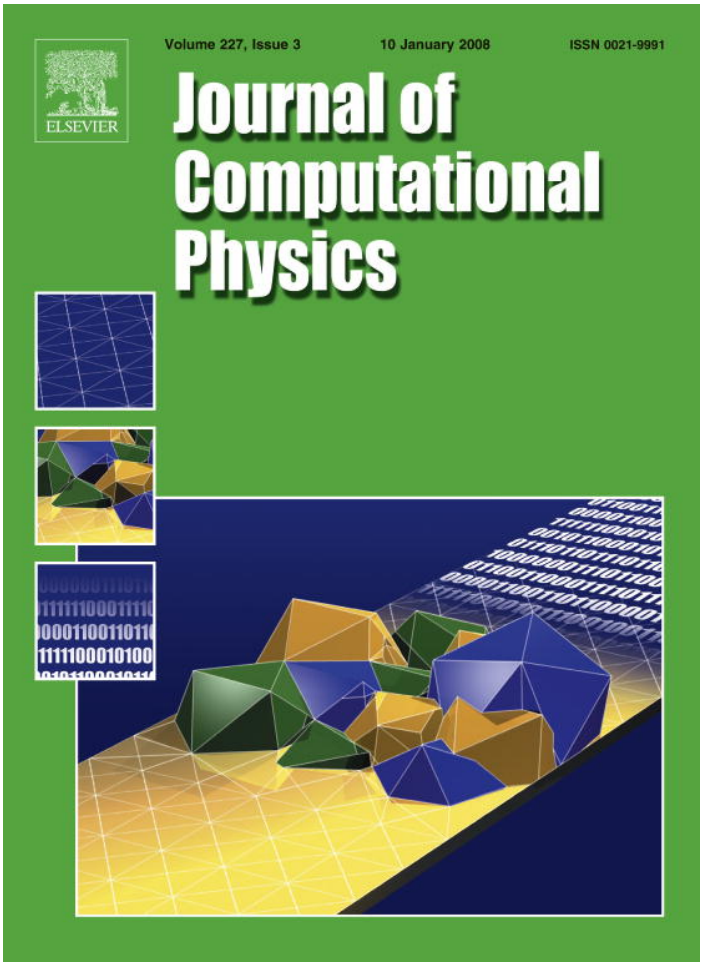

Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 227 (2008) 1863–1886
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

A cell-centered adaptive projection method for the incompressible Navier–Stokes equations in three dimensions

Daniel F. Martin *, Phillip Colella, Daniel Graves

Lawrence Berkeley National Laboratory, Berkeley, CA, United States

Received 20 November 2006; received in revised form 26 September 2007; accepted 29 September 2007
Available online 25 October 2007

Abstract

We present a method for computing incompressible viscous flows in three dimensions using block-structured local refinement in both space and time. This method uses a projection formulation based on a cell-centered approximate projection, combined with the systematic use of multilevel elliptic solvers to compute increments in the solution generated at boundaries between refinement levels due to refinement in time. We use an L_0-stable second-order semi-implicit scheme to evaluate the viscous terms. Results are presented to demonstrate the accuracy and effectiveness of this approach.
Published by Elsevier Inc.

Keywords: Adaptive mesh refinement; Incompressible flow; Projection methods

1. Introduction

Adaptive mesh refinement is a powerful tool for computing solutions to problems which are otherwise inaccessible due to limits in computational resources. In a previous work [19], we presented a projection method for two-dimensional inviscid incompressible flow on adaptive locally refined meshes. The algorithm in [19] employs refinement in time as well as space (subcycling), and is second-order accurate in time and space. The cell-centered projection discretization is based on composite (multilevel) and single-level operators. Also, an advection-velocity correction was computed based on a passively advected scalar to ensure that the algorithm was approximately freestream-preserving.

In this work, the algorithm presented in [19] is extended to three-dimensional viscous flow. There have been many approaches to computing adaptive solutions for this problem which have not employed subcycling in time [9,22,23,29], instead opting to advance all levels with a uniform timestep. Extension of the non-subcycled schemes to multiphase flows was done in [28], and to the immersed-boundary method in [15,25]. Almgren et al. [1] presented an algorithm for the solution of the three-dimensional incompressible Navier–Stokes equations

* Corresponding author. Tel.: +1 510 495 2852; fax: +1 510 495 2505.
E-mail address: DFMartin@lbl.gov (D.F. Martin).

0021-9991/$ - see front matter Published by Elsevier Inc.
doi:10.1016/j.jcp.2007.09.032

which also subcycles in time. As detailed in [1], subcycling results in better accuracy and AMR performance, at the expense of greater algorithmic complexity.

The work presented here represents a different set of algorithmic design choices from those employed in [1], many of which have been chosen to simplify the eventual extension of this work to Cartesian-mesh embedded-boundary geometries like those in [11]. Features of the algorithm presented here include:

- *Projection discretization*. This work employs the cell-centered approximate projection discretization developed in [19] as opposed to the node-centered discretization employed in [1]. Since cell-centered solvers are already required for other parts of the algorithm, the cell-centered projection discretization enables the use of a single set of elliptic solvers and will substantially lessen the work required to extend this algorithm to embedded boundary computations and other applications.
- *Treatment of viscous terms*. Previous semi-implicit methods have used the Crank–Nicolson scheme to compute the viscous terms in the update. However, for the discretizations used in this work, we found the neutrally-stable Crank–Nicolson scheme suffered from weak instabilities at coarse–fine interfaces, similar to the behavior noticed at embedded boundaries in [16,20]. To eliminate this problem, we employ a second-order semi-implicit Runge–Kutta scheme based on the L_0-stable scheme in [30].
- *Approach to synchronization*. Synchronizing the computed solution between AMR levels for an adaptive projection method requires additional elliptic solves to ensure that the divergence constraint is satisfied. Also, a flux-correction step is performed to ensure conservation at coarse–fine interfaces. For stability, this correction is computed in an implicit manner, requiring an additional elliptic solve during the synchronization step, as in [13]. Extending the ideas in [19], our approach has been to perform these as multilevel elliptic solves over all of the appropriate refinement levels. In contrast, the work in [1] performs synchronizations one pair of levels at a time using single-level elliptic solves, interpolating corrections to finer levels.
- *Re-initialization after regridding*. In an AMR computation, refined regions can change as the solution evolves. After the mesh hierarchy is changed, the solution must be re-initialized. Once data is interpolated to fill newly-refined regions, the velocity field is projected to ensure that the new velocity field is divergence-free. Also, pressures and the freestream preservation correction are recomputed. Previous work [1] has simply interpolated these values from coarser levels where needed.
- *Freestream preservation for advective transport*. We use the volume discrepancy approach described in [19] to ensure that freestream preservation is approximately enforced. In contrast, freestream preservation is maintained exactly in [1] by computing a correction to the advection velocity field, performing a correction advection step, and then interpolating the corrections to finer levels.

A basic principle in the design of this algorithm, as seen in the last three points above, has been to avoid the interpolation of corrections computed by elliptic and parabolic solves onto finer levels. While such interpolation can easily be done to second-order accuracy, the gradients of such interpolated corrections can be non-smooth with mesh imprinting and interpolation artifacts. Instead, we perform multilevel solves to compute corrections over entire multilevel hierarchies, which results in smooth corrections with smooth gradients.

1.1. Formulation of the problem

We are solving the incompressible constant-density Navier–Stokes equations with a passively-advected scalar Λ, included as an auxiliary quantity for freestream preservation as in [19]:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u}\cdot\nabla)\vec{u} = -\nabla p + \nu\Delta\vec{u}, \tag{1}$$

$$(\nabla\cdot\vec{u}) = 0, \tag{2}$$

$$\vec{u} = 0 \quad \text{on } \partial\Omega, \tag{3}$$

$$\frac{\partial \Lambda}{\partial t} + \nabla\cdot(\vec{u}\Lambda) = 0, \tag{4}$$

where $\vec{u}$ is the fluid velocity, p is the pressure, and ν is the kinematic viscosity.

As in [19], we transform the constrained problem (1, 2) into an initial value problem using the Hodge projection. The Hodge projection operator $\mathbb{P}$ applied to a vector field $\vec{w}$ extracts the divergence-free component

$$\mathbb{P}(\vec{w}) = \vec{w}_d$$
$$\nabla \cdot \vec{w}_d = 0$$

and is computed by solving an elliptic equation and then subtracting the gradient piece of the decomposition:

$$\mathbb{P}(\vec{w}) = (I - \nabla(\Delta^{-1})\nabla\cdot)\vec{w}.$$

Using the projection operator, we transform the constrained problem (1) and (2) into an evolution equation:

$$\frac{\partial \vec{u}}{\partial t} = \mathbb{P}(-(\vec{u}\cdot\nabla)\vec{u} + \nu\Delta\vec{u}),$$
$$\nabla \cdot \vec{u}(\cdot, t=0) = 0.$$

1.2. Single-level time discretization

On a single level of refinement, our algorithm is a predictor–corrector formulation of the projection method [10] of a form first introduced by Bell, Colella and Glaz (BCG) [7]. We first compute an intermediate velocity field and then project it onto the space of vectors which satisfy the divergence constraint. The intermediate velocity field $\vec{u}^*$ is computed as an approximation to $\vec{u}(t+\Delta t)$:

$$\vec{u}^* = \vec{u}(t) - \Delta t[(\vec{u}\cdot\nabla)\vec{u}]^{\mathrm{H}} + \Delta t L^{\mathrm{H}}_{\mathrm{visc}} - \Delta t \nabla p\left(t - \frac{1}{2}\Delta t\right),$$
$$L^{\mathrm{H}}_{\mathrm{visc}} = \nu\Delta\vec{u}\left(t + \frac{1}{2}\Delta t\right),$$

where the superscript H indicates centering at the half time $\left(t + \frac{1}{2}\Delta t\right)$. The updated velocity field is then computed by projecting the intermediate velocity field onto the space of divergence-free vectors:

$$\vec{u}(t+\Delta t) = \mathbb{P}\left(\vec{u}^* + \Delta t\nabla p\left(t - \frac{1}{2}\Delta t\right)\right),$$
$$\nabla p\left(t + \frac{1}{2}\Delta t\right) = \frac{1}{\Delta t}(\mathbb{I} - \mathbb{P})\left(\vec{u}^* + \Delta t\nabla p\left(t - \frac{1}{2}\Delta t\right)\right).$$

Note that we project an approximation to $\vec{u} + \nabla p$ rather than $\vec{u}$. We have found this formulation to be better behaved in the presence of local refinement; work by Almgren, Bell, and Crutchfield [3] supports this choice of formulations. Updates to the scalar Λ are computed using a conservative unsplit Godunov method [12,26].

In the original BCG algorithm, the Crank–Nicolson scheme is used to compute $L^{\mathrm{H}}_{\mathrm{visc}}$. However, Crank–Nicolson is only neutrally stable, and we found that it led to weak instabilities at coarse–fine interfaces, given the other choices we made in this algorithm. This behavior was similar to that noted at embedded boundaries in [16,20]. To eliminate this problem, we found it necessary to employ a different approach to computing the viscous terms in (1), using the L_0 scheme described in [30].

We consider a parabolic equation of the form

$$\frac{\partial q}{\partial t} = L(q) + f, \tag{5}$$

where L is a second-order linear elliptic operator. Following [30], we discretize (5) in time:

$$q^{n+1} = (I - \mu_1 L)^{-1}(I - \mu_2 L)^{-1}\left[(I + \mu_3 L)q^n + \Delta t(I + \mu_4 L)f^{n+\frac{1}{2}}\right], \tag{6}$$

where $q^n = q(n\Delta t)$, $f^{n+\frac{1}{2}} = f\left(\left(n + \frac{1}{2}\right)\Delta t\right)$, and the coefficients μ_1, μ_2, μ_3, μ_4 are the values suggested in [30]:

$$\mu_1 = \frac{2a - 1}{a + \mathrm{discr}}\Delta t,$$

$$\mu_2 = \frac{2a-1}{a-\text{discr}}\Delta t,$$
$$\mu_3 = (1-a)\Delta t,$$
$$\mu_4 = \left(\frac{1}{2}-a\right)\Delta t,$$
$$a = 2-\sqrt{2}-\epsilon,$$
$$\text{discr} = \sqrt{a^2-4a+2},$$

where ϵ is a small quantity (we use 10^{-8}). The treatment of the source term f presented here differs from that in [30] due to differences in time centering; the source terms in [30] are centered at the old and new times, while $f^{n+\frac{1}{2}}$ in this work is centered at the half-time. We use this to define the operator $L^{\text{TGA}}\left(q^n, f^{n+\frac{1}{2}}\right)$ as follows:

$$L^{\text{TGA}}\left(q^n, f^{n+\frac{1}{2}}\right) \equiv \frac{q^{n+1}-q^n}{\Delta t} - f^{n+\frac{1}{2}} \approx (Lq)\left(\left(n+\frac{1}{2}\right)\Delta t\right) + \mathrm{O}(\Delta t^2). \tag{7}$$

where $q^{n+1} = q^{n+1}\left(q^n, f^{n+\frac{1}{2}}\right)$ is defined to be the expression (6). In the present work, we take

$$\begin{aligned} L^{\text{H}}_{\text{visc}} &\equiv L^{\text{TGA}}_{\text{visc}}\left(\vec{u}^n, \vec{f}^{n+\frac{1}{2}}\right), \\ \vec{f}^{n+\frac{1}{2}} &= -[(\vec{u}\cdot\nabla)\vec{u}]^{\text{H}} - \nabla p\left(t-\frac{1}{2}\Delta t\right). \end{aligned} \tag{8}$$

1.3. AMR notation

In this work, we use the same notation as in [19]. Following [8], our adaptive mesh calculations are performed 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 Cartesian mesh Γ^ℓ with grid spacing h_ℓ. Level 0 is the coarsest level, while each level $\ell+1$ is a factor $n^\ell_{\text{ref}} = \frac{h_\ell}{h_{\ell+1}}$ finer than level ℓ; the refinement ratio n^ℓ_{ref} is an integer. Because refined grids overlay 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 $\mathcal{C}_r(i,j,k) = \left(\left\lfloor\frac{i}{r}\right\rfloor, \left\lfloor\frac{j}{r}\right\rfloor, \left\lfloor\frac{k}{r}\right\rfloor\right)$. In that case, $\{\mathcal{C}_r\}^{-1}\{(i,j,k)\}$ 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,k).

