A Cell-Centered Adaptive Projection Method for the Incompressible Navier-Stokes Equations in Three Dimensions

Daniel F. Martin *,[1] Phillip Colella [1] Daniel T. Graves [2]

Lawrence Berkeley National Laboratory, Berkeley, CA

Abstract

We present an extension of a previously presented method to compute locally adaptive solutions for incompressible viscous flows in three dimensions using block-structured local refinement in both space and time. Like the previous work, this method uses a projection formulation based on a cell-centered approximate projection, and performs a set of synchronization operations to maintain solution accuracy in the presence of refinement in time. We use an L_0-stable second-order semi-implicit scheme to evaluate the viscous terms.

Key words: 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 and single-level operators. Also, an advection-velocity

* Corresponding Author
Email address: DFMartin@lbl.gov (Daniel F. Martin).

[1] 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.
[2] Research supported by the Office of Advanced Scientific Computing Research of the US Department of Energy under contract number DE-AC02-05CH11231.

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 sybcycling in time [21,28,22,9], instead opting to advance all levels with a uniform timestep. Extension of the non-subcycled schemes to multiphase flows was done in [27], and to the immersed-boundary method in [24,15]. Almgren et al [1] presented an algorithm for the solution of the three-dimensional incompressible Navier-Stokes equations 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]. Another feature in our algorithmic choices has been the choice to use fully multilevel (rather than single-level) elliptic solvers wherever possible. 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]. While developing and maintaining two sets of elliptic solvers for the regular Cartesian grid case requires a fairly large development effort, doing this for embedded-boundary computations would represent an overwhelming amount of effort. 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.
- **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]. Instead, we employ a second-order semi-implicit Runge-Kutta scheme based on the L_0-stable scheme in [29], which eliminated such instabilities.
- **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.

- **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. The advantages of the approach used in this work are that it produces a conceptually simpler algorithm and avoids interpolation of corrections to finer levels (corrections are naturally computed for all levels through the multilevel solves). Disadvantages to our approach are the need for multilevel elliptic solvers and that our approach tolerates small deviations from freestream preservation, which are generally confined to cells adjacent to coarse-fine interfaces. [19].

1.1 Formulation of the Problem

We are solving the incompressible constant-density Navier-Stokes equations with a passively-advected scalar s:

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

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

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

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

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

$$\mathbb{P}(\vec{w}) = \vec{w}_d \tag{5}$$

and is computed by solving an elliptic equation for ϕ and then subtracting $\nabla\phi$:

$$\mathbb{P}(\vec{w}) = (\vec{I} - \nabla(\Delta^{-1})\nabla\cdot)\vec{w}. \tag{6}$$

Using the projection operator, we transform the constrained problem (1-2)

into an evolution equation:

$$\frac{\partial \vec{u}}{\partial t} = \mathbb{P}(-(\vec{u}\cdot\nabla)\vec{u} + \nu\Delta\vec{u}) \tag{7}$$

$$\nabla\cdot\vec{u}(\cdot, t=0) = 0. \tag{8}$$

1.1.1 Viscous term discretization

The heat equation with a source term may be written as:

$$\frac{\partial q}{\partial t} = L(q) + f \tag{9}$$

where L is a discrete second-order elliptic operator. Following [29], we discretize (9) as

$$q^{n+1} = (I-\mu_1 L)^{-1}(I-\mu_2 L)^{-1}\Big[(i+\mu_3 L)q^n + (I+\mu_4)f^{n+\frac{1}{2}}\Big], \tag{10}$$

where $f^{n+\frac{1}{2}} = f\Big((n+\frac{1}{2})\Delta t\Big)$, $q^n = q(n\Delta t)$, and the coefficients $\mu_1, \mu_2, \mu_3, \mu_4$ are the values suggested in [29]:

$$\begin{aligned}
\mu_1 &= \frac{2a-1}{a+discr}\Delta t,\\
\mu_2 &= \frac{2a-1}{a-discr}\Delta t,\\
\mu_3 &= (1-a)\Delta t,\\
\mu_4 &= (\tfrac{1}{2}-a)\Delta t\\
a &= 2-\sqrt{2}-\epsilon,\\
discr &= \sqrt{a^2-4a+2},
\end{aligned}$$

where ϵ is a small quantity (we use 10^{-8}). The treatment of the source term presented here differs from that in [29] due to the different time-centerings of the source term; the source terms in [29] are centered at the old and new times, while the source term $f^{n+\frac{1}{2}}$ in this work is centered at the half-time. We use this to define the diffusive term $L^{TGA}(q^n, f^{n+\frac{1}{2}})$ as follows:

$$\begin{aligned}
L^{TGA}(q^n, f^{n+\frac{1}{2}}) &= \frac{q^{n+1}-q^n}{\Delta t} - f^{n+\frac{1}{2}} \\
&\approx (Lq)\Big((n+\tfrac{1}{2})\Delta t\Big) + O(\Delta t^2)
\end{aligned} \tag{11}$$

Note that L^{TGA} may be written in conservative form:

$$L^{TGA}(q^n, f^{n+\frac{1}{2}}) = Div(\vec{F^{TGA}}) \tag{12}$$

1.1.2 Time discretization

As in [19], our algorithm is a predictor-corrector formulation in which we first compute an intermediate velocity field and then project it onto the space of vectors which satisfy the divergence constraint. Updates to the scalar s are computed using a conservative update. The intermediate velocity field $\vec{u}^*$ is computed as an approximation to $\vec{u}(t+\Delta t)$:

$$\vec{u}^* = \vec{u}(t) - \Delta t\Big[(\vec{u}\cdot\nabla)\vec{u}\Big]^H + \Delta t\nu L^{TGA}(\vec{u}(t), f^{SRC}) - \Delta t\nabla p(t-\frac{\Delta t}{2}), \quad (13)$$

where the superscript H indicates centering at the half time $(t+\frac{\Delta t}{2})$. The notation $\nu L^{TGA}(\vec{u}(t), f^{RHS})$ indicates that the viscous terms are computed using the semi-implicit method detailed in section 1.1.1, where $f^{SRC} = -\Big[(\vec{u}\cdot\nabla)\vec{u}\Big]^H - \nabla p(t-\frac{\Delta t}{2})$. The updated velocity field is then computed by projecting the intermediate velocity field onto the space of divergence-free vectors:

$$\vec{u}(t+\Delta t) = \mathbb{P}(\vec{u}^* + \Delta t\nabla p(t-\frac{\Delta t}{2})) \quad (14)$$

$$\nabla p(t+\frac{\Delta t}{2}) = \frac{1}{\Delta t}(\mathbb{I}-\mathbb{P})\Big(\vec{u}^* + \Delta t\nabla p(t-\frac{\Delta t}{2})\Big) \quad (15)$$

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 [2] supports this choice of formulations. Also, while in our approach the pressure is only first-order accurate in time, it can be made more accurate using the ideas in [8].

1.2 AMR Notation

In this work, we use the same notation as in [19]. Following [7], our adaptive mesh calculations are performed on a hierarchy of nested, cell-centered grids (Figure 1). At each AMR level $\ell = 0, ..., \ell_{max}$, 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 $n^\ell_{ref} = \frac{h_\ell}{h_{\ell+1}}$ 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 $\mathcal{C}_r(i,j) = (\lfloor\frac{i}{r}\rfloor, \lfloor\frac{j}{r}\rfloor)$. In that 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 two. Calculations are performed on

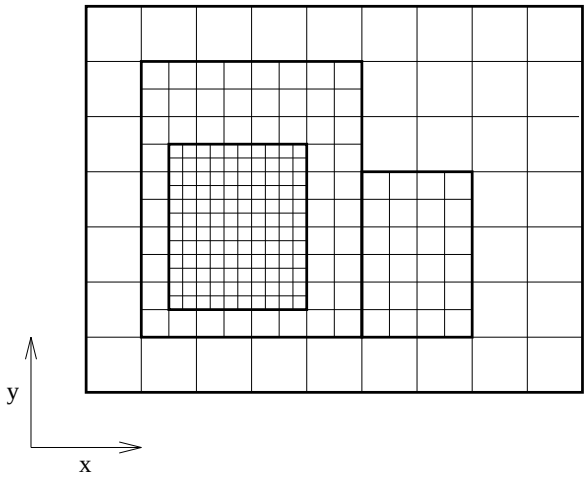


Fig. 1. Block-structured local refinement. Note that refinement is by an integer factor and is organized into rectangular patches.

a hierarchy of meshes $\Omega^\ell \subset \Gamma^\ell$, with $\Omega^\ell \supset \mathcal{C}_{n^\ell_{ref}}(\Omega^{\ell+1})$. 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 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: $\Omega^\ell_{valid} = \Omega^\ell - \mathcal{C}_{n^\ell_{ref}}(\Omega^{\ell+1})$. The grids on each level satisfy a *proper nesting* condition [7]: 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 $\Omega^{\ell,*}_{valid}$ 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: $\mathcal{C}_{n^\ell_{ref}}(\Omega^{\ell+1,*})$ 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 $\vec{u}^{face,comp}$ 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.

