperbolic conservation laws, *J. Comput. Phys.* **87**(1), 171 (1990).
17. M. S. Day, P. Colella, M. Lijewski, C. Rendleman, and D. L. Marcus, *Embedded Boundary Algorithms for Solving Poisson's Equation on Complex Domains*, Technical Report LBNL-41811, Lawrence Berkeley National Laboratory, 1998.
18. F. H. Harlow and J. E. Welch, Numerical calculation of time-dependent viscous incompressible flow of fluids with free surfaces, *Phys. Fluids* **8**(12), 2182 (1965).
19. J. Hilditch, *A Projection Method for Low Mach Number Reacting Flow in the Fast Chemistry Limit*, Ph.D. thesis (Univ. Calif. Berkeley, 1997).
20. R. D. Hornung and J. A. Trangenstein, Adaptive mesh refinement and multilevel iteration for flow in porous media, *J. Comput. Phys.* **136**, 522 (1997).
21. L. H. Howell and J. B. Bell, An adaptive mesh projection method for viscous incompressible flow, *SIAM J. Sci. Comput.* **18**(4), 996 (1997).
22. H. Johansen and P. Colella, A Cartesian grid embedded boundary method for Poisson's equation on irregular domains, *J. Comput. Phys.* **147**, 60 (1998).
23. M. F. Lai, J. Bell, and P. Colella, A projection method for combustion in the zero Mach number limit, in *Proceedings of the Eleventh AIAA Computational Fluid Dynamics Conference* (AIAA, June 1993), p. 776.
24. M. Lai, *An Approximate Projection Method for Reacting Flow in the Zero Mach Number Limit*, Ph.D. thesis, (Univ. Calif. Berkeley, 1994).
25. D. F. Martin and K. L. Cartwright, *Solving Poisson's Equation Using Adaptive Mesh Refinement*, Technical Report UCB/ERL M96/66 (Univ. Calif. Berkeley, 1996).
26. D. Martin, *An Adaptive Cell-Centered Projection Method for the Incompressible Euler Equations*, Ph.D. thesis (Univ. Calif. Berkeley, 1998).
27. M. Minion, *Two Methods for the Study of Vortex Patch Evolution on Locally Refined Grids*, Ph.D. thesis (Univ. Calif. Berkeley, 1994).
28. M. L. Minion, A projection method for locally refined grids, *J. Comput. Phys.* **127**(1), 158 (1996).
29. R. B. Pember, J. B. Bell, P. Colella, W. Y. Crutchfield, and M. L. Welcome, Cartesian grid method for unsteady compressible flow in irregular regions, *J. Comput. Phys.* **120**, 278 (1995).
30. R. Propp and P. Colella, An adaptive mesh refinement algorithm for porous media flows, submitted for publication; also LBNL Technical Report LBNL-45143.
31. R. Propp, *Numerical Modeling of a Trickle Bed Reactor*, Ph.D. thesis (Univ. Calif. Berkeley, 1998).
32. W. J. Rider, *Approximate Projection Methods for Incompressible Flow: Implementation, Variants, and Robustness*, Technical Report LA-UR-2000 (Los Alamos National Laboratory, 1994).

33. W. J. Rider, *The Robust Formulation of Approximate Projection Methods for Incompressible Flows*, Technical Report LA-UR-3015 (Los Alamos National Laboratory, 1994).

34. M. C. Thompson and J. H. Ferziger, An adaptive multigrid technique for the incompressible Navier–Stokes equations, *J. Comput. Phys.* **82**, 94 (1989).

35. J. A. Trangenstein and J. B. Bell, Mathematical structure of the black-oil model for petroleum reservoir simulation, *SIAM J. Appl. Math.* **49**(3), 749 (1989).

36. Y. Zang, R. L. Street, and J. R. Koseff, A non-staggered grid, fractional step method for time-dependent inompressible Navier–Stokes equations in curvilinear coordinates, *J. Comput. Phys.* **114**, 18 (1994).