Embedded Boundary Algorithms for Solving the Poisson Equation on Complex Domains

Marcus S. Day
Phillip Colella
Micheal J. Lijewski
Charles A. Rendleman

Lawrence Berkeley National Laboratory
Berkeley, CA, 94720

Daniel L. Marcus

Sage IT Partners
San Francisco, CA 94111

Subject classifications: 65N55, 35J05, 94C15

Key words: cartesian grid, embedded boundary, adaptive mesh refinement, multigrid, poisson equation

Proposed running head: Embedded Boundary Algorithms on Complex Domains

Mailing address:

Marcus S. Day
MS 50-D
Lawrence Berkeley National Laboratory
1 Cyclotron Rd.
Berkeley, CA 94720

Email: MSDay@lbl.gov
FAX: (510) 486-6900

Embedded Boundary Algorithms for Solving the Poisson Equation on Complex Domains*

Abstract

We present a graph-based strategy for representing the computational domain for embedded boundary discretizations of conservation-law PDE's. The representation allows recursive generation of coarse-grid geometry representations suitable for multigrid and adaptive mesh refinement calculations. Using this scheme, we implement a simple multigrid V-cycle relaxation algorithm to solve the linear elliptic equations arising from a block-structured adaptive discretization of Poisson's equation over an arbitrary two-dimensional domain. We demonstrate that the resulting solver is robust to a wide range of two-dimensional geometries, and performs as expected for multigrid-based schemes, exhibiting $\mathcal{O}(N \log N)$ scaling with system size, N.

1 Introduction

In the Embedded Boundary (EB) approach to discretizing PDE's in complex geometries, the physical domain is embedded completely within a larger uniform mesh. The bulk of the data underlying an EB discretization utilizes rectangular indexing, and only a small number of cells near the embedded boundary require special treatment. In this paper, we extend a class of EB discretization schemes to allow for arbitrarily complex domain boundaries in building multi-level discretization scheme components, such as multigrid and adaptive mesh refinement. We focus here on an adaptive multigrid scheme for the Poisson equation. The framework however, would extend readily as the basis for hyperbolic and incompressible flow discretizations.

EB methods have been applied to a wide range of conservation-law PDE's (alternative names for EB include "Cartesian Grid", or "Immersed Boundary"). The earliest use was in 1977 by Reyhner[1] in the context of axisymmetric transonic potential flow solutions. In 1978, Purvis and Burkhalter[2] presented a finite-volume formulation of the full potential equation via Cartesian Grid methods. In 1983, Wedan and South[3] extended this idea in two dimensions to allow multi-element and internal flow geometries. The TRANAIR code, presented in [4], employed a three-dimensional finite-element based discretization of the full potential equation, complete with local adaptivity. Clarke, et al.[5], Gaffney, et al.[6], Epstein, et al.[7] and Morinishi[8] presented work that extended the Cartesian schemes to steady Euler flows. Chiang, et al.[9] presented an Euler solver for Cartesian grids which featured adaptive gridding. Coirer[10] developed a locally adaptive upwind

*This work was supported by the Office of Energy Research, Applied Mathematical Sciences Program, Office of Computational and Technology Research, Mathematical, Information and Computational Sciences Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098

finite-volume scheme in two dimensions, and incorporated the viscous terms necessary to compute steady Navier-Stokes flow, following the work of DeZeeuw[11, 12] and Gooch[13]. Melton[14] extended the work of Coirer into three dimensions. Melton's contribution also included the capability for automated grid generation via a collection of specified "water-tight" components, and the logic for handling split irregular cells at the finest level. The latter feature, introduced in [11], reduces considerably the grid resolution required to capture the details of geometries containing sharp edges and thin bodies, such as the trailing edges of airfoils.

When applying Embedded Boundary methods to the time-dependent Euler equations, researchers must additionally deal with the overly severe CFL constraints arising from small cells cut by the boundary. The earliest schemes to deal with cut cells in a time-dependent framework were presented by Noh [15]. Some of Noh's ideas related to cell-merging and flux redistribution, are used in more contemporary works, such as the schemes presented by Quirk[16], Pember, et al.[17] and Yang, et al.[18]. Additional methods to ameliorate the CFL time step restriction have been constructed based on geometrical wave-propagation, and rotated difference schemes, and are presented in a series of papers by LeVeque[19, 20], and Berger and LeVeque[21, 22, 22].