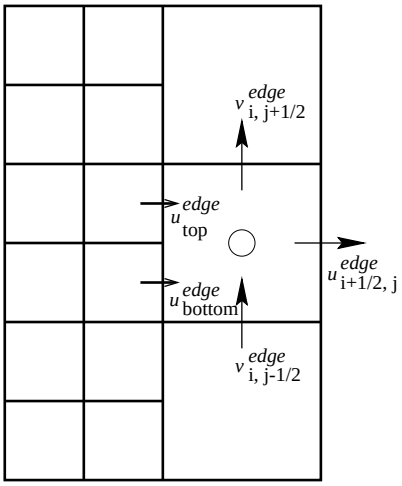


Fig. 2. Sample coarse-fine interface with a face-centered vector field. Cell (i,j) (open circle) is to the right of the coarse-fine interface.

It is also necessary to transfer information from finer grids to coarser ones. 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.

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 $\vec{u}^{face,\ell}$ 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 $D^{comp,\ell}(\vec{u}^{face,\ell}, \vec{u}^{face,\ell+1})$; the level operator only depends on level ℓ data: $D^{\ell}(\vec{u}^{face,\ell})$.

Assume that the vector field $\vec{u}^{face,\ell}$ 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 $D^{comp}\vec{u}^{face,comp}$ 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 $\mathcal{C}_{n^{\ell}_{ref}}(\partial\Omega^{\ell+1,*})$, which stores the difference in the face-centered quantity $\vec{u}^{face}$ 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:

$$(D^{comp,\ell}\vec{u}^{face})_{\boldsymbol{i}} = (D^{\ell}\vec{u}^{face,\ell})_{\boldsymbol{i}} + D_R^{\ell}(\delta\vec{u}^{\ell+1})_{\boldsymbol{i}}, \tag{16}$$

$$\delta\vec{u}^{\ell+1} = \langle\vec{u}^{face,\ell+1}\rangle - \vec{u}^{face,\ell} \quad \text{on} \quad \mathcal{C}_{n_{ref}^{\ell}}(\partial\Omega^{\ell+1,*}).$$

For the level ℓ cell $(\boldsymbol{i})$, D_R^{ℓ} can be defined as:

$$D_R^{\ell}(\delta\vec{u}^{\ell+1})_{\boldsymbol{i}} = \frac{1}{h_\ell}\sum_p \pm(\delta\vec{u}^{\ell})_p, \tag{17}$$

where the sum is over the set of all faces of cell $(\boldsymbol{i})$ 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 $(\boldsymbol{i})$, 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 a face-centered, centered-difference gradient of a cell-centered variable ϕ. $G^{comp}\phi$ is a composite vector field, defined on all valid faces in the multilevel domain. To compute $G^{comp}\phi$ 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 appendix A. We denote this quadratic coarse-fine interpolation operator as $I(\phi^{\ell},\phi^{\ell-1})$:

$$\phi^{\ell} = I(\phi^{\ell},\phi^{\ell-1}) \quad \text{on} \quad \partial\Omega^{\ell} \tag{18}$$

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 $\Omega^{\ell,*}_{valid}$) 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, $G^{comp,\ell}$, is dependent on level ℓ and coarse-level $(\ell-1)$ data: $G^{comp,\ell}(\phi^{\ell},\phi^{\ell-1})$. 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:

$$L^{comp}\phi^{comp} = D^{comp}G^{comp}\phi^{comp} \tag{19}$$

$$L^\ell \phi^\ell = D^\ell G^\ell \phi^\ell. \tag{20}$$

On the interiors of grids, (19) and (20) 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: $L^{comp,\ell}(\phi^\ell, \phi^{\ell+1}, \phi^{\ell-1})$ and $L^\ell(\phi^\ell, \phi^{\ell-1})$.

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 $Av^{C\to F}$ and $Av^{F\to C}$. 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:

$$D^{CC,comp}\vec{u}^{CC} = D^{comp}(Av^{C\to F}\vec{u}^{CC}). \tag{21}$$

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

$$G^{CC,comp}\phi = Av^{F\to C} G^{comp}\phi. \tag{22}$$

The level-operator divergence and gradient operators are defined similarly.

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 [7,19], 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 $\Delta t^{\ell+1} = \frac{\Delta t^\ell}{n^\ell_{ref}}$. Once level $\ell + 1$ (and 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 velocity field $\vec{u}^\ell(\vec{x}, t^\ell) = (u^\ell, v^\ell, w^\ell)^T$, an advected scalar $s^\ell(\vec{x}, t^\ell)$, the freestream preservation scalar $\Lambda^\ell(\vec{x}, t^\ell)$, and the staggered-grid freestream preservation correction $\vec{u}_p$ from the most recent synchronization step, which has been extended to the invalid regions on level ℓ with $\langle \vec{u}_p^{\ell+1} \rangle$. We also have the lagged approximation to the pressure $\pi^{n-\frac{1}{2}}$.

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 δs^ℓ and $\delta\Lambda^\ell$ contain the fluxes of the advected scalars s and Λ.

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 Figure 3.

(1) **Compute advection velocities** We compute a set of upwinded face-centered velocities $\vec{u}_G^{half}$ at time $t^\ell + \frac{1}{2}\Delta t^\ell$, using the unsplit scheme outlined in Appendix B. Note that the source term used in the upwind scheme is $S = \Delta t^\ell(-G\pi^\ell + \nu L^\ell \vec{u}^\ell(t^\ell))$.

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

$$L^\ell \phi = D^\ell \vec{u}_G^{half}. \tag{23}$$

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

$$\phi^\ell = I(\phi^\ell, \frac{\Delta t}{2}\pi^{\ell-1}). \tag{24}$$

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:

$$\vec{u}^{half,\ell} = \vec{u}_G^{half} - G^\ell\phi^\ell. \tag{25}$$

Coarse-fine boundary conditions for computing $G^\ell\phi^\ell$ are given by (24).

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

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

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

Predict $\vec{u}^{\text{half}}$

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

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

Project $\vec{u}^{*,\ell}\to\vec{u}^\ell(t^\ell+\Delta t^\ell)$:
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 $(\ell < \ell_{\max})$
$\Delta t^{\ell+1} = \frac{1}{n^\ell_{\text{ref}}}\Delta t^\ell$
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**$(\ell, t^\ell+\Delta t^\ell, t^\ell)$
end if
end NSLevelAdvance

Fig. 3. Recursive level time step for the incompressible Navier-Stokes equations.