In the present work, we assume that the problem domain is a rectangle, and that the refinement ratios are powers of two. 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

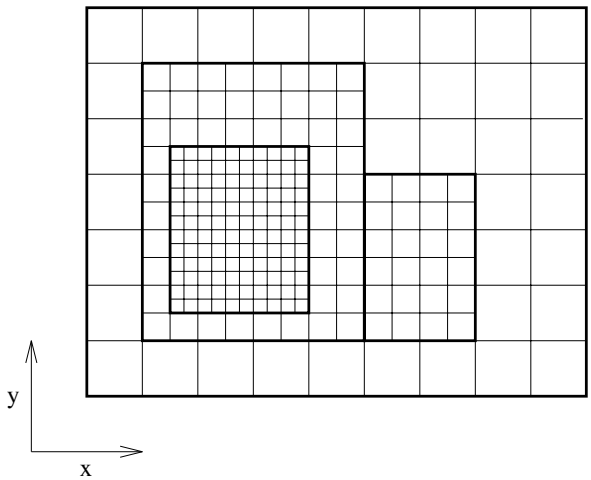

Fig. 1. Block-structured local refinement. Note that refinement is by an integer factor and is organized into rectangular patches.

implementation to simplify computations on the hierarchy of meshes. On the coarsest level, $\Omega^0 = \Gamma^0$. A cell on a level is either completely covered by cells at the next finer level, or it 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 invalid region is made up of cells which are covered by refined regions. The grids on each level satisfy a *proper nesting* condition [8]: 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 faces of level ℓ cells, while $\Omega^{\ell,*}_{\text{valid}}$ refers to the cell faces on level ℓ not covered by level $\ell+1$ faces. 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 faces: $\mathcal{C}_{n^\ell_{\text{ref}}}(\Omega^{\ell+1,*})$ is the set of level ℓ faces overlain by level $\ell+1$ faces.

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 faces. Like other face-centered variables, a composite vector field $\vec{u}^{\text{face,comp}}$ is valid on all faces not overlain by finer faces (Fig. 2). Likewise, we define composite and level operators which operate on composite and level variables, respectively.

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

1.3.1. Divergence, flux registers, and reflux-divergence

The operator discretizations employed in this work are identical to those employed in [19]; a short description of the operators is included here for convenience.

The basic multilevel divergence D^{comp} is a cell-centered divergence of a face-centered vector field. The level-operator divergence D^ℓ of a level variable $\vec{u}^{\text{face},\ell}$ is defined by ignoring any finer levels and computing D^ℓ everywhere in Ω^ℓ as if there were no finer level. Since the composite divergence on level ℓ depends on both level ℓ and level $\ell+1$ data [19], it may be written as $D^{\text{comp},\ell}(\vec{u}^{\text{face},\ell}, \vec{u}^{\text{face},\ell+1})$; the level operator only depends on level ℓ data: $D^\ell(\vec{u}^{\text{face},\ell})$.

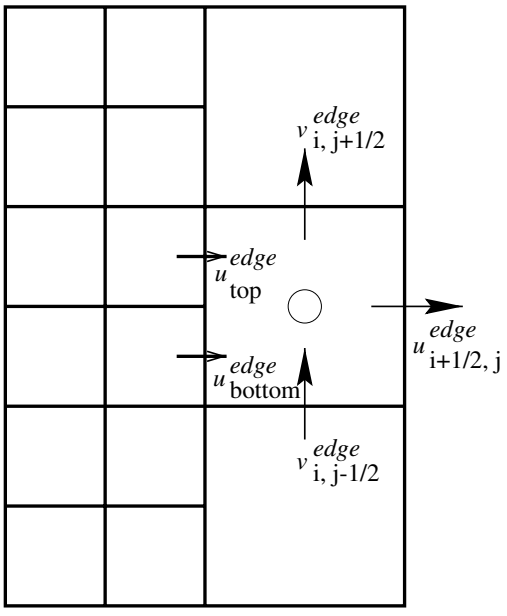

Fig. 2. Sample coarse–fine interface with a face-centered vector field. Cell (i,j) (open circle) is to the right of the coarse–fine interface.

Assume that the vector field $\vec{u}^{\text{face},\ell}$ can be extended to all faces in $\Omega^{\ell,*}$, including those covered by the coarse–fine interface face $\partial\Omega^{\ell+1,*}$. The composite divergence $D^{\text{comp}}\vec{u}^{\text{face,comp}}$ 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* $\delta\vec{u}^{\ell+1}$ defined on $\mathcal{C}_{n^{\ell}_{\text{ref}}}(\partial\Omega^{\ell+1,*})$, which stores the difference in the face-centered quantity $\vec{u}^{\text{face}}$ on the coarse–fine interface between levels ℓ and $\ell+1$. Notationally, $\delta\vec{u}^{\ell+1}$ 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 face-centered vectors on the coarse–fine interface with level $\ell+1$; the general composite operator can then be expressed as:

$$\begin{aligned}(D^{\text{comp},\ell}\vec{u}^{\text{face}})_{ijk} &= (D^{\ell}\vec{u}^{\text{face},\ell})_{ijk} + D^{\ell}_{\text{R}}(\delta\vec{u}^{\ell+1})_{ijk},\\ \delta\vec{u}^{\ell+1} &= \langle\vec{u}^{\text{face},\ell+1}\rangle - \vec{u}^{\text{face},\ell} \quad \text{on } \mathcal{C}_{n^{\ell}_{\text{ref}}}(\partial\Omega^{\ell+1,*}).\end{aligned} \tag{9}$$

For the level ℓ cell (ijk), D^{ℓ}_{R} can be defined as:

$$D^{\ell}_{\text{R}}(\delta\vec{u}^{\ell+1})_{ijk} = \frac{1}{h_{\ell}}\sum_{p}\pm(\delta\vec{u}^{\ell})_{\text{p}}, \tag{10}$$

where the sum is over the set of all faces of cell (ijk) which are also coarse–fine interfaces with level $\ell+1$, and the $\pm$ is $+$ if the face p is on the high side of cell (ijk), and $-$ if p is on the low side. Note that D^{ℓ}_{R} only affects the set of level ℓ cells immediately adjacent to the coarse–fine interface with level $\ell+1$.

1.3.2. Gradient and coarse–fine interpolation

The gradient is a face-centered, centered-difference gradient of a cell-centered variable ϕ. $G^{\text{comp}}\phi$ is a composite vector field, defined on all valid faces in the multilevel domain. To compute $G^{\text{comp}}\phi$ at a coarse–fine interface, we interpolate values for ϕ using both coarse- and fine-level values. The details of this interpolation process are discussed in the Appendix. We denote this quadratic coarse–fine interpolation operator as $I(\phi^{\ell},\phi^{\ell-1})$:

$$\phi^{\ell} = I(\phi^{\ell},\phi^{\ell-1}) \quad \text{on } \partial\Omega^{\ell,*} \tag{11}$$

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

The level-operator gradient G^{ℓ} is defined by extending G^{comp} (which is only defined on $\Omega^{\ell,*}_{\text{valid}}$) to all faces in $\Omega^{\ell,*}$ as if no finer level existed. At interfaces with a coarser level $\ell-1$, the interpolation operator $I(\phi^{\ell},\phi^{\ell-1})$ is used to compute ghost cell values.

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 $G^{\ell}(\phi^{\ell},\phi^{\ell-1})$.

1.3.3. Laplacian

The Laplacian is defined as the divergence of the gradient:

$$L^{\text{comp}}\phi^{\text{comp}} = D^{\text{comp}}G^{\text{comp}}\phi^{\text{comp}}, \tag{12}$$

$$L^{\ell}\phi^{\ell} = D^{\ell}G^{\ell}\phi^{\ell}. \tag{13}$$

Away from coarse–fine interfaces and domain boundaries, (12) and (13) reduce to the usual seven-point (five-point in 2D) second-order discrete Laplacian. The dependencies of the Laplacian operators may again be expressed explicitly: $L^{\text{comp},\ell}(\phi^{\ell},\phi^{\ell+1},\phi^{\ell-1})$ and $L^{\ell}(\phi^{\ell},\phi^{\ell-1})$.

1.3.4. Cell-centered operators

Cell-centered versions of the gradient and divergence operators are defined in the same way as in [19] using the operators defined above along with cell-to-face and face-to-cell averaging $\text{Av}^{\text{C}\to\text{F}}$ and $\text{Av}^{\text{F}\to\text{C}}$. For example, the composite cell-centered divergence operator is defined as the composite divergence applied to a cell-centered vector field which has been averaged from cells to faces:

$$D^{\mathrm{CC,comp}}\vec{u}^{\mathrm{CC}} = D^{\mathrm{comp}}(\mathrm{Av}^{\mathrm{C}\to\mathrm{F}}\vec{u}^{\mathrm{CC}}).$$

Similarly, the cell-centered gradient is defined by averaging the face-centered gradient to cell centers:

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

The cell-centered level-operator divergence and gradient operators are defined similarly:

$$D^{\mathrm{CC},\ell}\vec{u}^{\mathrm{CC}} = D^{\ell}(\mathrm{Av}^{\mathrm{C}\to\mathrm{F}}\vec{u}^{\mathrm{CC}}),$$

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

2. Multilevel update algorithm

In this section, we present the recursive algorithm used to update the solution on a single level ℓ from time t^ℓ to time $t^\ell + \Delta t^\ell$. Implied in this advance is the update of all levels finer than ℓ and synchronization with them.

As in [8,19], we organize our update around single-level updates and then a synchronization step to ensure proper matching between the solutions at different refinement levels. This can be described as a recursive advance for a single AMR level ℓ which advances the solution at levels ℓ and finer from time t^ℓ to $t^\ell + \Delta t^\ell$. First, the solution on level ℓ is advanced using a single-level update from time t^ℓ to $t^\ell + \Delta t^\ell$. This update is generally performed using single-level operators without regard for the solution at finer levels. Once the single-level update has been completed, the next finer level $\ell + 1$ is advanced n^ℓ_{ref} times with a timestep $\Delta t^{\ell+1} = \frac{\Delta t^\ell}{n^\ell_{\mathrm{ref}}}$. Once level $\ell + 1$ (along with any levels finer than $\ell + 1$) has been advanced to the time $t^\ell + \Delta t^\ell$, a *synchronization* step is performed to ensure proper matching between the solutions at different refinement levels.

2.1. Variables

We start the level ℓ advance with the solution at time t^ℓ, which includes the cell-centered velocity field $\vec{u}^\ell = (u^\ell, v^\ell, w^\ell)^{\mathrm{T}}$, the cell-centered freestream preservation scalar Λ^ℓ, and the face-centered freestream preservation correction $\vec{u}_{\mathrm{p}}$ from the most recent synchronization step, which has been extended to the invalid regions on level ℓ with $\langle\vec{u}^{\ell+1}_{\mathrm{p}}\rangle$. We also have the lagged cell-centered approximation to the pressure $\pi^{\ell,n-\frac{1}{2}}$. (Following [19], we have decomposed the pressure into $p = \pi + e$, where $\pi^{\ell,n-\frac{1}{2}} \approx p^\ell\left(t^{n-\frac{1}{2}}\right)$ is the pressure computed during single-level updates and e is a correction computed during synchronization to ensure that the velocity is divergence-free in a composite-operator sense.) For notational simplicity, we suppress the time centering of π and henceforth refer simply to π^ℓ.

We also need flux registers to contain coarse–fine matching information. $\delta\vec{V}^\ell$ contains the normal and tangential (to the coarse–fine interface) momentum fluxes across the coarse–fine interface between level ℓ and the coarser level $\ell - 1$, while $\delta\Lambda^\ell$ contains the fluxes of the advected scalar Λ. A complete list of the variables used in this algorithm appears in Fig. 3.

2.2. Single-level update

The complete recursive algorithm used to advance the level ℓ solution from time t^ℓ to $t^\ell + \Delta t^\ell$ is presented in pseudocode form in Fig. 4.

Steps 1, 2, and 3 are very similar to those in [19]. The hyperbolic tracing scheme predicts face-centered values at the half-time as detailed in the Appendix in [19], implementing the three-dimensional transverse predictor as described in [26] and the stable treatment of source terms described in [21].