A variety of embedded boundary schemes have been presented in the literature for elliptic problems on irregular domains. Peskin[23], LeVeque[24], Tome and McKee[25], Tau[26], and Almgren, et al.[27] present specialized elliptic solution schemes for Cartesian grids, as required to implement the elliptic solve step of the projection schemes for incompressible flows[28]. In a more general setting, LeVeque and Li[29] extend the methods in [23] to allow internal interfaces in the elliptic transport coefficients and source terms. Yang[30] then extended that scheme to incorporate complex domain boundaries embedded in a uniform rectangular grid. The resulting logically rectangular system can be inverted with fast Poisson-solver schemes, including a specially tailored multigrid-based implementation[31]. Hewitt[32] presents an embedded curved boundary scheme which is similar to that of Yang[30], except that additional care was taken to allow efficient use of ADI-based solvers. Johansen[33] presents a different type of two-dimensional scheme for elliptic equations, along the lines of the hyperbolic schemes, where the embedded boundary is treated as a physical domain boundary, and not just an internal interface. Johansen employs a novel data-centering scheme to avoid conditioning and accuracy problems exhibited by many of the previous schemes for elliptic and parabolic systems.

In this paper, we present a generalized EB domain specification for the data associated with the numerical integration of conservation-law PDEs. In our scheme, the computational domain is represented as a connected graph; the nodes of the graph represent the discrete cells on the grid, and the edges represent the cell faces. The graph representation of gridded data leads to an intuitive cell merging strategy for generating successively coarser geometry descriptions from the finer grid. This framework is applied toward the construction of an EB Poisson solver, employing block-structured adaptive mesh refinement[34] and multigrid(cf. [35]). Here, we extend the elliptic EB discretization of Johansen and Colella[33] to fully arbitrary geometrical configurations, thereby allowing complete geometry coarsening that is not limited by complex boundary shape. In a more general sense, this application also extends [11, 14] to allow "split-cells" at all levels of refinement, rather than at only the finest level. For this Poisson-solver example, we present convergence results which verify

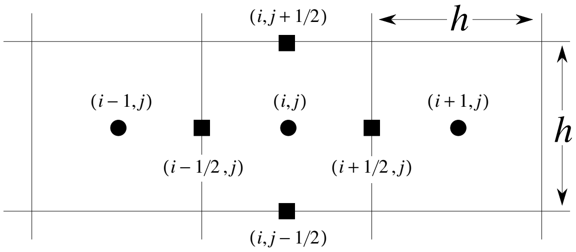


Figure 1: Indexing scheme for Uniform gridding with mesh spacing, h, in two dimensions.

that, even in complex domains, our scheme converges at the expected rates, in terms of both grid-refinement, and multigrid relaxation performance.

The notation of our embedded boundary framework is motivated in Section 2 via the finite-volume discretization of the Poisson equation. We present our simple multigrid scheme in Section 3, including details of the EB grid-coarsening strategy. In Section 4, we present the block-structured adaptive mesh refinement scheme and associated extensions to the single-level multigrid iteration. In Section 5, a simple geometry generator is described based on the requirements of the grid-coarsening strategy, and the Poisson discretization. We demonstrate the generality and convergence of these schemes in Section 6 through a variety of example test cases. We add some concluding remarks in Section 7.

2 Embedded Boundary Poisson Discretization

As a prototypical example of a conservation law PDE, take the Poisson equation for the potential, $\varphi(\vec{x})$, d dimensions:

$$\nabla^2 \varphi(\vec{x}) = \rho(\vec{x}) \tag{1}$$

and for clarity of exposition, consider Neumann conditions to apply on all boundaries, $\partial\Omega$, of the computational domain, Ω (we will remove this restriction in subsequent sections). We solve Equation 1 on a discrete grid of uniform cells. The cells are indexed by the vector $\mathbf{i} = (i_1, \ldots, i_d)$, and are located at $\Delta_{\mathbf{i}} = [(i_1 - 1/2)\,h, (i_1 + 1/2)\,h] \times \ldots \times [(i_d - 1/2)\,h, (i_d + 1/2)\,h]$ (see Figure 1). Each cell has $2 \cdot d$ faces, $\mathbf{s_i} = \mathbf{i} \pm u_k/2$, $k \in [1, d]$, where u_k is the unit vector in the k^{th} direction, and the discretized dependent variable is defined on cell centers, $\varphi_{\mathbf{i}} \approx \varphi(\mathbf{i}h)$.