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

$$\vec{u}^{\ell}_{AD} = \vec{u}_{half,\ell} + \vec{u}^{\ell}_{p} \tag{26}$$

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

$$[s,\Lambda]^{\ell}(t^{\ell}+\Delta t^{\ell}) = [s,\Lambda]^{\ell} - \Delta t^{\ell} D^{\ell}(\vec{u}^{\ell}_{AD}[s,\Lambda]^{half,\ell}). \tag{27}$$

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

$$[(\vec{u}\cdot\nabla)\vec{u}]^{half,\ell} = Av^{F\rightarrow C}(\vec{u}^{\ell}_{AD})\cdot(G^{\ell}\vec{u}^{half,\ell}). \tag{28}$$

(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 discretization described in Section 1.1.1. The update proceeds as follows:

(a) *Compute diffused source term*

$$\vec{f^{*}} = -[(\vec{u}\cdot\nabla)\vec{u}]^{\frac{1}{2},\ell} - G^{CC,\ell}\pi^{\ell} \tag{29}$$

$$\vec{f} = \Delta t^{\ell}(I+\mu^{\ell}_{4}\nu L^{\ell})\vec{f^{*}}, \tag{30}$$

where coarse-fine boundary conditions for the computation of $\vec{f^{\ell}}$ are given by higher-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 $\vec{u}^{\ell}_{e}$:

$$(I-\mu^{\ell}_{2}\nu L^{\ell})\vec{u}^{\ell}_{e} = \vec{u}^{\ell} + \mu^{\ell}_{3}\nu L^{\ell}\vec{u}^{\ell}(t^{\ell}) + \vec{f} \tag{31}$$

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

$$\vec{u}^{\ell}_{e} = I(\vec{u}^{\ell}_{e}, \vec{u}^{\ell-1}(t^{\ell}+(\Delta t^{\ell}-\mu^{\ell}_{1})) \tag{32}$$

(c) *Solve for* $\vec{u}^*$
A second solve is then computed for the intermediate velocity $\vec{u}^*$:

$$(I - \mu_1^\ell \nu L^\ell)\vec{u}^{*,\ell} = \vec{u}_e^\ell, \tag{33}$$

with coarse-fine boundary conditions (if required):

$$\vec{u}^{*,\ell} = I(\vec{u}^{*,\ell}, \vec{u}^{\ell-1}(t^\ell + \Delta t^\ell)). \tag{34}$$

(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: For convenience, define the viscous flux F^{TGA}:

$$F^{TGA,\ell} = (\tfrac{1}{2} - a)\nu\Delta t^\ell G^\ell(f^{*,\ell}) - \nu G^\ell(r_1\vec{u}^{*,\ell} + r_2\vec{u}_e^\ell + (1-a)\vec{u}^\ell(t^\ell)) \tag{35}$$

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

$$\begin{aligned}
\delta\vec{V}^{\ell+1} &:= -\vec{u}_{AD}^\ell \vec{u}^{half,\ell} + F^{TGA,\ell} \\
\delta s^{\ell+1} &:= -\vec{u}_{AD}^\ell s^{half,\ell} \\
\delta\Lambda^{\ell+1} &:= -\vec{u}_{AD}^\ell \Lambda^{half,\ell}
\end{aligned}$$

- *if* $(\ell > 0)$

$$\begin{aligned}
\delta\vec{V}^\ell &:= \delta\vec{V}^\ell + \frac{1}{n_{ref}^{\ell-1}}\left\langle \vec{u}_{AD}^\ell \vec{u}^{half,\ell} \right\rangle - \frac{1}{n_{ref}^{\ell-1}}\left\langle F^{TGA,\ell} \right\rangle \\
\delta s^\ell &:= \delta s^\ell - \frac{1}{n_{ref}^{\ell-1}}\langle \vec{u}_{AD}^\ell s^{half,\ell} \rangle \\
\delta\Lambda^\ell &:= \delta\Lambda^\ell - \frac{1}{n_{ref}^{\ell-1}}\langle \vec{u}_{AD}^\ell \Lambda^{half,\ell} \rangle
\end{aligned}$$

(6) **Project** $\vec{u}^{*,\ell} \to \vec{u}^\ell(t^\ell + \Delta t^\ell)$ In the same way as in [19], the cell-centered level-operator projection $\mathbb{P}^{CC,\ell}$ is applied to the intermediate velocity field $\vec{u}^{*,\ell}$. First, solve for the approximation to the pressure $\pi^\ell(t^\ell + \frac{1}{2}\Delta t^\ell)$:

$$L^\ell \pi^\ell(t^\ell + \frac{\Delta t^\ell}{2}) = \frac{1}{\Delta t} D^{CC,\ell}(\vec{u}^{*,\ell} + \Delta t^\ell G^{CC,\ell}\pi^\ell(t^\ell - \frac{\Delta t^\ell}{2}) \tag{36}$$

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

$$\vec{u}^{*,\ell} = I(\vec{u}^{*,\ell}, \vec{u}^{\ell-1} + \Delta t^\ell G^{\ell-1}\pi^{\ell-1}). \tag{37}$$

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:

$$\pi^\ell = I(\pi^\ell, \pi^{\ell-1}). \tag{38}$$

Then, the velocity field is corrected:

$$\vec{u}^\ell(t^\ell + \Delta t^\ell) = (\vec{u}^{*,\ell} + \Delta t^\ell G^{CC,\ell}\pi^\ell(t^\ell - \frac{\Delta t^\ell}{2})) - \Delta t^\ell G^{CC,\ell}\pi^\ell(t^\ell + \frac{\Delta t^\ell}{2}) \tag{39}$$

(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 $\Delta t^{\ell+1} = \frac{1}{n^\ell_{ref}}\Delta t^\ell$. 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.

2.3 Synchronization

Synchronization is an essentially multilevel operation which ties together the different AMR levels after they have been advanced fairly 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 $\ell \geq \ell_{base}$, where ℓ_{base} is 1. A pseudocode representation of the synchronization operations is presented in Figure 4.

(1) **Reflux for conservation** To preserve conservation, the flux mismatch stored in the flux registers $\delta\vec{V}$, $\delta\Lambda$, and δS is used to correct the solution along the coarse side of coarse-fine interfaces. For non-diffusive scalars (Λ and s), we apply the correction explicitly, as in [19]:

$$[s, \Lambda]^\ell = [s, \Lambda]^\ell - \Delta t^\ell D^\ell_R([\delta S, \delta\Lambda]^{\ell+1}) \quad \text{for} \quad \ell \geq \ell_{base}. \tag{40}$$

Since the momentum flux correction will contain diffusive fluxes, stability considerations require that we apply this correction implicitly. We

Synchronize($\ell_{\rm base}, t^{\rm sync}, \Delta t^{\rm sync}$)

Reflux for conservation:

for $\ell = \ell_{\max} - 1, \ell_{\rm base}, -1$

$s^\ell(t^{\rm sync}) := s^\ell(t^{\rm sync}) - \Delta t^\ell D_R^\ell(\delta s^{\ell+1})$

$\Lambda^\ell(t^{\rm sync}) := \Lambda^\ell(t^{\rm sync}) - \Delta t^\ell D_R^\ell(\delta \Lambda^{\ell+1})$

end for

$(\boldsymbol{I} - \nu \Delta t^{\ell_{base}} L^{comp}) \delta \vec{u} = -\Delta t^\ell D_R^\ell(\delta \mathbf{V}^{\ell+1})$ for $\ell \geq \ell_{base}$

$\vec{u}^\ell(t^{\rm sync}) := \vec{u}^\ell(t^{\rm sync}) + \delta \vec{u}$

Apply Synchronization Projection:

Solve $L^{\rm comp} e_s = \frac{1}{\Delta t^{\rm sync}} D^{\rm CC,comp} \vec{u}(t^{\rm sync})$ for $\ell \geq \ell_{\rm base}$

$e_s^{\ell_{\rm base}} = I(e_s^{\ell_{\rm base}}, e_s^{\ell_{\rm base}-1})$

$\vec{u}(t^{\rm sync}) := \vec{u}(t^{\rm sync}) - \Delta t^{\rm sync} G^{\rm CC,comp} e_s$ for $\ell \geq \ell_{\rm base}$

Freestream Preservation Solve:

Solve $L^{\rm comp} e_\Lambda = \frac{(\Lambda(t^{\rm sync})-1)}{\Delta t^{\rm sync}} \eta$ for $\ell \geq \ell_{\rm base}$

$e_\Lambda^{\ell_{\rm base}} = I(e_\Lambda^{\ell_{\rm base}}, e_\Lambda^{\ell_{\rm base}-1})$

$\vec{u}_{\rm p} = G^{\rm comp} e_\Lambda$

Average finer solution onto coarser levels:

for $\ell = \ell_{\max} - 1, \ell_{\rm base}, -1$

$\vec{u}^\ell(t^{\rm sync}) = \langle \vec{u}^{\ell+1}(t^{\rm sync}) \rangle$ on $\mathcal{C}_{n_{\rm ref}^\ell}(\Omega^{\ell+1})$

$s^\ell(t^{\rm sync}) = \langle s^{\ell+1}(t^{\rm sync}) \rangle$ on $\mathcal{C}_{n_{\rm ref}^\ell}(\Omega^{\ell+1})$

$\Lambda^\ell(t^{\rm sync}) = \langle \Lambda^{\ell+1}(t^{\rm sync}) \rangle$ on $\mathcal{C}_{n_{\rm ref}^\ell}(\Omega^{\ell+1})$

end for

end Synchronize

Fig. 4. Synchronization for incompressible Navier-Stokes equations.