(1) *Compute advection velocities*. We predict a set of upwinded face-centered velocities $\vec{u}^{\mathrm{half}}_G$ at time $t^\ell + \frac{1}{2}\Delta t^\ell$, using the hyperbolic tracing scheme. The source term used in the predictor step is $S = L^\ell_{\mathrm{visc}}\vec{u}^\ell(t^\ell)$, where L^ℓ_{visc} is the level-operator form of the diffusion operator L_{visc}.

These velocities are then projected using the face-centered projection to compute a set of divergence-free face-centered velocities at the intermediate time $t^\ell + \frac{1}{2}\Delta t^\ell$. First, we perform an elliptic solve for a correction:

$\vec{u}^{\ell}$	$(u^{\ell}, v^{\ell}, w^{\ell})$ – cell-centered velocity on level ℓ
$\vec{u}_{AD}$	face-centered advection velocity, centered at half-time
$\vec{u}_G^{half}$	(unprojected) Godunov-predicted face velocities at half-time
$\vec{u}^{half}$	projected predicted face-centered velocities at the half-time
$\vec{u}^{*,\ell}$	(unprojected) approximation to cell-centered velocity at new time
Λ^{ℓ}	cell-centered freestream-preservation advected scalar
π^{ℓ}	half-time-centered cell-centered level pressure
ν	kinematic viscosity
e_{Λ}	cell-centered freestream preservation correction on level ℓ
$\vec{u}_p$	$= G^{comp} e_{\Lambda}$: face-centered correction to $\vec{u}_{AD}$ to maintain freestream preservation
e_s	cell-centered correction from multilevel synchronization projection
ϕ^{ℓ}	cell-centered correction computed by face-centered projection
$\mu_1, \mu_2, \mu_3, \mu_4$	parameters for viscous term computation
$D^{comp,\ell}$	composite multilevel divergence, evaluated on level ℓ
$G^{comp,\ell}$	composite multilevel face-centered gradient, evaluated on level ℓ
$L^{comp,\ell}$	composite multilevel Laplacian, evaluated on level ℓ
D^{ℓ}	single-level divergence on level ℓ
G^{ℓ}	single-level face-centered gradient on level ℓ
L^{ℓ}	single-level Laplacian on level ℓ
$L_{visc}\vec{u}$	discrete viscous operator $\approx \nu\Delta\vec{u}$
$L^{\ell}_{visc}\vec{u}$	single-level discrete viscous operator $\nu L^{\ell}\vec{u}$
$D^{CC,comp,\ell}$	composite multilevel cell-centered divergence, evaluated on level ℓ
$G^{CC,comp,\ell}$	composite multilevel cell-centered gradient, evaluated on level ℓ
$D^{CC,\ell}$	single-level cell-centered divergence on level ℓ
$G^{CC,\ell}$	single-level cell-centered gradient on level ℓ
D^{ℓ}_R	reflux divergence on coarse side of $(\ell+1)/\ell$ interface
$\delta V^{\ell}, \delta\Lambda^{\ell}$	hyperbolic flux registers containing flux mismatches along the $\ell/(\ell-1)$ coarse-fine interface
$I(\phi^{\ell},\phi^{\ell-1})$	quadratic coarse-fine interpolation of ϕ along $\ell/(\ell-1)$ coarse-fine interface
Ω^{ℓ}	portion of domain contained in AMR level ℓ
Ω^{ℓ}_{valid}	portion of Ω^{ℓ} not covered by finer levels
$\partial\Omega^{\ell,*}$	boundary of level ℓ, including coarse-fine interface with level $\ell-1$
t^{ℓ}	current time on level ℓ
Δt^{ℓ}	timestep on level ℓ
h^{ℓ}	mesh spacing on level ℓ
n^{ℓ}_{ref}	refinement ratio between levels ℓ and $\ell+1$
$Av^{F\to C}$	averaging operator from faces to cell centers
$Av^{C\to F}$	averaging operator from cell centers to faces
$\mathcal{C}_r(i,j,k)$	Coarsening operator (factor of r) for cell (i,j,k).

Fig. 3. List of variables.

$$L^{\ell}\phi = D^{\ell}\vec{u}_G^{\text{half}}. \tag{14}$$

If $\ell > 0$, coarse–fine boundary conditions for the solve are quadratic interpolation with the level pressure π:

$$\phi^{\ell} = I\left(\phi^{\ell}, \frac{1}{2}\Delta t\pi^{\ell-1}\right). \tag{15}$$

Domain boundary conditions are the standard projection boundary conditions ($\frac{\partial\phi}{\partial n} = 0$ at solid wall boundaries) [14]. Then the face-centered velocity field is corrected:

$$\vec{u}^{\text{half},\ell} = \vec{u}_G^{\text{half}} - G^{\ell}\phi^{\ell}. \tag{16}$$

Coarse–fine boundary conditions for computing $G^{\ell}\phi^{\ell}$ are given by (15).

To correct for freestream preservation errors, the freestream preservation correction $\vec{u}_{\text{p}}$ is added to create a set of advection velocities:

$$\vec{u}^{\ell}_{\text{AD}} = \vec{u}_{\text{half},\ell} + \vec{u}^{\ell}_{\text{p}}. \tag{17}$$

NSLevelAdvance$(\ell, t^\ell, \Delta t^\ell)$

Compute advection velocities $\vec{u}^\ell_{AD}$ (1)

Compute advective updates: (2)
$\Lambda^\ell_{\boldsymbol{i}}(t^\ell + \Delta t^\ell) = \Lambda^\ell_{\boldsymbol{i}}(t^\ell) - \Delta t^\ell D^\ell(\mathbf{F}^{\Lambda,\ell})_{\boldsymbol{i}}$

Predict $\vec{u}^{\text{half}}$ (3)

Compute $\vec{u}^{*,\ell}_{\boldsymbol{i}} = \vec{u}^\ell_{\boldsymbol{i}}(t^\ell) - \Delta t[(\vec{u}\cdot\nabla)\vec{u}]^{n+\frac{1}{2}}_{\boldsymbol{i}} - \Delta t G^{\text{CC},\ell}\pi^\ell - \Delta t[L_{visc}\vec{u}]^{TGA}$ (4)

Update advective and velocity flux registers: (5)
if $(\ell < \ell_{\max})$ **then**
$\delta\mathbf{V}^{\ell+1}_i = -\vec{u}^{\text{AD},\ell}_i\vec{u}^{\text{half},\ell} - (F^{TGA}_i)^\ell \quad \text{on } \mathcal{C}_{n^\ell_{\text{ref}}}(\partial\Omega^{\ell+1<*})$
$\delta\Lambda^{\ell+1}_i = -\vec{u}^{\text{AD},\ell}_i\Lambda^{half,\ell} \quad \text{on } \mathcal{C}_{n^\ell_{\text{ref}}}(\partial\Omega^{\ell+1<*})$
end if
if $(\ell > 0)$ **then**
$\delta\mathbf{V}^\ell_i = \delta\mathbf{V}^\ell_i + \frac{1}{n^{\ell-1}_{\text{ref}}}\langle\vec{u}^{\text{AD},\ell}_i\vec{u}^{\text{half},\ell}\rangle + \frac{1}{n^{\ell-1}_{ref}}\left\langle(F^{TGA}_i)^\ell\right\rangle \quad \text{on } \mathcal{C}_{n^{\ell-1}_{\text{ref}}}(\partial\Omega^{\ell,*})$
$\delta\Lambda^\ell_i = \delta\Lambda^\ell_i + \frac{1}{n^{\ell-1}_{\text{ref}}}\langle\vec{u}^{\text{AD},\ell}_i\Lambda^{half,\ell}\rangle \quad \text{on } \mathcal{C}_{n^{\ell-1}_{\text{ref}}}(\partial\Omega^{\ell,*})$
end if

Level Project $\vec{u}^{*,\ell} \to \vec{u}^\ell(t^\ell + \Delta t^\ell)$: (6)
Remove old $\nabla\pi$: $\vec{u}^{*,\ell} := \vec{u}^{*,\ell} + \Delta t G^{\text{CC},\ell}\pi^\ell$
Solve $L^\ell\pi^\ell = \frac{1}{\Delta t^\ell}D^{\text{CC},\ell}\vec{u}^{*,\ell}$
$\vec{u}^\ell(t^\ell + \Delta t^\ell) = \vec{u}^{*,\ell} - \Delta t^\ell G^{\text{CC},\ell}\pi^\ell$

if $(\ell < \ell_{\max})$ **(7)**
$\Delta t^{\ell+1} = \frac{1}{n^\ell_{\text{ref}}}\Delta t^\ell$
for $n = 0 < n^\ell_{\text{ref}} - 1$
NSLevelAdvance$(\ell+1, t^\ell + n\Delta t^{\ell+1}, \Delta t^{\ell+1})$
end for

if $((t^\ell + \Delta t^\ell) < (t^{\ell-1} + \Delta t^{\ell-1}))$ **Synchronize**$(\ell, t^\ell + \Delta t^\ell, \Delta t^\ell)$ **(8)**
end if
end NSLevelAdvance

Fig. 4. Recursive level time step for the incompressible Navier–Stokes equations. Numbers refer to algorithmic items in Section 2.2.

(2) *Update* Λ. The scalar update scheme is unchanged from [19]; upwinded face-centered values at the half time Λ^{half} are predicted using the hyperbolic tracing scheme, which are then used with the advection velocities to compute a conservative scalar update. As in [19], the update equation used is

$$\Lambda^\ell(t^\ell + \Delta t^\ell) = \Lambda^\ell - \Delta t^\ell D^\ell(\vec{u}^\ell_{\text{AD}}\Lambda^{\text{half},\ell}). \tag{18}$$

(3) *Predict transverse* $\vec{u}^{\text{half}}$ *and* $[(\vec{u}\cdot\nabla)\vec{u}]^{\text{half},\ell}$. Using the advection velocities $\vec{u}^\ell_{\text{AD}}$, the transverse components of the staggered-grid $\vec{u}^{\text{half}}$ are computed (the normal components of $\vec{u}^{\text{half}}$ were computed in step (1)) using the hyperbolic tracing scheme and are corrected using the projection correction computed in (1), as in [19]. At this point, the nonlinear advection term is computed as follows:

$$[(\vec{u}\cdot\nabla)\vec{u}]^{\text{half},\ell} = \text{Av}^{\text{F}\to\text{C}}(\vec{u}^\ell_{\text{AD}})\cdot(G^\ell\vec{u}^{\text{half},\ell}). \tag{19}$$

Following [19], the nonlinear advection terms are computed using an advective form rather than the conservative form because the freestream preservation correction to the advection velocities produces an advection velocity field which is not discretely divergence-free.

(4) *Compute* $\vec{u}^*$ (*evaluate viscous terms*). The viscous terms are evaluated semi-implicitly and the intermediate velocity $\vec{u}^{*,\ell}$ is computed using the method described in Section 1.2. The update proceeds as follows:

(a) *Compute diffused source term.*

$$\vec{f}^* = -[(\vec{u}\cdot\nabla)\vec{u}]^{n+\frac{1}{2},\ell} - G^{\mathrm{CC},\ell}\pi^\ell, \tag{20}$$

$$\vec{f} = \Delta t^\ell (I + \mu_4^\ell L_{\mathrm{visc}}^\ell)\vec{f}^*, \tag{21}$$

where coarse–fine boundary conditions for the computation of $\vec{f}^\ell$ are given by second-order extrapolation of $\vec{f}^*$ normal to the coarse–fine interface. Physical boundary conditions for the computation of $\vec{f}^\ell$ are the same as the viscous boundary conditions on velocity (homogeneous Dirichlet for solid walls).

(b) *Intermediate solve.*

Then, an intermediate solve is performed for $\vec{u}_e^\ell$:

$$(I - \mu_2^\ell L_{\mathrm{visc}}^\ell)\vec{u}_e^\ell = (I + \mu_3^\ell L_{\mathrm{visc}}^\ell)\vec{u}^\ell(t^\ell) + \vec{f}. \tag{22}$$

Coarse-fine boundary conditions for $\vec{u}_e^\ell$ are quadratic interpolation with the coarse-level velocity linearly interpolated in time:

$$\vec{u}_e^\ell = I(\vec{u}_e^\ell, \vec{u}^{\ell-1}(t^\ell + (\Delta t^\ell - \mu_1^\ell))). \tag{23}$$

(c) *Solve for* $\vec{u}^*$.