The divergence of the conserved flux, $\vec{F} = \nabla\varphi$, over the control volume, $\Delta_{\mathbf{i}}$, can be written via the divergence theorem as

$$\begin{aligned}\left(\nabla \cdot \vec{F}\right)_{\mathbf{i}} &= \int_{\Delta_{\mathbf{i}}} \nabla \cdot \vec{F} dV \Big/ \int_{\Delta_{\mathbf{i}}} dV = \oint_{\partial\Delta_{\mathbf{i}}} \vec{F} \cdot d\vec{S} \Big/ \int_{\Delta_{\mathbf{i}}} dV \\ &= \frac{1}{h^d} \oint_{\partial\Delta_{\mathbf{i}}} \vec{F} \cdot \hat{n}\, h^{d-1}, \text{ for uniform grid spacing, } h\end{aligned} \tag{2}$$

where $\hat{n}$ is the normal on the surface, $\partial\Delta_{\mathbf{i}}$, of the control volume. Using the midpoint rule to evaluate the surface integrals,

$$\left(\nabla \cdot \vec{F}\right)_{\mathbf{i}} \approx \frac{1}{h^d} \sum_{s \in \mathbf{s_i}} \left(\vec{F} \cdot \hat{n}\right)_s h^{d-1} + \mathcal{O}\left(h^2\right) \tag{3}$$

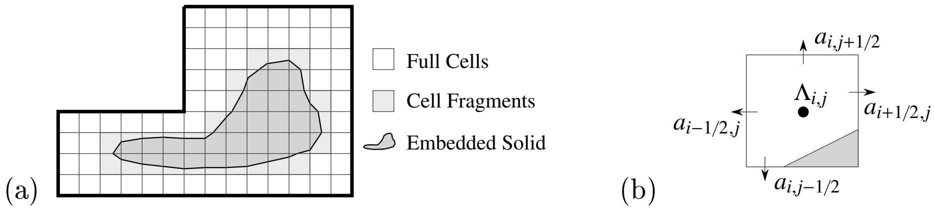


Figure 2: (a) A two-dimensional embedded boundary domain, which excludes the embedded solid. (b) A standard cell for simple Embedded Boundary discretization methods, with volume fraction, Λ, and apertures, $\mathcal{A}_s$. For the Neumann case, there is no flux into the embedded boundary.

Here, $\left(\vec{F} \cdot \hat{n}\right)_s$ is the normal component of the flux at the center of face s.

For the Poisson equation, we may compute $\vec{F}$ using centered differences. For $d = 2$, $\mathbf{i} = (i, j)$, and $s = \{(i \pm 1/2, j), (i, j \pm 1/2)\}$, the terms in the sum expand explicitly to

$$\begin{aligned} \left(\vec{F} \cdot \hat{n}\right)_{i+1/2,j} &= \frac{\varphi_{i+1,j} - \varphi_{i,j}}{h}, & \left(\vec{F} \cdot \hat{n}\right)_{i-1/2,j} &= \frac{\varphi_{i-1,j} - \varphi_{i,j}}{h} \\ \left(\vec{F} \cdot \hat{n}\right)_{i,j+1/2} &= \frac{\varphi_{i,j+1} - \varphi_{i,j}}{h}, & \left(\vec{F} \cdot \hat{n}\right)_{i,j-1/2} &= \frac{\varphi_{i,j-1} - \varphi_{i,j}}{h} \end{aligned} \tag{4}$$

If the face s coincides with the edge of the computational domain, we simply set $\vec{F}_s = 0$ to enforce the Neumann boundary conditions. Inserting these expressions into Equation 3, and setting the result equal to the cell-centered discrete values of $\rho_{\mathbf{i}}$, we obtain an elliptic linear system of equations, which can be solved using a variety methods, including multigrid relaxation.