first solve a Helmholtz equation for a correction:

$$(I - \nu \Delta t^{\ell_{base}} L^{comp}) \delta \vec{u} = \Delta t^\ell D_R(\delta \vec{V}^{\ell+1}) \quad \text{for} \quad \ell \geq \ell_{base} \tag{41}$$

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 $(\ell_{base} - 1)$, the coarse-fine boundary condition for $\delta \vec{u}$ is quadratic interpolation with 0's on the coarser level $(\ell_{base}-1)$: $\delta \vec{u}^{\ell_{base}} = I(\delta \vec{u}^{\ell_{base}}, \delta \vec{u}^{\ell_{base}-1} = 0)$. Then, the correction is added to the velocity field

$$\vec{u}^\ell := \vec{u}^\ell + \delta \vec{u}^\ell \quad \text{for} \quad \ell \geq \ell_{base}. \tag{42}$$

(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:

$$L^{comp} e_s = D^{CC,comp} \vec{u}^{comp} \quad \text{for} \quad \ell \geq \ell_{\text{base}}. \tag{43}$$

As in [19], physical domain boundary conditions are the homogeneous form of those used for the level projection. If $\ell_{base} > 0$, coarse-fine boundary conditions are required. When computing $D^{CC,comp}\vec{u}^{comp}$, the coarse-fine boundary condition is quadratic interpolation with the coarser-level velocity field, linearly interpolated in time to t^{sync}: $\vec{u}^{\ell_{base}} = I(\vec{u}^{\ell_{base}}, \vec{u}^{\ell_{base}-1}(t^{sync}))$. The coarse-fine boundary condition for the elliptic solve is $e_s^{\ell_{base}} = I(e_s^{\ell_{base}}, \Delta t^{sync} e_s^{\ell_{base}-1})$.

Once the correction has been computed, the velocity field is corrected:

$$\vec{u}^{\ell} := \vec{u}^{\ell} - G^{CC,comp} e_s^{\ell} \quad \text{for} \quad \ell \geq \ell_{base}. \tag{44}$$

If $\ell_{base} > 0$, the coarse-fine boundary condition for computing the gradient is $\mathrm{e}_s^{\ell_{base}} = I(e_s^{\ell_{base}}, \Delta t^{sync} e_s^{\ell_{base}-1})$. For all steps in the composite projection, physical boundary conditions are the homogeneous form of those used for the single-level projection.

(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

$$L^{comp} e_\Lambda = \eta \frac{(\Lambda - 1)}{\Delta t^{\ell_{base}}} \quad \text{for} \quad \ell \geq \ell_{base}. \tag{45}$$

If required, the coarse-fine boundary condition is quadratic interpolation: $e_\Lambda^{\ell_{base}} = I(e_\Lambda^{\ell_{base}}, e_\Lambda^{\ell_{base}-1})$. Then, a face-centered correction to the advection velocities is computed:

$$\vec{u}_p = G^{comp} e_\Lambda \quad \text{for} \quad \ell \geq \ell_{base}. \tag{46}$$

Coarse-fine boundary conditions for the gradient are the same as those used in the solve: $e_\Lambda^{\ell_{base}} = I(e_\Lambda^{\ell_{base}}, e_\Lambda^{\ell_{base}-1})$.

(4) **Average fine solution onto coarser grids** Finally, all quantities on covered regions, including the face-centered $\vec{u}_p$, 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

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

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

  Initialize π:
    $\pi^{\ell} = 0$ for $\ell > \ell_{base}$
    for $n = 1, n_{passes}$
      $\widetilde{\Delta t} = \frac{\Delta t^{\ell_{max}}}{2}$
      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. 5. Initialization for incompressible Navier-Stokes equations.

variables π and e_Λ (along with $\vec{u}_p$), which are required for the single-level updates. A pseudocode description of the initialization procedure appears in Figure 5. For initialization, ℓ_{base} is the finest *unchanged* level (at the initial time, $\ell_{base} = -1$). $\Delta t^{\ell_{base}}$ 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:

$$\phi^{\ell_{base}+1} = I(\phi^{\ell_{base}+1}, 0^{\ell_{base}}). \tag{47}$$

(2) **Initialize Freestream Preservation Correction** While the freestream preservation correction need not be initialized at the start of the computation, it must be re-computed after regridding. First, an elliptic equation is solved for the potential e_Λ:

$$L^{comp} e_\Lambda = \frac{\Lambda(t^{init}) - 1}{\Delta t^{\ell_{base}}}\eta \quad \text{for} \quad \ell \geq \ell_{base}. \tag{48}$$

If $\ell_{base} > 0$, then the coarse-fine boundary condition for e_Λ in the elliptic solve is homogeneous quadratic interpolation: $e_\Lambda^{\ell_{base}} = I(e^{\ell_{base}}, 0^{\ell_{base}-1})$. The face-centered freestream-preservation correction is then given by:

$$\vec{u}_p = G^{comp} e_\Lambda. \tag{49}$$

Coarse-fine boundary conditions on e_Λ when computing $\vec{u}_p$ are also $e_\Lambda^{\ell_{base}} = I(e^{\ell_{base}}, 0^{\ell_{base}-1})$.

(3) **Initialize** π During the single-level update, computation of the intermediate velocity field $\vec{u}^*$ uses the lagged level pressure $\pi^\ell(t^\ell - \frac{\Delta t^\ell}{2})$. 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 ($\widetilde{\Delta t} = \frac{\Delta t^{\ell_{max}}}{2}$). 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. However, 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.

Then, we project $\widetilde{\vec{u}^*}$ to compute π, solving

$$L^\ell \pi^\ell = \frac{1}{\widetilde{\Delta t}} D^{CC,\ell}(\widetilde{u^*}) \tag{50}$$

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

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

$$\widetilde{\vec{u}^\ell}(t^\ell + \widetilde{\Delta t}) := \widetilde{\vec{u}^*} - \widetilde{\Delta t} G^{CC,\ell} \pi^\ell. \tag{51}$$

3 Results

In order for this method to be useful, it must satisfy three criteria. First, solutions computed with local refinement should converge at 2nd-order rates (just as single-level solutions do). Second, solutions computed with well-placed local refinement should be as accurate as a uniform-mesh solution with the equivalent resolution. Finally, the use of local refinement for appropriate problems should result in significant savings in either computational time or size when compared to the equivalent uniform-mesh solution.

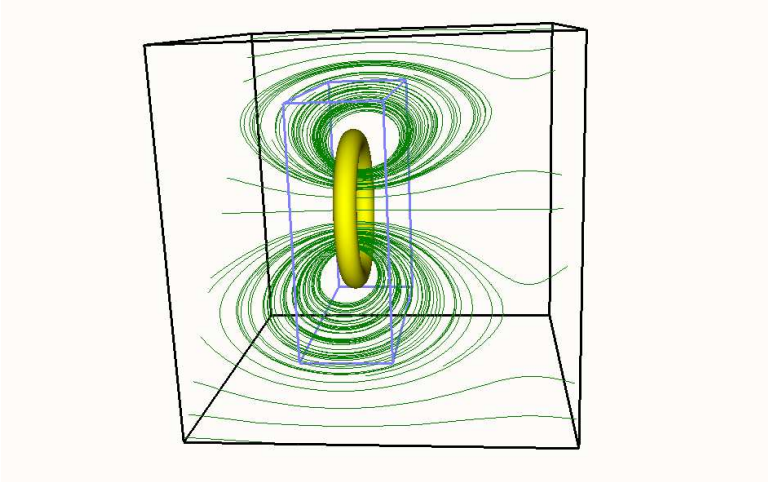


Fig. 6. Vortex ring test problem. Yellow vorticity isosurface depicts location of vortex ring, green lines depict streamlines, blue box is example of refined region. Black box depicts computational domain.

3.1 Convergence and Accuracy

To demonstrate the convergence and accuracy of this approach, we use a single vortex ring in a cubic domain. 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

$$\omega(\rho) = \frac{\Gamma}{a\sigma^2} e^{(\frac{-\rho}{\sigma})^3} \tag{52}$$

where ρ is the local distance from the center of the ring cross-section, $a = 2268.85$, and $\sigma = 0.0275$.