A second solve is then performed for the intermediate velocity $\vec{u}^*$:

$$(I - \mu_1^\ell L_{\mathrm{visc}}^\ell)\vec{u}^{*,\ell} = \vec{u}_e^\ell, \tag{24}$$

with coarse–fine boundary conditions (if required):

$$\vec{u}^{*,\ell} = I(\vec{u}^{*,\ell}, \vec{u}^{\ell-1}(t^\ell + \Delta t^\ell)). \tag{25}$$

(5) *Initialize/update momentum and advective flux registers.* Once the updates have been completed, the flux registers may be updated to contain the mismatches between the coarse- and fine-level fluxes along coarse–fine interfaces. We define the viscous flux $\vec{F}^{\mathrm{TGA}}$:

$$\vec{F}^{\mathrm{TGA},\ell} = \frac{\nu}{\Delta t} G^\ell(\mu_1\vec{u}^{*,\ell} + \mu_2\vec{u}_e^\ell + \mu_3\vec{u}^\ell(t^\ell) + \Delta t^\ell\mu_4\vec{f}^{*,\ell}). \tag{26}$$

For faces with normals in the ith direction,

- if $(\ell < \ell_{\max})$

$$\delta\vec{V}_i^{\ell+1} := -\vec{u}_{\mathrm{AD},i}^\ell \vec{u}^{\mathrm{half},\ell} - F_i^{\mathrm{TGA},\ell}$$

$$\delta\Lambda_i^{\ell+1} := -\vec{u}_{\mathrm{AD},i}^\ell \Lambda^{\mathrm{half},\ell}$$

- if $(\ell > 0)$

$$\delta\vec{V}_i^\ell := \delta\vec{V}_i^\ell + \frac{1}{n_{\mathrm{ref}}^{\ell-1}}\langle \vec{u}_{\mathrm{AD},i}^\ell \vec{u}^{\mathrm{half},\ell} + F_i^{\mathrm{TGA},\ell}\rangle$$

$$\delta\Lambda_i^\ell := \delta\Lambda_i^\ell - \frac{1}{n_{\mathrm{ref}}^{\ell-1}}\langle \vec{u}_{\mathrm{AD},i}^\ell \Lambda^{\mathrm{half},\ell}\rangle$$

where $\vec{u}_{\mathrm{AD},i}^\ell$ is the component of $\vec{u}_{\mathrm{AD}}^\ell$ in the ith component direction.

(6) *Project* $\vec{u}^{*,\ell} \rightarrow \vec{u}^{\ell}(t^{\ell} + \Delta t^{\ell})$. In the same way as in [19], the cell-centered level-operator projection $\mathbb{P}^{\mathrm{CC},\ell}$ is applied to the intermediate velocity field $\vec{u}^{*,\ell}$. First, solve for the approximation to the pressure $\pi^{\ell}_{\mathrm{new}} = \pi^{\ell}\left(t^{\ell} + \frac{1}{2}\Delta t^{\ell}\right)$. Note that the pressure gradient $G^{\mathrm{CC},\ell}\pi^{\ell}_{\mathrm{old}}$, included in the computation of $\vec{u}^{*,\ell}$ in step (4), is removed before projecting:

$$L^{\ell}\pi^{\ell}_{\mathrm{new}} = \frac{1}{\Delta t} D^{\mathrm{CC},\ell}(\vec{u}^{*,\ell} + \Delta t^{\ell} G^{\mathrm{CC},\ell}\pi^{\ell}_{\mathrm{old}}). \tag{27}$$

If $\ell > 0$, then coarse–fine boundary conditions are required both for $D^{\mathrm{CC},\ell}$ and $L^{\mathrm{CC},\ell}$. The coarse–fine boundary condition used to compute the source term for the projection is quadratic interpolation:

$$\vec{u}^{*,\ell} = I(\vec{u}^{*,\ell}, \vec{u}^{\ell-1} + \Delta t^{\ell} G^{\ell-1}\pi^{\ell-1}). \tag{28}$$

Note that this coarse–fine boundary condition differs from that used for the single-level projection in [19] (which was an extrapolation of $\vec{u}^{*,\ell}$ at the coarse–fine interface). It was found that using this quadratic interpolation boundary condition provided better matching at the coarse–fine interface for viscous flows, which in turn results in a smaller correction when the multilevel projection is applied during the synchronization phase. The coarse–fine boundary condition used for π^{ℓ} in the elliptic solve is:

$$\pi^{\ell} = I(\pi^{\ell}, \pi^{\ell-1}). \tag{29}$$

Then, the velocity field is corrected:

$$\vec{u}^{\ell}(t^{\ell} + \Delta t^{\ell}) = \vec{u}^{*,\ell} + \Delta t^{\ell} G^{\mathrm{CC},\ell}\pi^{\ell}_{\mathrm{old}} - \Delta t^{\ell} G^{\mathrm{CC},\ell}\pi^{\ell}_{\mathrm{new}}. \tag{30}$$

(7) *Recursive update of finer levels*. If a finer level $\ell + 1$ exists, it is then updated n^{ℓ}_{ref} times with a timestep of $\Delta t^{\ell+1} = \frac{1}{n^{\ell}_{\mathrm{ref}}}\Delta t^{\ell}$. This brings all levels finer than level ℓ to time $t^{\ell} + \Delta t^{\ell}$.

(8) *Synchronize with finer levels*. If a finer level $\ell + 1$ exists, we now synchronize level ℓ with all finer levels, as described in the next section.

Synchronize($\ell_{\mathrm{base}}, t^{\mathrm{sync}}, \Delta t^{\mathrm{sync}}$)
  Refluxing: (1)
    for $\ell = \ell_{\max} - 1, \ell_{\mathrm{base}}, -1$
      $\Lambda^{\ell}(t^{\mathrm{sync}}) := \Lambda^{\ell}(t^{\mathrm{sync}}) - \Delta t^{\ell} D^{\ell}_{R}(\delta\Lambda^{\ell+1})$
    end for
    Solve $(\boldsymbol{I} - \Delta t^{\ell_{base}} L^{comp}_{visc})\delta\vec{u} = -\Delta t^{\ell} D^{\ell}_{R}(\delta\mathbf{V}^{\ell+1})$ for $\ell \geq \ell_{base}$
    $\vec{u}^{\ell}(t^{\mathrm{sync}}) := \vec{u}^{\ell}(t^{\mathrm{sync}}) + \delta\vec{u}$

  Apply Synchronization Projection: (2)
    Solve $L^{\mathrm{comp}} e_s = D^{\mathrm{CC}\cdot\mathrm{comp}}\vec{u}(t^{\mathrm{sync}})$ for $\ell \geq \ell_{\mathrm{base}}$
      $e_s^{\ell_{\mathrm{base}}} = I(e_s^{\ell_{\mathrm{base}}}, \frac{\Delta t^{sync}}{\Delta t^{sync\cdot(\ell_{base}-1)}} e_s^{\ell_{\mathrm{base}}-1})$
    $\vec{u}(t^{\mathrm{sync}}) := \vec{u}(t^{\mathrm{sync}}) - \Delta t^{\mathrm{sync}} G^{\mathrm{CC}\cdot\mathrm{comp}} e_s$ for $\ell \geq \ell_{\mathrm{base}}$

  Freestream Preservation Solve: (3)
    Solve $L^{\mathrm{comp}} e_{\Lambda} = \frac{(\Lambda(t^{\mathrm{sync}})-1)}{\Delta t^{\mathrm{sync}}}\eta$ for $\ell \geq \ell_{\mathrm{base}}$
      $e_{\Lambda}^{\ell_{\mathrm{base}}} = I(e_{\Lambda}^{\ell_{\mathrm{base}}}, e_{\Lambda}^{\ell_{\mathrm{base}}-1})$
    $\vec{u}_{\mathrm{p}} = G^{\mathrm{comp}} e_{\Lambda}$

  Average finer solution onto coarser levels: (4)
    for $\ell = \ell_{\max} - 1, \ell_{\mathrm{base}}, -1$
      $\vec{u}^{\ell}(t^{\mathrm{sync}}) = \langle \vec{u}^{\ell+1}(t^{\mathrm{sync}}) \rangle$ on $\mathcal{C}_{n^{\ell}_{\mathrm{ref}}}(\Omega^{\ell+1})$
      $\Lambda^{\ell}(t^{\mathrm{sync}}) = \langle \Lambda^{\ell+1}(t^{\mathrm{sync}}) \rangle$ on $\mathcal{C}_{n^{\ell}_{\mathrm{ref}}}(\Omega^{\ell+1})$
    end for
end Synchronize

Fig. 5. Synchronization for incompressible Navier–Stokes equations. Numbers refer to algorithmic items in Section 2.3.

2.3. Synchronization

Synchronization is an essentially multilevel operation which ties together the different AMR levels after they have been advanced independently of each other. For this reason, synchronization operations are applied to all levels which have reached the synchronization time t^{sync} simultaneously. We denote the coarsest level which has reached t^{sync} as ℓ_{base}. For example, in a computation with a finest refinement level of 3, the first synchronization operations will be performed when levels 2 and 3 reach the same time. Then, as the nested advance proceeds, eventually levels 1, 2, and 3 will reach the same time t^{sync}; at that point, the synchronization will be performed over all levels $\ell \geqslant \ell_{\text{base}}$, where ℓ_{base} is 1. A pseudocode representation of the synchronization operations is presented in Fig. 5.

(1) *Refluxing*. Flux mismatches stored in the flux registers $\delta\vec{V}$ and $\delta\Lambda$ are used to correct the solution along the coarse side of coarse–fine interfaces. For the non-diffusive scalar Λ, we apply the correction explicitly, as in [19]:

$$\Lambda^{\ell} = \Lambda^{\ell} - \Delta t^{\ell} D_{\text{R}}^{\ell}(\delta\Lambda^{\ell+1}) \quad \text{for } \ell \geqslant \ell_{\text{base}}. \tag{31}$$

The hyperbolic flux corrections, when used in conjunction with the upwind predictor step for computing the edge-centered values, serve the purpose of applying the appropriate local inflow/outflow boundary conditions for the advection operator. This is true even for velocity advection, even though it is not differenced in conservation form. Since the velocity flux correction contains diffusive fluxes, stability considerations require that we apply this correction implicitly. We first solve a Helmholtz equation for a correction:

$$(I - \Delta t^{\ell_{\text{base}}} L_{\text{visc}}^{\text{comp}})\delta\vec{u} = \Delta t^{\ell} D_{\text{R}}(\delta\vec{V}^{\ell+1}) \quad \text{for } \ell \geqslant \ell_{\text{base}}. \tag{32}$$

At physical boundaries, the correction $\delta\vec{u}$ satisfies the homogeneous form of the viscous boundary conditions for the velocity. If the base level has a coarse–fine interface with level $(\ell_{\text{base}} - 1)$, the coarse–fine boundary condition for $\delta\vec{u}$ is quadratic interpolation with 0's on the coarser level $(\ell_{\text{base}} - 1)$:

$$\delta\vec{u}^{\ell_{\text{base}}} = I(\delta\vec{u}^{\ell_{\text{base}}}, 0^{\ell_{\text{base}}-1}).$$

Then, the correction is added to the velocity field

$$\vec{u}^{\ell} := \vec{u}^{\ell} + \delta\vec{u}^{\ell} \quad \text{for } \ell \geqslant \ell_{\text{base}}. \tag{33}$$

(2) *Apply multilevel projection*. To ensure that the velocity field is divergence-free in a composite sense, we apply a composite projection during synchronization, as in [19]. We solve a multilevel Poisson equation for the correction:

$$L^{\text{comp}} e_{\text{s}} = D^{\text{CC,comp}} \vec{u}^{\text{comp}} \quad \text{for } \ell \geqslant \ell_{\text{base}}. \tag{34}$$

If $\ell_{\text{base}} > 0$, coarse–fine boundary conditions are required. When computing $D^{\text{CC,comp}}\vec{u}^{\text{comp}}$, the coarse–fine boundary condition is quadratic interpolation with the coarser-level velocity field, linearly interpolated in time to t^{sync}:

$$\vec{u}^{\ell_{\text{base}}} = I(\vec{u}^{\ell_{\text{base}}}, \vec{u}^{(\ell_{\text{base}}-1)}(t^{\text{sync}})).$$

The coarse–fine boundary condition for the elliptic solve is

$$e_{\text{s}}^{\ell_{\text{base}}} = I\left(e_{\text{s}}^{\ell_{\text{base}}}, \frac{\Delta t^{\text{sync}}}{\Delta t^{\text{sync},(\ell_{\text{base}}-1)}} e_{\text{s}}^{\ell_{\text{base}}-1}\right), \tag{35}$$

where $\Delta t^{\text{sync},(\ell_{\text{base}}-1)}$ is the Δt^{sync} used during the computation of $e_{\text{s}}^{(\ell_{\text{base}}-1)}$.
Once the correction has been computed, the velocity field is corrected:

$$\vec{u}^{\ell} := \vec{u}^{\ell} - G^{\text{CC,comp}} e_{\text{s}}^{\ell} \quad \text{for } \ell \geqslant \ell_{\text{base}}. \tag{36}$$

If $\ell_{\text{base}} > 0$, the coarse–fine boundary condition for computing the gradient is given by (35). Physical boundary conditions for the correction e_s and its gradient are the homogeneous form of those used for the single-level projection [19].

(3) *Compute freestream preservation correction*. The computation of the freestream preservation correction is unchanged from that presented in [19] and is presented again here for convenience. An elliptic equation is first solved for the potential:

$$L^{\mathrm{comp}} e_{\Lambda} = \eta \frac{(\Lambda - 1)}{\Delta t^{\ell_{\mathrm{base}}}} \quad \text{for } \ell \geqslant \ell_{\mathrm{base}}. \tag{37}$$

If required, the coarse–fine boundary condition is quadratic interpolation:

$$e_{\Lambda}^{\ell_{\mathrm{base}}} = I(e_{\Lambda}^{\ell_{\mathrm{base}}}, e_{\Lambda}^{\ell_{\mathrm{base}}-1}). \tag{38}$$

Then, a face-centered correction to the advection velocities is computed:

$$\vec{u}_{\mathrm{p}} = G^{\mathrm{comp}} e_{\Lambda} \quad \text{for } \ell \geqslant \ell_{\mathrm{base}}. \tag{39}$$

Coarse-fine boundary conditions for the gradient are given by (38).