For this problem, 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 Figure 6.

The L_2 norm of error in the $x-$velocity is shown in Table 3.1 (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). The solution converges at second-order rates both with and without local refinement, which indicates that the presence of coarse-fine interfaces is not interfering with convergence. Comparing the errors for equivalent resolutions demonstrates the effectiveness of the local refinement. For example, the error for the 128^3 single-level case should be compared with the 64^3 $n_{ref} = 2$ case,

$\frac{1}{h_0}$	single-level	rate	$n_{ref} = 2$	rate	$n_{ref} = 4$	rate	$nref = (2,2)$	rate
16	1.5101e-04	–	5.1954e-05	–	1.2172e-05	–	1.0350e-05	–
32	4.2104e-05	1.84	1.0468e-05	2.31	2.6984e-06	2.17	2.8654e-06	1.85
64	7.8983e-06	2.41	2.4755e-06	2.08	6.7945e-07	1.99	6.6004e-07	2.12
128	1.8712e-06	2.08	5.6389e-07	2.13	–	–	–	–
256	4.5796e-07	2.03	–	–	–	–	–	–

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

the $32 \times 32 \times 32$ $n_{ref} = 4$ case, and the 32^3 $n_{ref} = (2,2)$ case. The $n_{ref} = (2,2)$ case refers to 2 levels of refinement, each with a factor of 2 refinement. This gives an equivalent resolution to a single $n_{ref} = 4$ 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.

3.2 Computational performance

To demonstrate the performance of the AMR algorithm, we measured the runtimes and total number of cells advanced 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 Figure 7. 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). Note that while the total number of cells advanced for $n_{ref} = 4$ is only slightly smaller than the number of cells advanced for $n_{ref} = 2$, 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 $n_{ref} = 2$ case.

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 [17,3,4] and experimentally in [18,5]. The initial conditions are two vortex rings which are

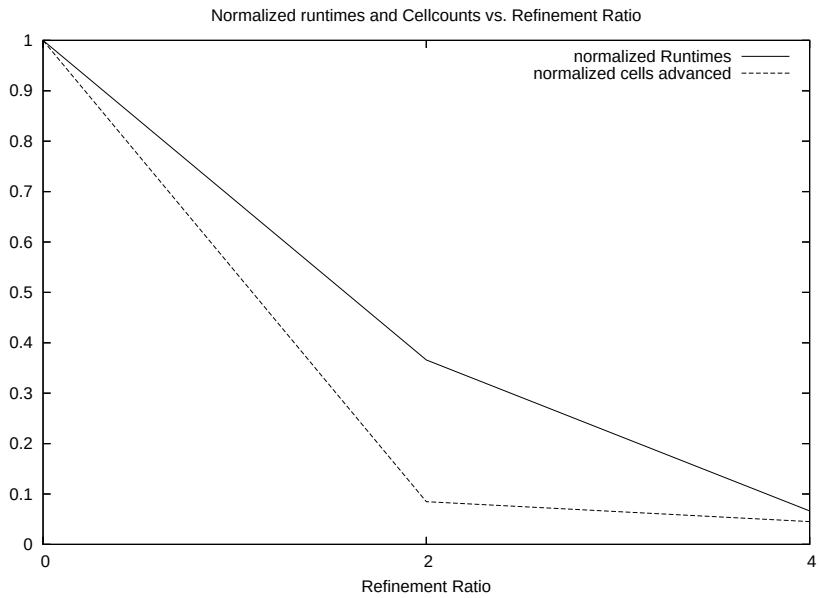


Fig. 7. Normalized run times and number of cells advanced for the 256^3 equivalent resolution vortex-ring problem. Refinement ratio of 0 is the non-AMR case.

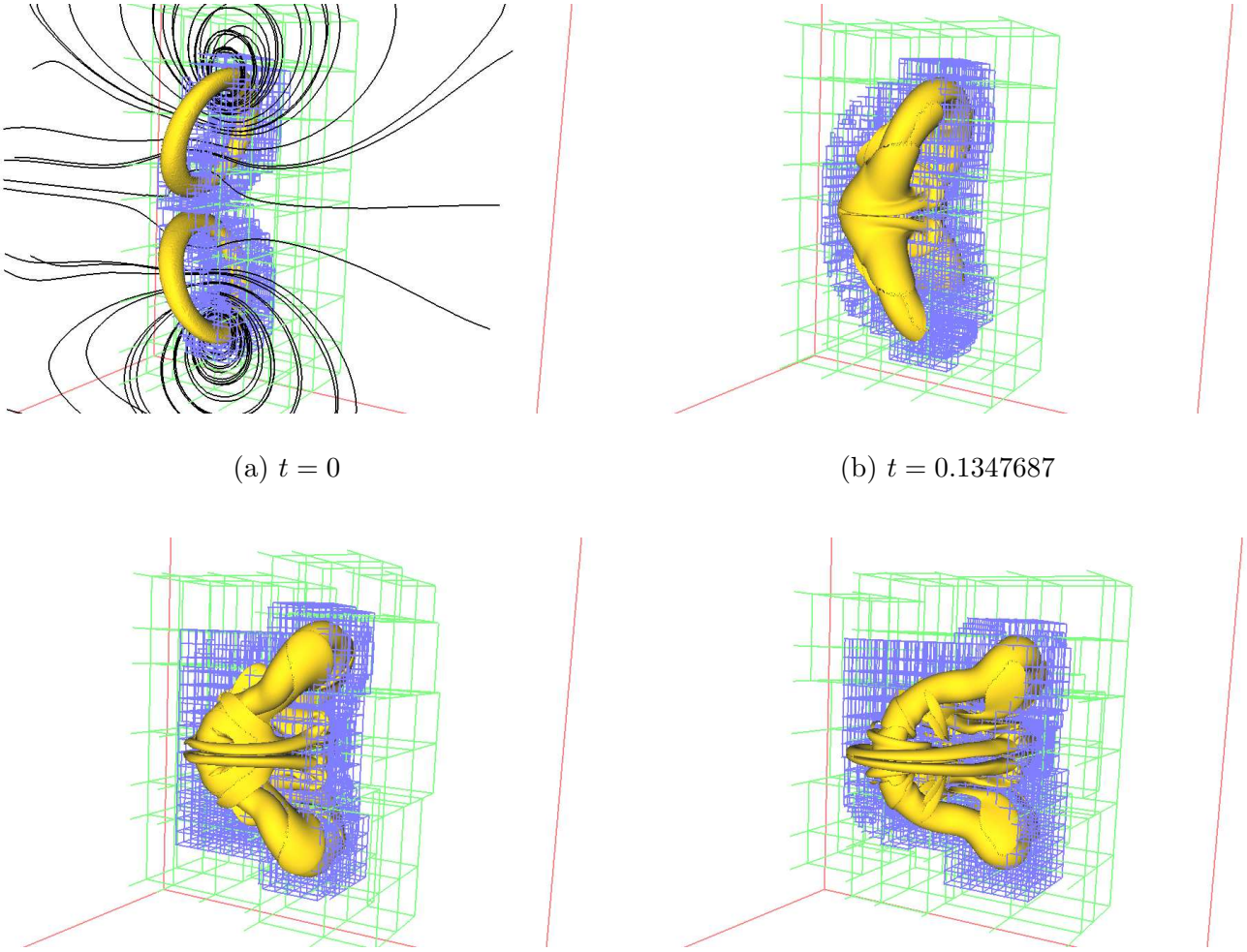
angled toward each other by an inclination angle ϕ from horizontal. The vortex rings are each initialized with a solid vorticity core, with $\omega = \omega_{interior}$ inside the vortex ring, and $\omega = 0$ outside. The parameters used for this example are:

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

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.

The problem was run in a unit cube with a 64^3 base mesh with 2 levels of refinement using $n_{ref} = 4$. 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 Figure 3.3. 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 figure 3.3; a closeup of the center as the vortex rings merge is shown in Figure 3.3, which demonstrates the how the vortex ring cores are deformed by differential

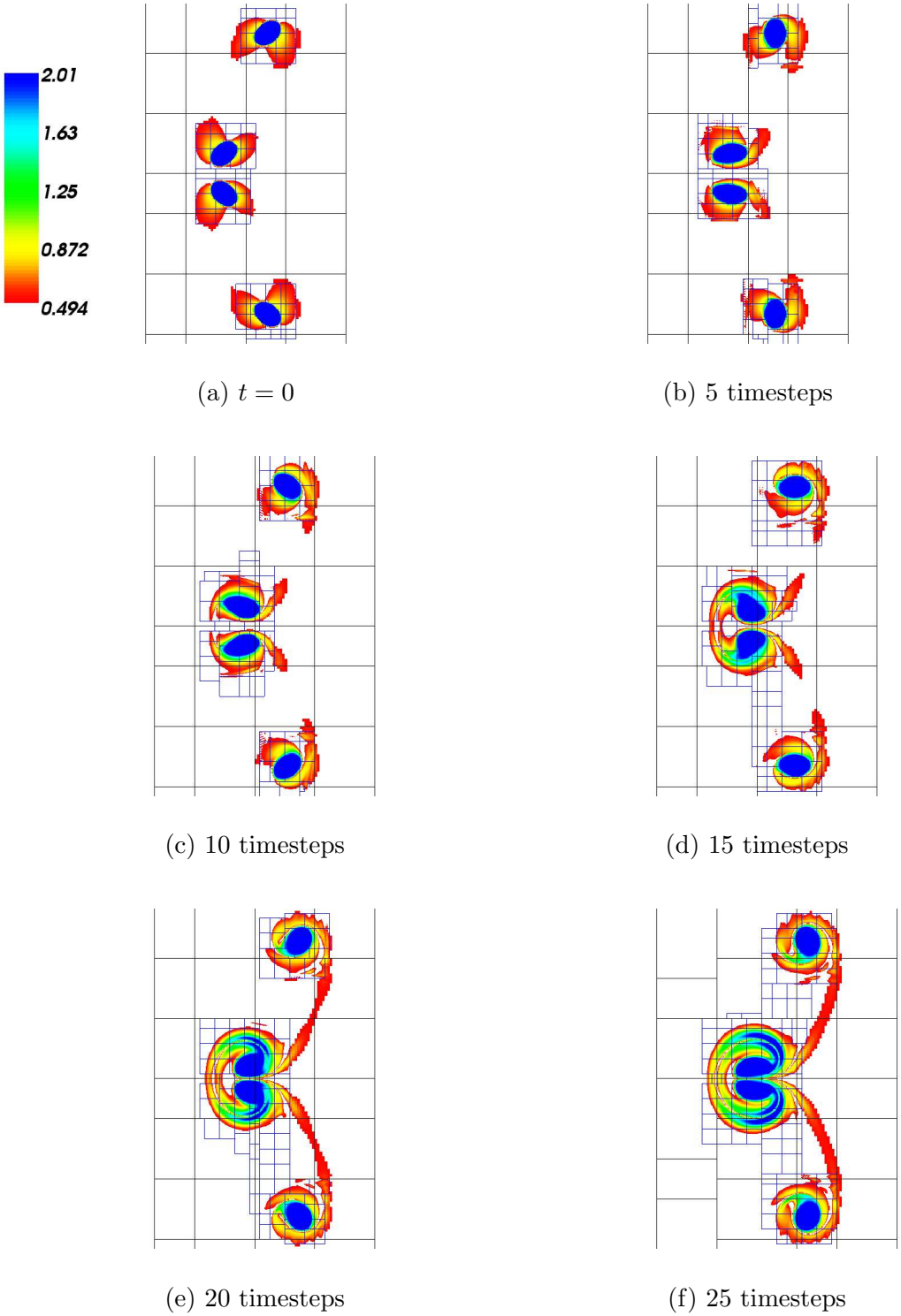


(a) $t = 0$ (b) $t = 0.1347687$

(c) $t = 0.2114612$ (d) $t = 0.2889325$

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.

shearing and vorticity diffusion. As the flow progresses, sheets of vorticity (seen in cross-section as narrow strips) are stripped from the central vortex cores and are then wrapped around and transported away. A longer-term evolution is shown in Figure 3.3, which shows the role of vortex stretching in the formation of the longer-term vortical structures in the flow.