(4) *Average fine solution onto coarser grids*. Finally, all quantities on covered regions, including the face-centered $\vec{u}_{\mathrm{p}}$, are replaced by the average of the overlying fine-grid solutions.

2.4. Initialization

At the beginning of a computation, an initial velocity field is specified. After a regridding operation, variables on any newly refined mesh are filled by interpolating the underlying coarse-cell values, while any regions which have been de-refined from a finer mesh are filled with averaged fine-level values. Once this has been done, a set of initialization operations is performed to ensure that the new velocity field is divergence-free and to initialize the lagged variables π and e_{Λ} (along with $\vec{u}_{\mathrm{p}}$), which are required for the single-level updates.

While re-initializing π results in a post-regridding initialization which is more computationally expensive, it is done because the existing pressures are centered at different temporal locations. Interpolating these would result in mismatched temporal errors in the pressure which would result in O(1) errors in the pressure gradient. Also, recomputing the pressure prevents interpolation artifacts from appearing in the pressure gradients. The velocity field is re-projected to prevent artifacts of the interpolation from appearing in the divergence; we have observed such artifacts to be significantly larger than the divergence resulting from the approximateness of the projection.

Initialize($\ell_{\mathrm{base}}, t^{\mathrm{init}}$)
  Project velocity field: (1)
    Solve $L^{comp}\phi = D^{comp}\vec{u}$ for $\ell > \ell_{\mathrm{base}}$
    Apply correction: $\vec{u}^{\ell} := \vec{u}^{\ell} - G^{comp}\phi$ for $\ell > \ell_{\mathrm{base}}$

  Initialize $\vec{u}_p$: (2)
    Solve $L^{\mathrm{comp}} e_{\Lambda} = \frac{(\Lambda(t^{\mathrm{init}})-1)}{\Delta t^{\ell_{base}}}\eta$ for $\ell \geq \ell_{\mathrm{base}}$
    $e_{\Lambda}^{\ell_{\mathrm{base}}} = I(e_{\Lambda}^{\ell_{\mathrm{base}}}, 0^{\ell_{\mathrm{base}}-1})$
    $\vec{u}_{\mathrm{p}} = G^{\mathrm{comp}} e_{\Lambda}$

  Initialize π: (3)
    $\pi^{\ell} = 0$ for $\ell > \ell_{base}$
    for $n = 1, n_{passes}$
      $\widetilde{\Delta t} = \frac{1}{2}\Delta t^{\ell_{max}}$
      for $\ell = \ell_{base}, \ell_{max}$
        Compute $\widetilde{\vec{u}^{*,\ell}}$ as in normal timestep
        Remove $\nabla\pi$ from $\widetilde{\vec{u}^{*,\ell}}$: $\widetilde{\vec{u}^{*,\ell}} := \widetilde{\vec{u}^{*,\ell}} + \widetilde{\Delta t} G^{CC,\ell}\pi^{\ell}$
        Solve $L^{\ell}\pi^{\ell} = D^{CC,\ell}\vec{u}^{*,\ell}$
        Correct $\widetilde{\vec{u}^{\ell}}$: $\widetilde{\vec{u}^{\ell}}(t^{\ell} + \widetilde{\Delta t}) = \widetilde{\vec{u}^{*,\ell}} - \widetilde{\Delta t}\pi^{\ell}$
      end for
    end for
end Initialize

Fig. 6. Initialization for incompressible Navier–Stokes equations. Numbers refer to algorithmic items in Section 2.4.

A pseudocode description of the initialization procedure appears in Fig. 6. For initialization, ℓ_{base} is the finest *unchanged* level (at the initial time, $\ell_{\text{base}} = -1$). $\Delta t^{\ell_{\text{base}}}$ is the most recent time step for level ℓ_{base}.

(1) *Project velocity field*. To ensure that the velocity field satisfies the divergence constraint, a multilevel projection is applied to the velocity field for all levels finer than ℓ_{base}. No correction is applied to the velocity field on ℓ_{base}. If ℓ_{base} is greater than -1, the coarse–fine boundary condition for the elliptic solve is a homogeneous quadratic coarse–fine interpolation with zeroes in the coarse grid cells:

$$\phi^{\ell_{\text{base}}+1} = I(\phi^{\ell_{\text{base}}+1}, 0^{\ell_{\text{base}}}). \tag{40}$$

(2) *Initialize freestream preservation correction*. While the freestream preservation correction $\vec{u}_{\text{p}}$ is simply initialized to 0 at the start of the computation, it must be recomputed after regridding. First, an elliptic equation is solved for the potential e_{Λ}:

$$L^{\text{comp}} e_{\Lambda} = \frac{\Lambda(t^{\text{init}}) - 1}{\Delta t^{\ell_{\text{base}}}} \eta \quad \text{for } \ell \geqslant \ell_{\text{base}}. \tag{41}$$

If $\ell_{\text{base}} > 0$, then the coarse–fine boundary condition for e_{Λ} in the elliptic solve is homogeneous quadratic interpolation:

$$e_{\Lambda}^{\ell_{\text{base}}} = I(e^{\ell_{\text{base}}}, 0^{\ell_{\text{base}}-1}). \tag{42}$$

The face-centered freestream-preservation correction is then given by:

$$\vec{u}_{\text{p}} = G^{\text{comp}} e_{\Lambda}. \tag{43}$$

Coarse-fine boundary conditions on e_{Λ} when computing $\vec{u}_{\text{p}}$ are also given by (42).

(3) *Initialize* $\boldsymbol{\pi}$. During the single-level update, computation of the intermediate velocity field $\vec{u}^*$ uses the lagged level pressure $\pi^{\ell}\left(t^{\ell} - \frac{1}{2}\Delta t^{\ell}\right)$. During the initialization step, we compute an approximation to the single-level pressure. To initialize π, simplified non-subcycled single-level timesteps are performed for all levels for which π must be initialized. The timestep used for this initialization step is half the timestep computed for the finest level in the AMR hierarchy ($\widetilde{\Delta t} = \frac{1}{2}\Delta t^{\ell_{\max}}$). Since there is no existing estimate of π when performing the velocity predictor and viscous updates, the pressure gradient terms are not included for these steps. If a more-accurate estimate of π is required, then a second iteration of the initialization timesteps may be performed. In practice, we have found one iteration to be sufficient to compute an adequate estimate for π. First, we compute $\widetilde{\vec{u}}^*$ as in a normal timestep, using the pressure gradient term if it is available. All of the coarse–fine boundary conditions for the initialization timesteps are the same as are used in a regular advance. We then project $\widetilde{\vec{u}}^*$ to compute π, solving

$$\begin{aligned} L^{\ell}\pi^{\ell} &= \frac{1}{\widetilde{\Delta t}} D^{\text{CC},\ell}(\widetilde{u}^*), \\ \pi^{\ell} &= I(\pi^{\ell}, \pi^{\ell-1}). \end{aligned} \tag{44}$$

Finally, $\widetilde{\vec{u}}^*$ is corrected for use as a boundary condition for initializing any finer levels:

$$\widetilde{\vec{u}}^{\ell}(t^{\ell} + \widetilde{\Delta t}) := \widetilde{\vec{u}}^* - \widetilde{\Delta t} G^{\text{CC},\ell}\pi^{\ell}. \tag{45}$$

3. Results

Ideally, we would like to show that the method described here is second-order accurate. Unfortunately, in the absence of exact solutions to the 3D incompressible Navier–Stokes equations, the standard method for doing this based on Richardson error estimation is not practical in the presence of time-dependent grid hierarchies. Instead, we investigate this indirectly, by comparing a Richardson-type error estimate for our adaptive method to the same error estimate for our second-order accurate method on a uniform-mesh solution with the equivalent finest-grid resolution. In addition, we show that the use of local refinement for appropriate problems results in significant savings in computational time and/or memory when compared to the equivalent uniform-mesh solution. Finally, we show results of a calculation of vortex ring merger that demonstrates the robustness of the method on a more complicated problem.

3.1. Convergence and accuracy

To demonstrate the convergence and accuracy of this approach, we use a single vortex ring in a unit cube domain. We use periodic boundary conditions in the z-direction and no-shear boundaries in the x- and y-directions. The vorticity distribution is specified, and the initial velocity is then computed based on the initial vorticity field. Each vortex ring is specified by the center of the vortex ring (x_0, y_0, z_0), the radius of the center of the local cross-section of the ring from the center of the vortex ring r, and the strength of the vortex ring Γ.

The cross-sectional vorticity distribution in the vortex ring is given by

$$\omega(\rho) = \frac{\Gamma}{a\sigma^2} e^{\left(\frac{-\rho}{\sigma}\right)^3}, \tag{46}$$

where ρ is the local distance from the center of the ring cross-section, $a = 2268.85$, and $\sigma = 0.0275$. The vortex ring is centered at $(x_0, y_0, z_0) = (0.5, 0.5, 0.4)$, with a radius of 0.2, and strength Γ of 1.5. This test problem is depicted in Fig. 7.

The solution is then advanced to a solution time of 0.06, with a fixed timestep ($\Delta t = 9.375 \times 10^{-4}$ for the 512^3 uniform mesh solution, double that for the 256^3 uniform mesh, and so on).

Since there is no analytic solution to this problem, the standard approach to estimate the rate of convergence would be to perform Richardson error estimation, in which one computes the solution on a succession of grid resolutions and estimates the error as the difference between successive resolutions. This approach was used in [1] by computing solutions on fixed AMR hierarchies which are refinements of each other, but this has the unfortunate effect of ignoring possible errors due to regridding, and generalizing this approach to time-varying grids would be extremely complicated. Instead, we have taken the indirect approach of computing an estimate of the error by comparing the solution to that computed on a uniform 512^3 mesh. One cannot use these errors directly to compute the order of accuracy. Such a calculation depends on knowing that the coefficient in front of the leading-order error term is the same for all of the calculations being compared, which is not the case here. Instead, we first demonstrate that the single-level scheme is second-order accurate in space and time using the standard Richardson extrapolation approach of comparing pairs of solutions to compute errors. Then, we compute errors compared to the uniform fine-grid solution to demonstrate that the AMR solutions achieve similar errors (and hence similar convergence) as the uniform mesh solutions with the equivalent finest resolution.

The L_2 and Ł$_\infty$ norms of error in the x-velocity are shown in Tables 1 and 2, respectively (solution errors for y- and z-velocities are similar). The left column indicates the number of cells on one side of the coarsest domain (so the $\frac{1}{h_0} = 16$ is a 16^3 computation). As expected, the single-level solutions converge at second-order rates, except for the coarsest cases, which are not well-resolved enough to be in the asymptotic regime. Comparing the errors for equivalent resolutions demonstrates the effectiveness and accuracy of the local

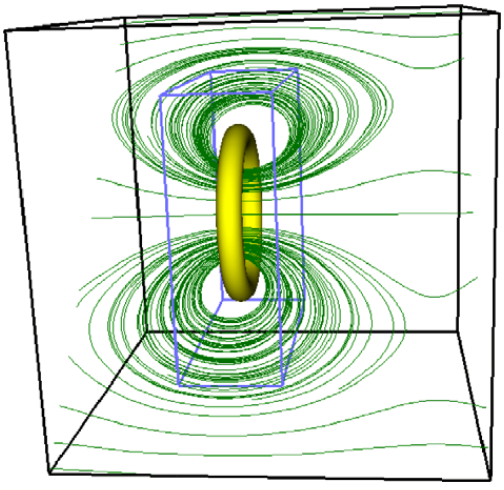

Fig. 7. Vortex ring test problem. Yellow vorticity isosurface depicts location of vortex ring, green lines depict streamlines, blue box is example of refined region. Black box depicts computational domain.

Table 1
L_2 Convergence of x-velocity for single-vortex test problem

$\frac{1}{h_0}$	Single-level Richardson	Rate	Single-level 512 difference	$n_{\text{ref}} = 2$	$n_{\text{ref}} = 4$	$n\text{ref} = (2,2)$
16	1.51e − 04	–	1.66e − 04	5.195e − 05	1.22e − 05	1.05e − 05
32	4.21e − 05	1.84	5.01e − 05	1.05e − 05	2.70e − 06	2.87e − 06
64	7.90e − 06	2.41	1.01e − 05	2.48e − 06	7.02e − 07	6.60e − 7
128	1.87e − 06	2.08	2.32e − 06	5.64e − 07	–	–
256	4.58e − 07	2.03	4.58e − 07	–	–	–

"Single-level Richardson" errors are computed by comparing uniform mesh computations which differ in resolution by a factor of 2. "Single-level 512 difference" errors are computed by comparing to a uniform fine (512^3) solution. AMR errors are computed using the "512 difference" approach.

Table 2
L_∞ Convergence of x-velocity for single-vortex test problem

$\frac{1}{h_0}$	Single-level Richardson	Rate	Single-level 512 difference	$n_{\text{ref}} = 2$	$n_{\text{ref}} = 4$	$n\text{ref} = (2,2)$
16	1.65e − 03	–	1.88e − 03	9.34e − 04	2.12e − 04	2.14e − 04
32	8.07e − 04	1.04	9.55e − 04	2.14e − 04	5.15e − 05	5.14e − 05
64	1.72e − 04	2.23	2.14e − 04	5.12e − 05	1.09e − 05	9.91e − 06
128	4.21e − 05	2.03	5.16e − 05	1.29e − 05	–	–
256	9.86e − 06	2.09	9.86e − 06	–	–	–

refinement. For example, the error for the 128^3 single-level case should be compared with the 64^3 $n_{\text{ref}} = 2$ case, the 32^3 $n_{\text{ref}} = 4$ case, and the 32^3 $n_{\text{ref}} = (2,2)$ case. The $n_{\text{ref}} = (2,2)$ case refers to two levels of refinement, each with a factor of 2 refinement, which results in the equivalent resolution as a single $n_{\text{ref}} = 4$ refinement. This case is included to demonstrate that the subcycled algorithm maintains its accuracy in the case with more than one level of refinement, and is a good indication that the synchronization step is correct for the case where ℓ_{base} is greater than 0. In all cases, we note that the AMR solutions achieve the accuracy of the equivalent uniform mesh computations.