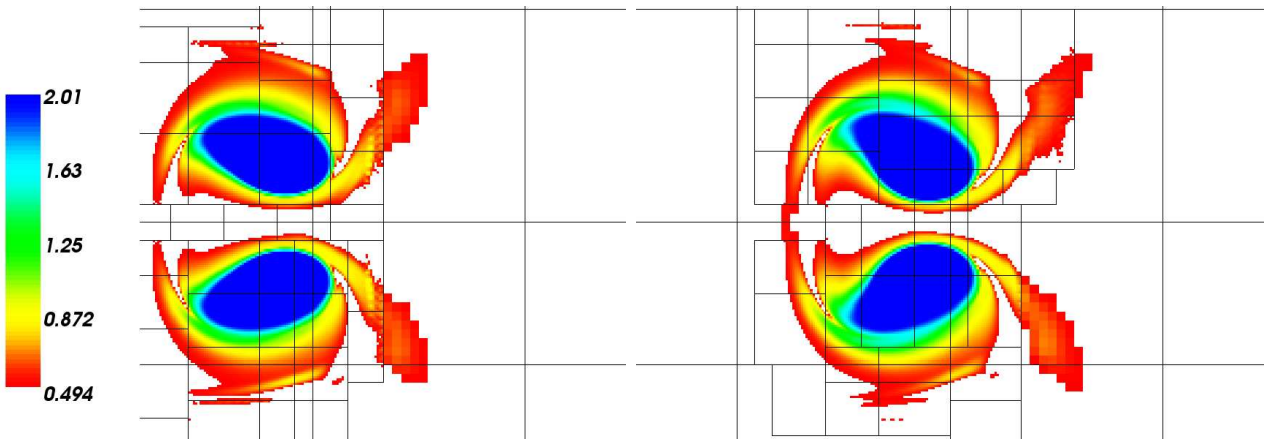


(a) $t = 0$ (b) 5 timesteps

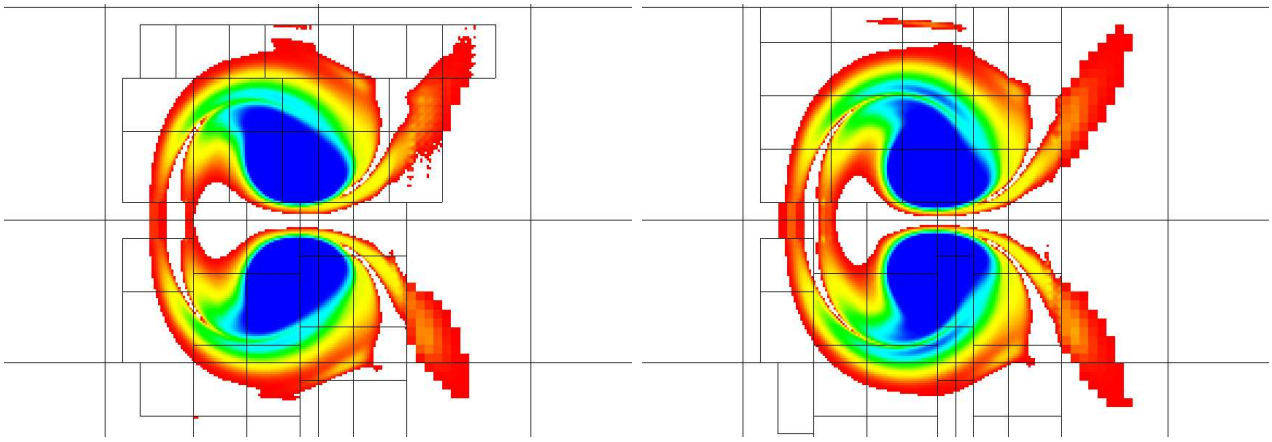
(c) 10 timesteps (d) 15 timesteps

(e) 20 timesteps (f) 25 timesteps

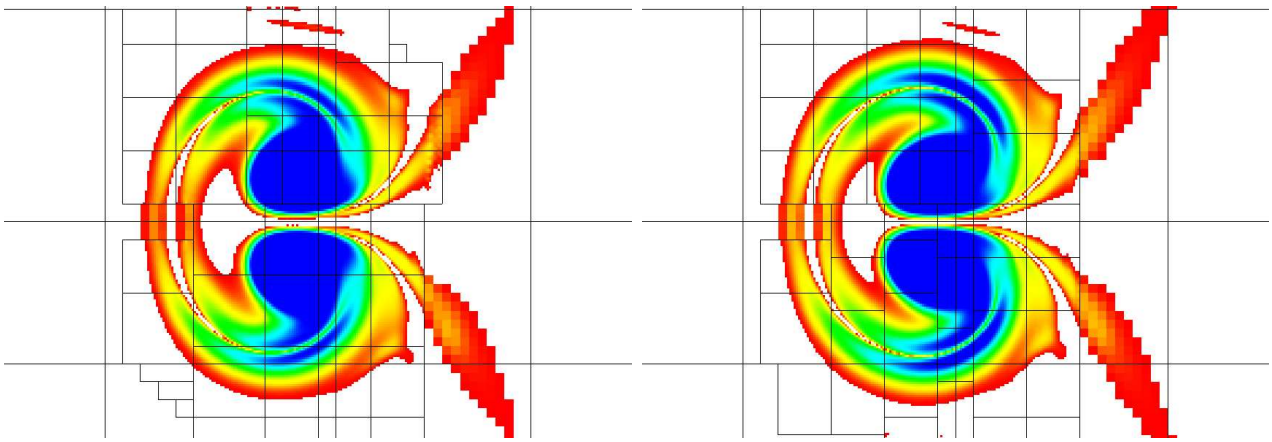
Fig. 9. Vortex merger problem – slice at $x = 0.505$ showing $log_{10}|\omega|$ at (a) initial time and after (b) 5 timesteps, (c) 10 timesteps, (d) 15 timesteps, (e) 20 timesteps, and (f) 25 timesteps. Colormap scale is from 0.5 to 2.0.



(a) $t = 0.02258$ (b) $t = 0.02740$

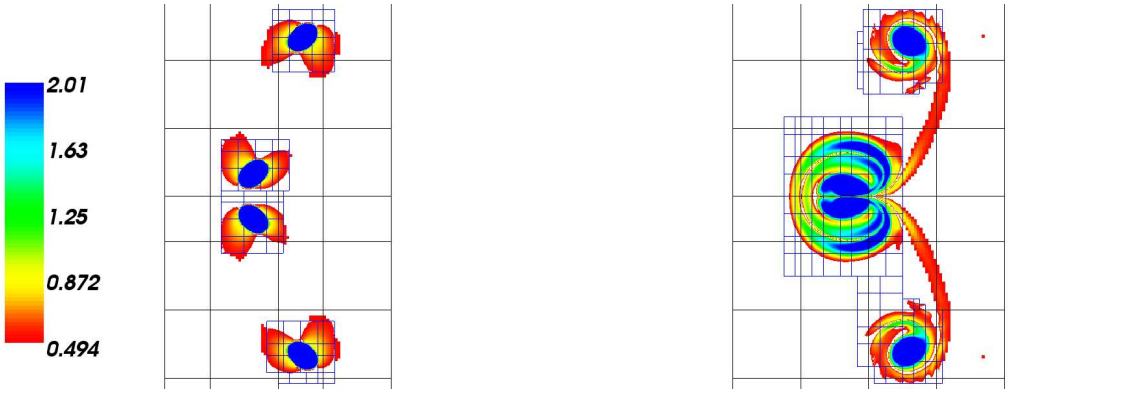


(c) $t = 0.02322$ (d) $t = 0.03706$



(e) $t = 0.04198$ (f) $t = 0.04688$

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.



(a) $t = 0$ (b) $t = 0.07217$

(c) $t = 0.13487$ (d) $t = 0.21157$

(e) $t = 0.28906$ (f) $t = 0.37270$

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.

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 forsee application of these ideas to other coupled elliptic-parabolic-hyperbolic systems, such as those found in porous media flows [23] and nonideal MHD [26].

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 $\varphi^{c,valid}$, we compute a single-level vector field $\vec{G}^f = (G_0^f, \ldots G_{\mathbf{D}-1}^f)$ that approximates the gradient to sufficient accuracy so that, its divergence is at least an $O(h)$ approximation to the Laplacian. For each $\Omega^{f,k} \in \mathcal{R}(\Omega^f)$, we construct an extension $\tilde{\varphi}$ of φ^f.

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

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

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

For this estimate of the gradient to be accurate to $O(h^2)$, it is necessary to compute an $O(h^3)$ extension of φ^f. On $\tilde{\Omega}_k^f \cap \Omega^f$, the values for $\tilde{\varphi}$ will be given

by $\tilde{\varphi}_{\boldsymbol{i}} = \varphi^f_{\boldsymbol{i}}$. The values for the remaining points in $\tilde{\Omega}^f_k - \Omega^f$ will be obtained by interpolating using φ^f and φ^c.

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

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

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

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

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

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

(ii) Interpolation in the normal direction.

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

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

$$\begin{aligned} &((\boldsymbol{i} \pm \tfrac{1}{2}(n^l_{ref} - 1)\boldsymbol{e}^d)h, \hat{\varphi}_{\boldsymbol{i}}), \\ &((\boldsymbol{i} \mp \boldsymbol{e}^d)h, \varphi^l_{\boldsymbol{i} \mp \boldsymbol{e}_d}), \\ &((\boldsymbol{i} \mp 2\boldsymbol{e}^d)h, \varphi^l_{\boldsymbol{i} \mp 2\boldsymbol{e}_d)} \end{aligned}$$

In (i), the quantities $D^{1,d'}\varphi^c$, $D^{2,d'}\varphi^c$ and $D^{d'd''}\varphi^c$ 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 $O(h^2)$, while the other two quantities need only be $O(h)$. Only values in Ω^c_{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 Ω^c_{valid}.

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

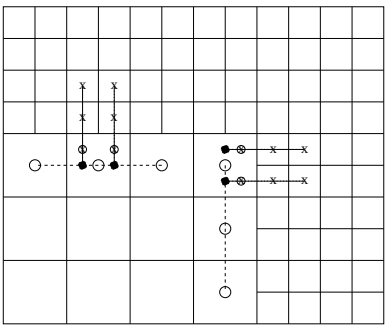


Fig. A.1. Interpolation at a coarse-fine interface. Left stencil is the usual stencil. Right stencil is the modified interpolation stencil; since the upper coarse cell is covered by a fine grid, use shifted coarse grid stencil (open circles) to get intermediate values (solid circles), then perform final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

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

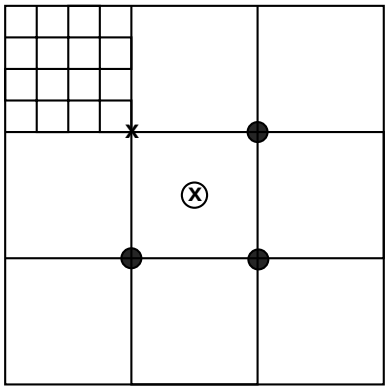


Fig. A.2. Mixed-derivative approximation illustration. The upper-left corner is covered by a finer level so the mixed derivative in the upper left (the uncircled x) has a stencil which extends into the finer level. We therefore average the mixed derivatives centered on the other corners (the filled circles)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

In the case of $D^{d'd''}\varphi^c$, 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 Ω^c_{valid} (Figure A.2)

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

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

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

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

B Unsplit Upwind Tracing

Our unsplit, second-order upwind advection scheme has its origins in Colella [12] and Saltzman [25].

We are solving a hyperbolic system of equations of the form

$$\frac{\partial U}{\partial t} + \sum_{d=0}^{\mathbf{D}-1} \frac{\partial F^d}{\partial x^d} = S \tag{B.1}$$

where $F^d = A^d U$. U is the quantity being advected, while A^d is the advection velocity in the $d-$direction.

B.1 Outline

Given $U_{\boldsymbol{i}}^n$ and $S_{\boldsymbol{i}}^n$, we want to compute a second-order accurate estimate of the face-centered fluxes: $F^{n+\frac{1}{2}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} \approx F^d(\boldsymbol{x}_0 + (\boldsymbol{i} + \frac{1}{2}\boldsymbol{e}^d)h, t^n + \frac{1}{2}\Delta t)$. In outline, the method is given as follows.

(1) Compute slopes $\Delta^d U_{\boldsymbol{i}}$, for $0 \le d < \mathbf{D}$ (the definition of $\Delta^d U_{\boldsymbol{i}}$ is given in section B.2):
(2) Compute the effect of the normal derivative terms and the source term on the extrapolation in space and time from cell centers to faces. For $0 \le d < \mathbf{D}$,

$$U_{\boldsymbol{i},\pm,d} = U_{\boldsymbol{i}}^n + \tfrac{1}{2}(\pm I - \frac{\Delta t}{h} A_{\boldsymbol{i}}^d)(\Delta^d U_{\boldsymbol{i}}) \tag{B.2}$$

$$U_{\boldsymbol{i},\pm,d} = U_{\boldsymbol{i},\pm,d} + \frac{\Delta t}{2} S_{\boldsymbol{i}}^n \tag{B.3}$$

(3) Compute estimates of F^d suitable for computing 1D flux derivatives $\frac{\partial F^d}{\partial x^d}$ using a Riemann solver, R. In this case, the Riemann solve is simply choosing the upwind direction based on A^d.

$$F^{\text{1D}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = R(U_{\boldsymbol{i},+,d}, U_{\boldsymbol{i}+\boldsymbol{e}^d,-,d}, d) \tag{B.4}$$

$$R(U_{\boldsymbol{i},+,d}, U_{\boldsymbol{i}+\boldsymbol{e}^d,-,d}, d, A^d) = \begin{cases} U_{\boldsymbol{i},+,d} & \text{if } A^d > 0 \\ U_{\boldsymbol{i}+\boldsymbol{e}^d,-,d} & \text{otherwise} \end{cases} \tag{B.5}$$

(4) In 3D compute corrections to $U_{\boldsymbol{i},\pm,d}$ corresponding to one set of transverse derivatives appropriate to obtain $(1,1,1)$ diagonal coupling. In 2D skip this step.

$$U_{\boldsymbol{i},\pm,d_1,d_2} = U_{\boldsymbol{i},\pm,d_1} - \frac{\Delta t}{3h}(F^{\text{1D}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d_2}} - F^{\text{1D}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^{d_2}}) \tag{B.6}$$

(5) In 3D compute fluxes corresponding to corrections made in the previous step. In 2D skip this step.

$$F_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d_1},d_2} = R(U_{\boldsymbol{i},+,d_1,d_2}, U_{\boldsymbol{i}+\boldsymbol{e}^{d_1},-,d_1,d_2}, d_1) \tag{B.7}$$

$$d_1 \neq d_2,\ 0 \leq d_1, d_2 < \mathbf{D}$$

(6) Compute final corrections to $U_{\boldsymbol{i},\pm,d}$ due to the final transverse derivatives.

$$\text{2D:}\quad U^{n+\frac{1}{2}}_{\boldsymbol{i},\pm,d} = U_{\boldsymbol{i},\pm,d} - \frac{\Delta t}{2h}(F^{\text{1D}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d_1}} - F^{\text{1D}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^{d_1}}) \tag{B.8}$$
$$d \neq d_1,\ 0 \leq d, d_1 < \mathbf{D}$$

$$\text{3D:}\quad U^{n+\frac{1}{2}}_{\boldsymbol{i},\pm,d} = U_{\boldsymbol{i},\pm,d} - \frac{\Delta t}{2h}(F_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d_1},d_2} - F_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^{d_1},d_2}) - \frac{\Delta t}{2h}(F_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^{d_2},d_1} - F_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^{d_2},d_1}) \tag{B.9}$$
$$d \neq d_1 \neq d_2,\ 0 \leq d, d_1, d_2 < \mathbf{D}$$

(7) Compute final estimate of fluxes.

$$F^{n+\frac{1}{2}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = R(U^{n+\frac{1}{2}}_{\boldsymbol{i},+,d}, U^{n+\frac{1}{2}}_{\boldsymbol{i}+\boldsymbol{e}^d,-,d}, d) \tag{B.10}$$

(8) Update the solution using the divergence of the fluxes.

$$U^{n+1}_{\boldsymbol{i}} = U^n_{\boldsymbol{i}} - \frac{\Delta t}{h}\sum_{d=0}^{\mathbf{D}-1}(F^{n+\frac{1}{2}}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} - F^{n+\frac{1}{2}}_{\boldsymbol{i}-\frac{1}{2}\boldsymbol{e}^d}) \tag{B.11}$$

B.2 Slope Calculation

We will use the 4th order slope calculation in Colella and Glaz [10]

$$\Delta^d U_{\boldsymbol{i}} = \frac{2}{3}((U - \frac{1}{4}\Delta_2^d U)_{\boldsymbol{i}+\boldsymbol{e}^d} - (U + \frac{1}{4}\Delta_2^d U)_{\boldsymbol{i}-\boldsymbol{e}^d})$$
$$\Delta_2^d U_{\boldsymbol{i}} = \tfrac{1}{2}(U^n_{\boldsymbol{i}+\boldsymbol{e}^d} - U^n_{\boldsymbol{i}-\boldsymbol{e}^d})$$
$$\Delta_-^d U_{\boldsymbol{i}} = U^n_{\boldsymbol{i}} - U^n_{\boldsymbol{i}-\boldsymbol{e}^d} \ ,$$

References

[1] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142(1):1–46, May 1998.

[2] A. S. Almgren, J. B. Bell, and W. Y. Crutchfield. Approximate projection methods: Part i. inviscid analysis. Technical Report LBNL-43374, LBNL, May 1999.

[3] Ann S. Almgren, Thomas Buttke, and Phillip Colella. A fast vortex method in three dimensions. *Journal of Computational Physics*, 113(2):177–200, 1994.

[4] Christopher Anderson and Claude Greengard. The vortex ring merger problem at infinite Reynolds number. *Communications on Pure and Applied Mathematics*, 42:1123–1139, 1989.

[5] Yuko Ashima and Saburo Asaka. Interaction of two vortex rings along parallel axes in air. *Journal of the Physical Society of Japan*, 42(2):708–713, 1977.

[6] M. Barad and P. Colella. A fourht-order accurate local refinement method for Poisson's equation. *JCP*, 209:1–18, 2005.

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

[8] D.L. Brown, R. Cortez, and M. L. Minion. Accurate projection methods for the incompressible navier-stokes equations. *J. Comput. Phys.*, 168:464–499, 2001.

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

[10] P. Colella and H. M. Glaz. Efficient solution algorithms for the Riemann problem for real gases. *J. Comput. Phys.*, 59:264, 1985.

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

[12] Phillip Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comput. Phys.*, 87:171–200, 1990.

[13] M. S. Day and J. B. Bell. Numerical sumulation of laminar reacting flows with complex chemistry. *Combustion Theory and Modelling*, 4:535–556, 2000.

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

[15] Boyce E. Griffith, Richard D Hornung, David M McQueen, and Charles S Peskin. An adaptive, formally second order accurate version of the immersed boundary method. *submitted to J. Comput. Phys.*, 2006.

[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):4–26, 2003.

[18] T. T. Lim. An experimental study of a vortex ring interacting with an inclined wall. *Experiments in Fluids*, 7:453–463, 1989.

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

[20] P. McCorquodale, P. Colella, and H Johansen. A Cartesian grid embedded boundary method for the heat equation on irregular domains. *JCP*, 2001.

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

[22] S Popinet. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *J. Comput. Phys.*, 190:572–600, 2003.

[23] R. Propp, P. Colella, W.Y. Crutchfield, and M.S. Day. A numerical model for trickle-bed reactors. *JCP*, 165:311–333, 2000.

[24] A.M. Roma, C.S. Peskin, and M.J.Berger. An adaptive version of the immersed boundary method. *J. Comput. Phys.*, 153:509–534, 1999.

[25] Jeff Saltzman. An unsplit 3d upwind method for hyperbolic conservation laws. *J. Comput. Phys.*, 115:153–168, 1994.

[26] R. Samtaney, P. Colella, S.C Jardin, and D.F. Martin. 3d adaptive mesh refinement simulations of pellet injection in tokamaks. *Computer Physics Communications*, 164:220–228, 2004.

[27] M. Sussman. A parallelized, adaptive algorithm for multiphase flows in general geometries. *Computers and Structures*, 83:435–444, 2005.

[28] M. C. Thompson and J. H. Ferziger. An adaptive multigrid technique for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 82(1):94–121, May 1989.

[29] E.H. Twizell, A.B. Gumel, and M.A. Arigu. Second-order, L_0-stable methods for the heat equation with time-dependent boundary conditions. *Advances in Computational Mathmatics*, 6:333–352, 1996.