3.2. Computational performance

To demonstrate the performance of the AMR algorithm, we measured the runtimes and total number of cells advanced (a rough indicator of memory usage) for the vortex-ring example for a single-level 256^3 computation, along with AMR computations with equivalent resolution. To make the performance effects of AMR clear, we then normalized the runtimes and cell counts by the single-level numbers, as shown in Table 3. In this figure, the refinement ratio of zero corresponds to a single-level 256^3 run, while the refinement ratio of 2 is a 128^3 base grid with one level of refinement with a refinement ratio of 2, etc. The difference between the timing line and the cell-count line in this case represents the overhead of adaptivity (regridding, synchronization, etc.). Both the $n_{\text{ref}} = 2$ and $n_{\text{ref}} = 4$ AMR computations show significant savings both in computational time and memory usage due to the use of adaptivity. Note that while the total number of cells advanced for $n_{\text{ref}} = 4$ is only slightly smaller than the number of cells advanced for $n_{\text{ref}} = 2$, the execution time is noticeably smaller. While approximately the same amount of work is done performing the fine-level updates, there are only half as many synchronization steps as the $n_{\text{ref}} = 2$ case. This agrees with the results presented in [1], and suggests that refinement ratios of 4 are to be preferred wherever possible. While it might be tempting to infer that the additional cost of synchronization makes the additional complexity of refinement in time less worthwhile, the additional efficiency resulting from subcycling (especially in the case of very deep AMR hierarchies) outweighs the additional overhead due to the synchronization step. This point was made in a fairly compelling way in [1].

3.3. Vortex Merger example

To demonstrate that the algorithm is robust enough to handle more complex problems, we also computed an adaptive solution for a viscous vortex ring merger problem, similar to the ones studied computationally in [2,4,17] and experimentally in [5,18]. The initial conditions are two vortex rings which are angled toward each

Table 3
AMR performance for single vortex ring problem

n_{ref}	Number of cells advanced	CPU time (s)	Normalized cells advanced	Normalized CPU time
0	536870912	23664	1.0	1.0
2	45547520	8661	0.0848	0.366
4	24346624	1571	0.0454	0.0664

Refinement ratio of 0 is the non-AMR (single-level) case. Normalized values are relative to the single-level case. CPU seconds are on an 1.8 g Hz Opteron using GNU compilers and a Linux operating system.

other by an inclination angle ϕ from horizontal. The vortex rings are each initialized with a solid vorticity core, with $\omega = \omega_{\text{interior}}$ inside the vortex ring, and $\omega = 0$ outside. The parameters used for this example are:

$$\omega_{\text{interior}} = 300.0,$$
$$r = 0.02,$$
$$R = 0.1,$$
$$\phi_1 = \frac{\pi}{9}, \quad \vec{x}_1 = (0.5, 0.625.0.5),$$
$$\phi_2 = \frac{-\pi}{9}, \quad \vec{x}_2 = (0.5, 0.375, 0.5),$$

where r is the cross-sectional radius of the vortex ring core, and R is the radius of the vortex ring around its center. $\vec{x}_1$ and $\vec{x}_2$ and ϕ_1 and ϕ_2 are the centers and inclinations of the two vortex rings. The viscosity ν is 0.001.

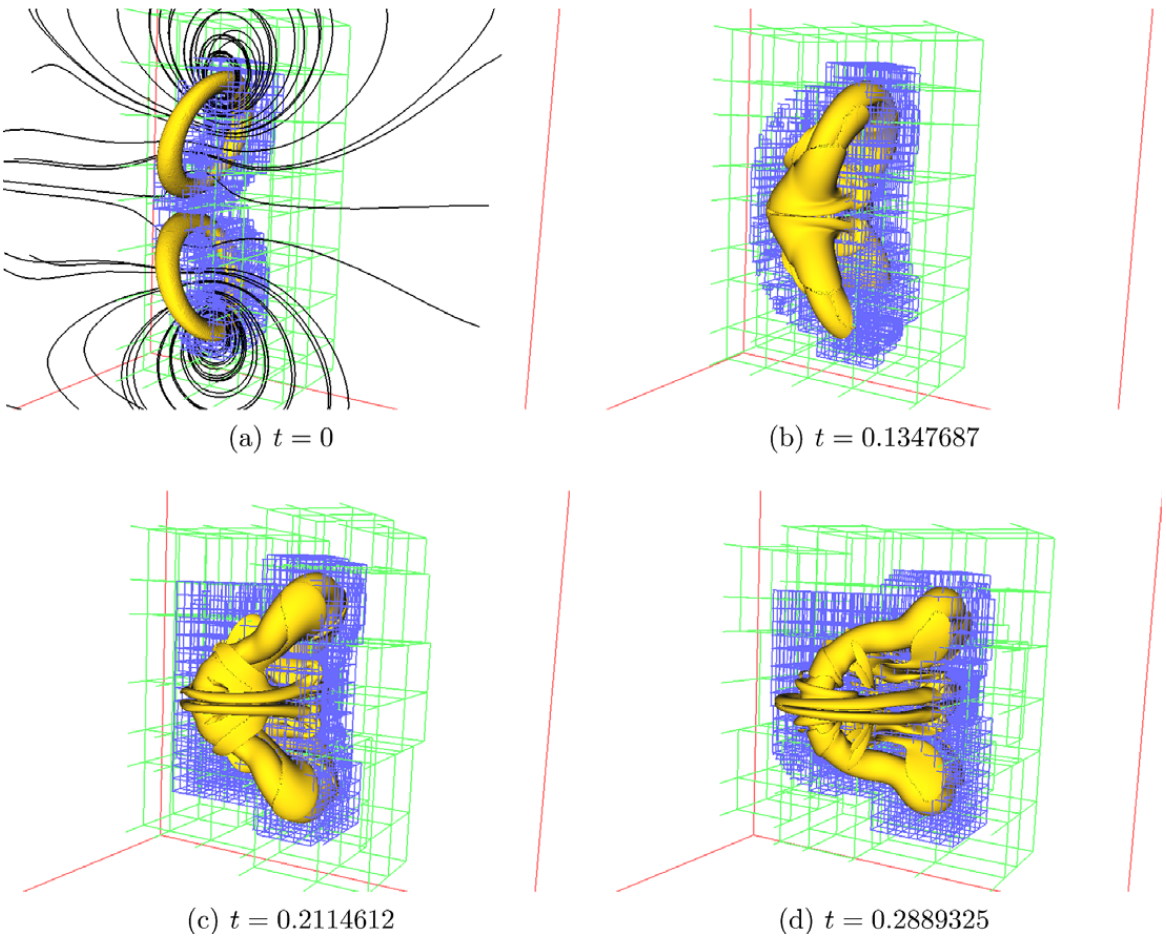

Fig. 8. Vortex merger problem – isosurface of $|\omega| = 50$: (a) at initial time, (b) after 60 timesteps, (c) after 90 timesteps, and (d) after 120 timesteps. Black lines depict streamlines, green boxes are level 1 grids, and blue boxes are level 2 grids. Note that for clarity the grid boxes are only shown in the rear half of the domain.

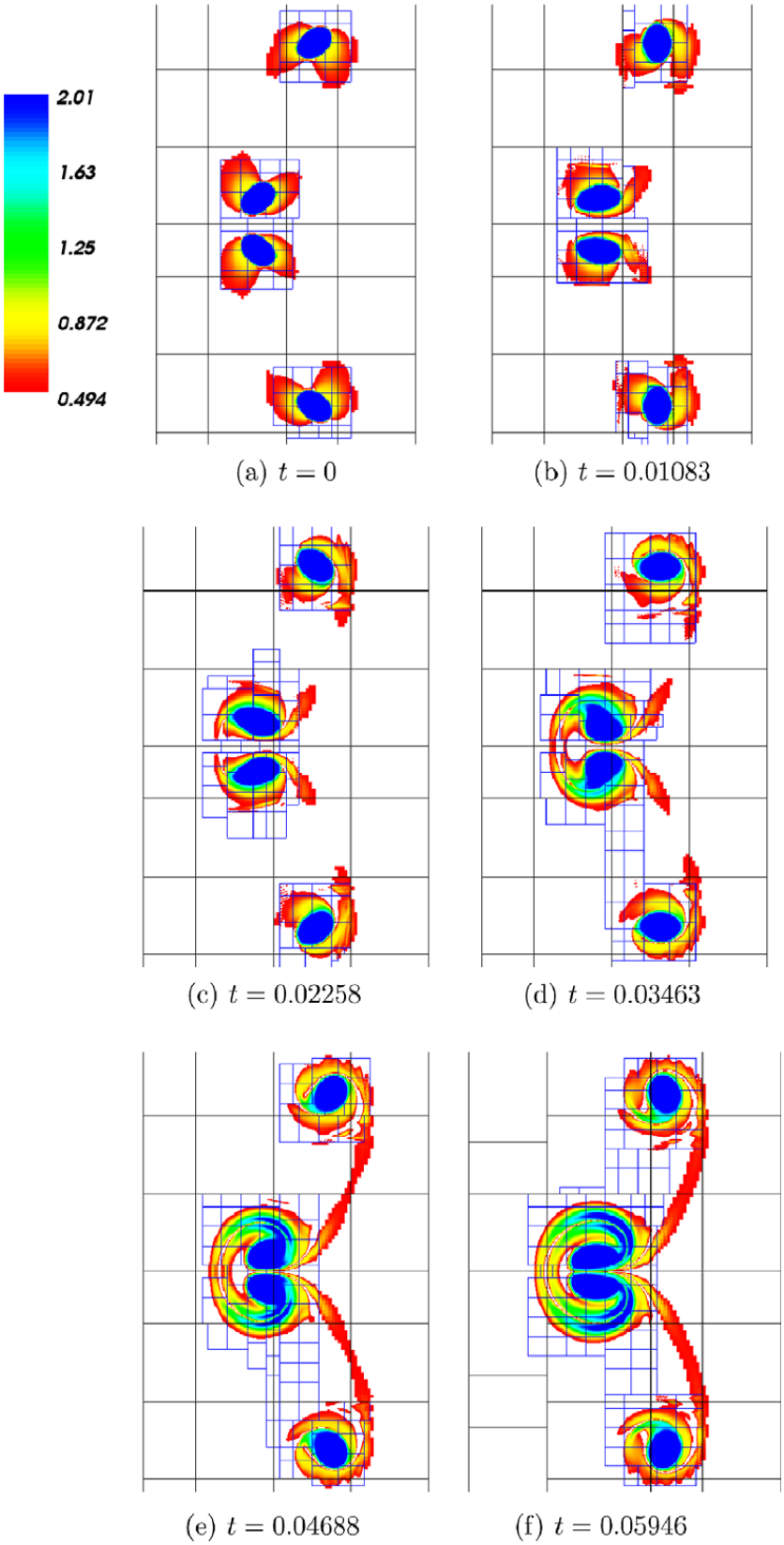

Fig. 9. Vortex merger problem – slice at $x = 0.505$ showing $\log_{10}|\omega|$ at: (a) initial time and after (b) 5 timesteps, (c) 10 timesteps, (d) 15 timesteps, (e) 20 timesteps, and (f) 25 timesteps. Colormap scale is from 0.5 to 2.0.

The problem was run in a unit cube with a 64^3 base mesh with 2 levels of refinement using $n_{\text{ref}} = 4$. Refinement is added wherever the undivided vorticity magnitude ($h_\ell * (\omega_x^2 + \omega_y^2 + \omega_z^2)^{\frac{1}{2}}$) is greater than 0.0625. Evolution of an isosurface of the vorticity magnitude is shown in Fig. 8. As can be seen, the two vortices merge in a fairly complicated way, with the refined regions smoothly following the vortical structures as they evolve.

To better see the structure of the merging vortices, we show the $\log_{10}$ of the vorticity magnitude in a slice through the $x = 0.505$ plane in Fig. 9; a closeup of the center as the vortex rings merge is shown in Fig. 10, which demonstrates the how the vortex ring cores are deformed by differential shearing and vorticity diffusion. As the flow progresses, sheets of vorticity (seen in cross-section as narrow strips) are stripped from the central

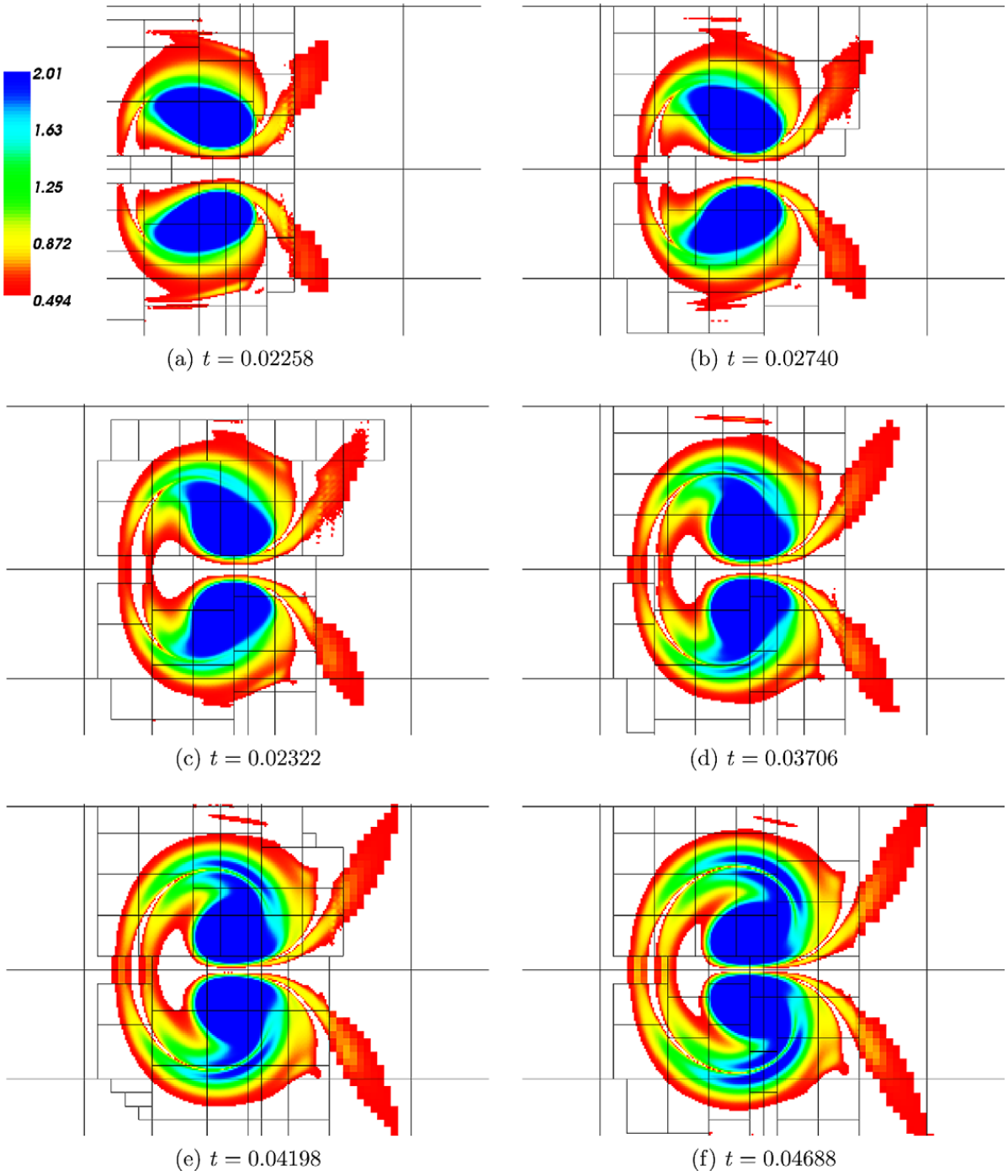

Fig. 10. Vortex merger problem – slice at $x = 0.505$ showing $\log_{10}|\omega|$ after: (a) 10 timesteps, (b) 12 timesteps, (c) 14 timesteps, (d) 16 timesteps, (3) 18 timesteps, and (f) 20 timesteps. Colormap scale is from 0.5 to 2.0.

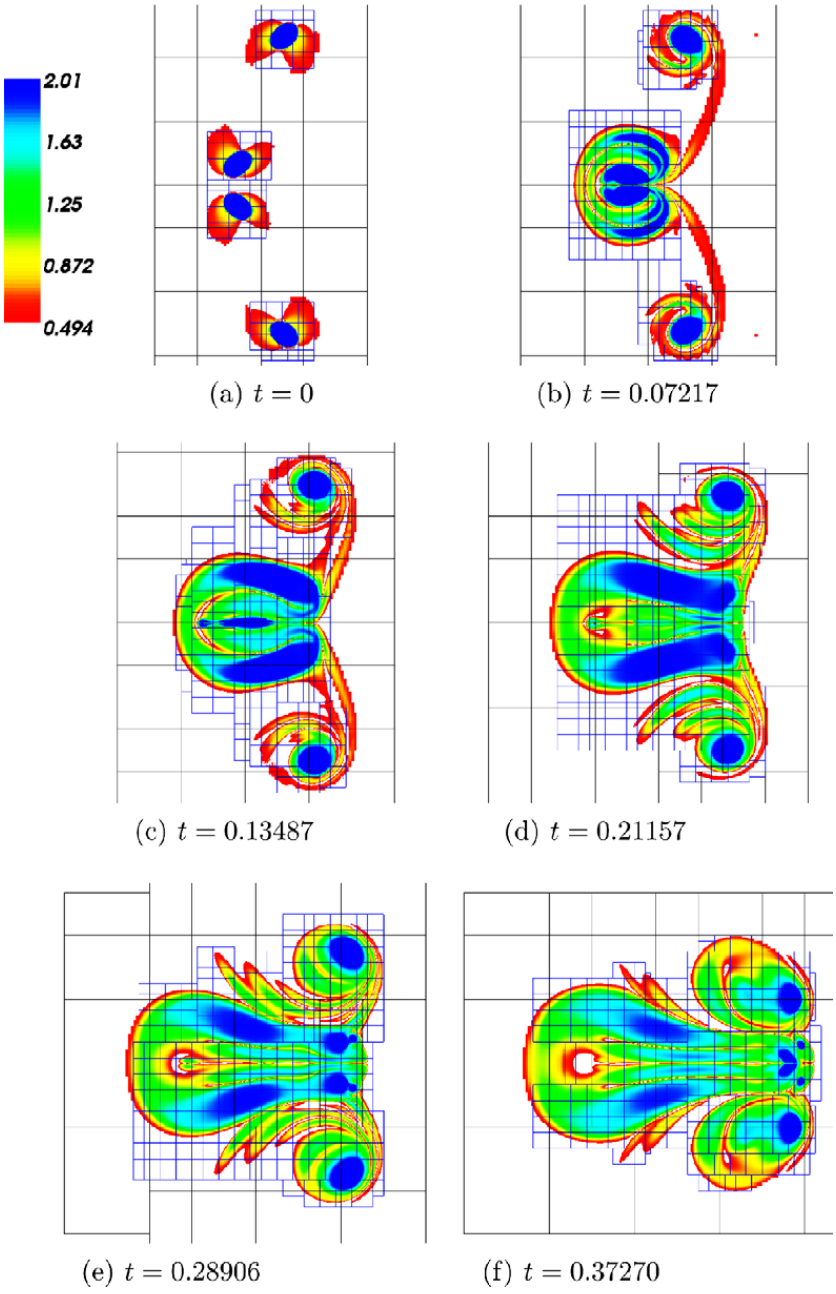

Fig. 11. Vortex merger problem – slice at $x = 0.505$ showing $\log_{10}|\omega|$ at (a) initial time and after (b) 30 timesteps, (c) 60 timesteps, (d) 90 timesteps, (e) 120 timesteps, and (f) 150 timesteps. Colormap scale is from 0.5 to 2.0.

vortex cores and are then wrapped around and transported away. A longer-term evolution is shown in Fig. 11, which shows the role of vortex stretching in the formation of the longer-term vortical structures in the flow.

4. Conclusions

In this work, an algorithm was presented to compute solutions to the incompressible Navier–Stokes equations with local refinement in time and space using a cell-centered discretization of the projection operator. Other key innovations differentiating this work from past work are the use of fully multilevel elliptic solves for synchronization and the use of an L_0-stable semi-implicit scheme (rather than Crank–Nicolson) to discretize the diffusive terms. We have demonstrated second-order convergence of the method, as well as the computational efficiencies enabled by the use of AMR.

This work will be extended in several directions. Extension to flows with variable properties, which will also entail the implementation of tensor solvers for the diffusion terms, is one such direction. Also, we plan to extend the ideas in this work to computing flows in complex geometries, using the embedded boundary approach [11]. Another possible direction would be the implementation of higher-order finite-volume schemes, such as those found in [6]. In general, we foresee application of these ideas to other coupled elliptic-parabolic-hyperbolic systems, such as those found in porous media flows [24] and non-ideal MHD [27].

Acknowledgments

Research supported by the Office of Advanced Scientific Computing Research of the US Department of Energy under contract number DE-AC02-05CH11231 and by the NASA Earth and Space Sciences Computational Technologies Program.

Appendix A. Quadratic coarse–fine boundary interpolation

This interpolation scheme is motivated by the requirement to construct consistent discretizations of second-order operators. Given the fine- and coarse-level variables φ^{f} and $\varphi^{\mathrm{c,valid}}$, we compute a single-level vector field $\vec{G}^{\mathrm{f}} = (G^{\mathrm{f}}_0, \ldots G^{\mathrm{f}}_{\mathbf{D}-1})$ that approximates the gradient to sufficient accuracy so that its divergence is at least an $\mathrm{O}(h)$ approximation to the Laplacian. For each $\Omega^{\mathrm{f},k} \in \mathcal{R}(\Omega^{\mathrm{f}})$, we construct an extension $\tilde{\varphi}$ of φ^{f}.

$$\tilde{\varphi} : \widetilde{\Omega}^{\mathrm{f}}_k \to \mathbb{R}^m,$$
$$\widetilde{\Omega}^{\mathrm{f}}_k = (\cup_{\pm=+,-} \cup_{d=0}^{\mathbf{D}-1} \Omega^{\mathrm{f}}_k \pm \boldsymbol{e}^d) \cap \Gamma^{\mathrm{f}}.$$

Then, for each $\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d$ such that both $\boldsymbol{i}, \boldsymbol{i} + \boldsymbol{e}^d \in \widetilde{\Omega}^{\mathrm{f}}_k$, we can compute a centered-difference approximation to the gradient on a staggered grid

$$G^{\mathrm{f}}_{d,\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \frac{1}{h^{\mathrm{f}}}(\tilde{\varphi}_{\boldsymbol{i}+\boldsymbol{e}^d} - \tilde{\varphi}_{\boldsymbol{i}}).$$

For this estimate of the gradient to be accurate to $\mathrm{O}(h^2)$, it is necessary to compute an $\mathrm{O}(h^3)$ extension of φ^{f}. On $\widetilde{\Omega}^{\mathrm{f}}_k \cap \Omega^{\mathrm{f}}$, the values for $\tilde{\varphi}$ will be given by $\tilde{\varphi}_{\boldsymbol{i}} = \varphi^{\mathrm{f}}_{\boldsymbol{i}}$. The values for the remaining points in $\widetilde{\Omega}^{\mathrm{f}}_k - \Omega^{\mathrm{f}}$ will be obtained by interpolating using φ^{f} and φ^{c}.

To perform this interpolation, we first observe that given $\boldsymbol{i} \in \widetilde{\Omega}^{\mathrm{f}}_k - \Omega^{\mathrm{f}}$, there is a unique choice of $\pm$ and d, such that $\boldsymbol{i} \mp \boldsymbol{e}^d \in \Omega^{\mathrm{f}}_k$. Having specified that choice, the interpolant is constructed in two steps (Fig. A.1).

(i) Interpolation in directions orthogonal to $\boldsymbol{e}^d$. We compute

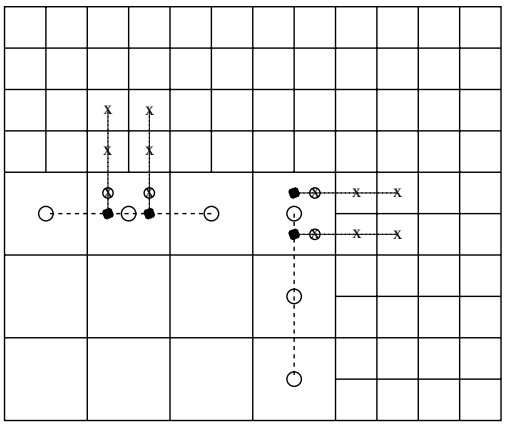

Fig. A.1. Interpolation at a coarse–fine interface. Left stencil is the usual stencil. Right stencil is the modified interpolation stencil; since the upper coarse cell is covered by a fine grid, use shifted coarse grid stencil (open circles) to get intermediate values (solid circles), then perform final 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.

$$\boldsymbol{x} = \frac{\boldsymbol{i} + \frac{1}{2}\boldsymbol{u}}{n_{\text{ref}}} - \left(\boldsymbol{i}^{\text{c}} + \frac{1}{2}\boldsymbol{u}\right),$$

where $\boldsymbol{i}^{\text{c}} = \mathcal{C}_{n_{\text{ref}}}(\boldsymbol{i})$. The real-valued vector $\boldsymbol{x}$ is the displacement of the cell center $\boldsymbol{i}$ on the fine grid from the cell center at $\boldsymbol{i}^{\text{c}}$ on the coarse grid, scaled by h^{c}.

$$\hat{\varphi}_{\boldsymbol{i}} = \varphi^{\text{c}}_{\boldsymbol{i}^{\text{c}}} + \sum_{d' \neq d} \left[\left(x_{d'} (D^{1,d'} \varphi^{\text{c}})_{\boldsymbol{i}^{\text{c}}} + \frac{1}{2} (x_{d'})^2 (D^{2,d'} \varphi^{\text{c}})_{\boldsymbol{i}^{\text{c}}} \right) + \sum_{d'' \neq d, d'' \neq d'} x_{d'} x_{d''} (D^{d'd''} \varphi^{\text{c}})_{\boldsymbol{i}^{\text{c}}} \right].$$

The second sum has only one term if $\mathbf{D} = 3$, and no terms if $\mathbf{D} = 2$.

(ii) Interpolation in the normal direction.

$$\tilde{\varphi}_{\boldsymbol{i}} = I^B_q(\varphi^{\text{f}}, \varphi^{\text{c,valid}}) \equiv 4a + 2b + c, \tilde{x_d} = x_d - \frac{1}{2}(n_{\text{ref}} + 3),$$

where a, b, c are computed to interpolate between the collinear data

$$\left(\left(\boldsymbol{i} \pm \frac{1}{2}(n^l_{\text{ref}} - 1)\boldsymbol{e}^d \right) h, \hat{\varphi}_{\boldsymbol{i}} \right),$$
$$((\boldsymbol{i} \mp \boldsymbol{e}^d)h, \varphi^l_{\boldsymbol{i} \mp \boldsymbol{e}_d}),$$
$$((\boldsymbol{i} \mp 2\boldsymbol{e}^d)h, \varphi^l_{\boldsymbol{i} \mp 2\boldsymbol{e}_d)}$$

In (i), the quantities $D^{1,d'}\varphi^{\text{c}}, D^{2,d'}\varphi^{\text{c}}$ and $D^{d'd''}\varphi^{\text{c}}$ are difference approximations to $\frac{\partial}{\partial x_{d'}}$, $\frac{\partial^2}{\partial x^2_{d'}}$, and $\frac{\partial^2}{\partial x_{d'} \partial x_{d''}}$, respectively. $D^{1,d}\varphi$ must be accurate to $\text{O}(h^2)$, while the other two quantities need only be $\text{O}(h)$. Only values in $\Omega^{\text{c}}_{\text{valid}}$ are used to compute these difference approximations. For $D^{1,d'}\varphi$ and $D^{2,d'}\varphi$, we use 3-point stencils, centered if possible, or shifted as required to consist of points on $\Omega^{\text{c}}_{\text{valid}}$.

$$(D^{1,d'}\varphi)_{\boldsymbol{i}} = \begin{cases} \frac{1}{2}(\varphi^{\text{c}}_{\boldsymbol{i}+\boldsymbol{e}^{d'}} - \varphi^{\text{c}}_{\boldsymbol{i}-\boldsymbol{e}^{d'}}) & \text{if both } \boldsymbol{i} \pm \boldsymbol{e}^{d'} \in \Omega^{\text{c}}_{\text{valid}}, \\ \pm\frac{3}{2}(\varphi^{\text{c}}_{\boldsymbol{i}\pm\boldsymbol{e}^{d'}} - \varphi^{\text{c}}_{\boldsymbol{i}}) \mp \frac{1}{2}(\varphi^{\text{c}}_{\boldsymbol{i}\pm 2\boldsymbol{e}^{d'}} - \varphi^{\text{c}}_{\boldsymbol{i}\pm\boldsymbol{e}^{d'}}) & \text{if } \boldsymbol{i} \pm \boldsymbol{e}^{d'} \in \Omega^{\text{c}}_{\text{valid}}, \boldsymbol{i} \mp \boldsymbol{e}^{d'} \notin \Omega^{\text{c}}_{\text{valid}}, \\ 0 & \text{otherwise}, \end{cases}$$

$$(D^{2,d'}\varphi)_{\boldsymbol{i}} = \begin{cases} \varphi^{\text{c}}_{\boldsymbol{i}+\boldsymbol{e}^{d'}} - 2\varphi^{\text{c}}_{\boldsymbol{i}} + \varphi^{\text{c}}_{\boldsymbol{i}-\boldsymbol{e}^{d'}} & \text{if both } \boldsymbol{i} \pm \boldsymbol{e}^{d'} \in \Omega^{\text{c}}_{\text{valid}}, \\ \varphi^{\text{c}}_{\boldsymbol{i}} - 2\varphi^{\text{c}}_{\boldsymbol{i}\pm\boldsymbol{e}^{d'}} + \varphi^{\text{c}}_{\boldsymbol{i}\pm 2\boldsymbol{e}^{d'}} & \text{if } \boldsymbol{i} \pm \boldsymbol{e}^{d'} \in \Omega^{\text{c}}_{\text{valid}}, \boldsymbol{i} \mp \boldsymbol{e}^{d'} \notin \Omega^{\text{c}}_{\text{valid}}, \\ 0 & \text{otherwise}. \end{cases}$$

In the case of $D^{d'd''}\varphi^{\text{c}}$, we use an average of all of the four-point difference approximations $\frac{\partial^2}{\partial x_{d'} \partial x_{d''}}$ centered at d',d'' corners adjacent to $\boldsymbol{i}$ such that all four points in the stencil are in $\Omega^{\text{c}}_{\text{valid}}$ (Fig. A.2)

$$(D^{d'd''}_{corner}\varphi^{\text{c}})_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d'}+\frac{1}{2}\boldsymbol{e}^{d''}} = \begin{cases} \frac{1}{h^2}(\varphi_{\boldsymbol{i}+\boldsymbol{e}^{d'}+\boldsymbol{e}^{d''}} + \varphi_{\boldsymbol{i}} - \varphi_{\boldsymbol{i}+\boldsymbol{e}^{d'}} - \varphi_{\boldsymbol{i}+\boldsymbol{e}^{d''}}) & \text{if } [\boldsymbol{i}, \boldsymbol{i} + \boldsymbol{e}^{d'} + \boldsymbol{e}^{d''}] \subset \Omega^{\text{c}}_{\text{valid}}, \\ 0 & \text{otherwise}, \end{cases}$$

$$(D^{2,d'd''}\varphi^{\text{c}})_{\boldsymbol{i}} = \begin{cases} \frac{1}{N_{\text{valid}}} \sum_{s'=\pm 1} \sum_{s'=\pm 1} (D^{d'd''}\varphi^{\text{c}})_{\boldsymbol{i}+\frac{1}{2}s'\boldsymbol{e}^{d'}+\frac{1}{2}s'\boldsymbol{e}^{d''}} & \text{if } N_{\text{valid}} > 0, \\ 0 & \text{otherwise}, \end{cases}$$

where N_{valid} is the number of nonzero summands. To compute (ii), we need to compute the interpolation coefficients a b, and c.

$$a = \frac{\hat{\varphi} - (n_{\text{ref}} \cdot |x_d| + 2)\varphi_{\boldsymbol{i}\mp\boldsymbol{e}^d} + (n_{\text{ref}} \cdot |x_d| + 1)\varphi_{\boldsymbol{i}\mp 2\boldsymbol{e}^d}}{(n_{\text{ref}} \cdot |x_d| + 2)(n_{\text{ref}} \cdot |x_d| + 1)},$$
$$b = \varphi_{\boldsymbol{i}\mp\boldsymbol{e}^d} - \varphi_{\boldsymbol{i}\mp 2\boldsymbol{e}^d} - ac = \varphi_{\boldsymbol{i}\mp 2\boldsymbol{e}^d}.$$

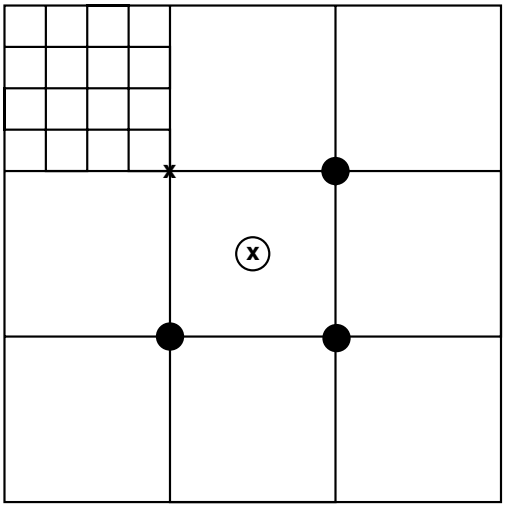

Fig. A.2. Mixed-derivative approximation illustration. The upper-left corner is covered by a finer level so the mixed derivative in the upper left (the uncircled ×) has a stencil which extends into the finer level. We therefore average the mixed derivatives centered on the other corners (the filled circles) to approximate the mixed derivatives for coarse–fine interpolation in three dimensions.

References

[1] A.S. Almgren, J.B. Bell, P. Colella, L.H. Howell, M. Welcome, A conservative adaptive projection method for the variable density incompressible Navier–Stokes equations, J. Comput. Phys. 142 (1) (1998) 1–46.

[2] Ann S. Almgren, Thomas Buttke, Phillip Colella, A fast vortex method in three dimensions, J. Comput. Phys. 113 (2) (1994) 177–200.

[3] A.S. Almgren, J.B. Bell, W. Crutchfield, Approximate projection methods: Part I. inviscid analysis, SIAM J. Sci. Comput. 22 (4) (2000) 1139–1159.

[4] Christopher Anderson, Claude Greengard, The vortex ring merger problem at infinite Reynolds number, Commun. Pure Appl. Math. 42 (1989) 1123–1139.

[5] Yuko Ashima, Saburo Asaka, Interaction of two vortex rings along parallel axes in air, J. Phys. Soc. Jpn. 42 (2) (1977) 708–713.

[6] M. Barad, P. Colella, A fourth-order accurate local refinement method for Poisson's equation, JCP 209 (2005) 1–18.

[7] J.B. Bell, P. Colella, H.M. Glaz, A second-order projection method for the incompressible Navier–Stokes equations, J. Comput. Phys. 85 (1989) 257–283.

[8] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1) (1989) 64–84.

[9] H.D. Ceniceros, A.M. Roma, Study of the long-time dynamics of a viscous vortex sheet with a fully adaptive nonstiff method, Phys. Fluids 16 (2004) 4285–4318.

[10] A.J. Chorin, Numerical solutions of the Navier–Stokes equations, Math. Comp. 22 (1968) 745–762.

[11] P. Colella, D. Graves, B. Keen, D. Modiano, A Cartesian grid embedded boundary method for hyperbolic conservation laws, J. Comput. Phys. 211 (2006) 347–366.

[12] Phillip Colella, Multidimensional upwind methods for hyperbolic conservation laws, J. Comput. Phys. 87 (1990) 71–200.

[13] M.S. Day, J.B. Bell, Numerical simulation of laminar reacting flows with complex chemistry, Combust. Theor. Model. 4 (2000) 535–556.

[14] P.M. Gresho, R.L. Sani, On pressure boundary conditions for the incompressible Navier–Stokes equations, Int. J. Numer. Methods Fluids 7 (1987) 1111–1145.

[15] Boyce E. Griffith, Richard D. Hornung, David M. McQueen, Charles S. Peskin, An adaptive, formally second order accurate version of the immersed boundary method, J. Comput. Phys. 223 (1) (2007) 10–49.

[16] Hans Svend Johansen, Cartesian grid embedded boundary finite difference methods for elliptic and parabolic partial differential equations on irregular domains, PhD Thesis, University of California, Berkeley, 1997.

[17] Egon Krause, On vortex loops and filaments: three examples of numerical predictions of flows containing vortices, Naturwissenschaften (90) (2003) 4–26.

[18] T.T. Lim, An experimental study of a vortex ring interacting with an inclined wall, Exp. Fluids 7 (1989) 453–463.

[19] D. Martin, P. Colella, A cell-centered adaptive projection method for the incompressible Euler equations, J. Comput. Phys. 163 (2) (2000) 271–312.

[20] P. McCorquodale, P. Colella, H. Johansen, A Cartesian grid embedded boundary method for the heat equation on irregular domains, JCP 173 (2) (2001) 620–635.

[21] Michael L. Minion, On the stability of Godunov-projection methods for incompressible flow, J. Comput. Phys. 123 (2) (1996) 435–449.

[22] Michael L. Minion, A projection method for locally refined grids, J. Comput. Phys. 127 (1) (1996) 158–178.

[23] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries, J. Comput. Phys. 190 (2003) 572–600.
[24] R. Propp, P. Colella, W.Y. Crutchfield, M.S. Day, A numerical model for trickle-bed reactors, JCP 165 (2000) 311–333.
[25] A.M. Roma, C.S. Peskin, M.J. Berger, An adaptive version of the immersed boundary method, J. Comput. Phys. 153 (1999) 509–534.
[26] Jeff Saltzman, An unsplit 3d upwind method for hyperbolic conservation laws, J. Comput. Phys. 115 (1994) 153–168.
[27] R. Samtaney, P. Colella, S.C. Jardin, D.F. Martin, 3D adaptive mesh refinement simulations of pellet injection in tokamaks, Comput. Phys. Commun. 164 (2004) 220–228.
[28] M. Sussman, A parallelized, adaptive algorithm for multiphase flows in general geometries, Comput. Struct. 83 (2005) 435–444.
[29] M.C. Thompson, J.H. Ferziger, An adaptive multigrid technique for the incompressible Navier–Stokes equations, J. Comput. Phys. 82 (1) (1989) 94–121.
[30] E.H. Twizell, A.B. Gumel, M.A. Arigu, Second-order, L_0-stable methods for the heat equation with time-dependent boundary conditions, Adv. Comput. Math. 6 (1996) 333–352.