2.1 Embedded Boundaries

The discrete cells for the Embedded Boundary method are based on a uniform underlying grid of *mesh cells*, just as in the regular case above. And like the regular case, physical boundaries may be represented by the grid-aligned edges of the uniform grid. However, within the regular mesh, we also allow "solid body" boundaries which may not align with the coordinate directions (see Figure 2(a)). These bodies are represented as piecewise linear surfaces (curves in 2D) cutting through the background rectangular mesh cells, leaving *cell fragments* in the domain. Cell fragments will be distinguished from *full cells*, which are neither cut nor covered by the embedded solid. The region inside the embedded boundary is not part of the computational domain.

If there is only one cell fragment at each mesh cell, full and partial cells can be identified uniquely by the multi-dimensional index, $\mathbf{i} \in \mathcal{Z}^d$. Indices of cells completely covered by the embedded body are considered invalid. Define the volume fraction, $\Lambda_{\mathbf{i}}$, as the ratio of cut to full cell volume for cell $\mathbf{i}$, and the face aperture, $\mathcal{A}_{\mathbf{i},s}$, as the ratio of cut to full face area on side s of cell $\mathbf{i}$ (see Figure 2(b) for the "reference" cut cell in two dimensions). Equation 3 can now be extended to apply to the regular and irregular cells in the Embedded Boundary description:

$$\left(\nabla \cdot \vec{F}\right)_{\mathbf{i}} \approx \frac{1}{\Lambda_{\mathbf{i}} h^d} \sum_{s \in \mathbf{s}_{\mathbf{i}}} \left(\vec{F} \cdot \hat{n}\right)_s \mathcal{A}_{\mathbf{i},s} h^{d-1} + \mathcal{O}\left(h^2\right) \tag{5}$$

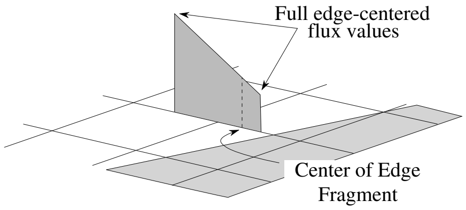


Figure 3: Second-order flux evaluation due to Johansen and Colella. The shaded region of the grid represents "solid body", outside the computational domain. The flux at the center of the partial face is linearly interpolated from the fluxes computed at the full edge centers.

This error estimate is valid only at the center of mass of the cut cell.

In order to compute the flux terms in Equation 5 from the state in the cut cells, we adopt the data-centering scheme detailed in [33]. In this scheme, all state data resides at the geometric center of the full cell containing the partial cell. Note that this position will actually lie outside the computational domain if the partial cell occupies less than half the full cell's volume, since the embedded boundary is a piecewise-linear interface. It follows then that the scheme may be applied only for problems where the solution profile may be smoothly extended into the embedded boundary region a distance $\mathcal{O}(h)$.

As detailed in [33], numerical fluxes for this scheme are computed at the center of the full edge underlying each of the partial edges using simple central differences–i.e. the flux resides midway between full cell centers, where the state resides[1]. [2] used a full-edge-centered flux in their conservative integral sum corresponding to Equation 5. However, for second-order accuracy, the surface integrals should be evaluated via the midpoint-rule, requiring flux values which are interpolated to the center of the *partial* edge. This is easily computed to second-order accuracy by linearly interpolating tangentially adjacent full-edge-centered quantities (see Figure 3). In [33], this scheme was shown to be a formally consistent approximation, with errors in the computed field quantity diminishing asymptotically to second order in all relevant norms. The truncation error, weighted by volume fraction, Λ, is first-order in h on the boundary cells, uniformly in Λ, so that the entire scheme has a second-order truncation.

2.1.1 Extension to Complex Geometries

The above procedures for discretizing conservation laws in the embedded boundary framework (based on cell-centered states, and tangentially interpolated fluxes) is limited to applications where the irregular solid bodies are "thick". In particular, the discretizations for the "thin-body" and "trailing-edge" scenarios such as those shown in Figure 4 are ill-defined, since we can no longer uniquely identify partial cells using the index, $\mathbf{i}$. Such situations arise when constructing multiple-level numerics, such as multigrid linear solvers and adaptive mesh refinement. In the literature, these cases also arise if the immersed body has very thin fingers or trailing edges (such as airfoils).

[1]This position may actually lie outside the computational domain as well, depending on the associated partial cell volumes. Also, when we later extend the scheme for thin bodies, edge-centered flux values for edges on opposite sides of the body will coexist at the same physical location.

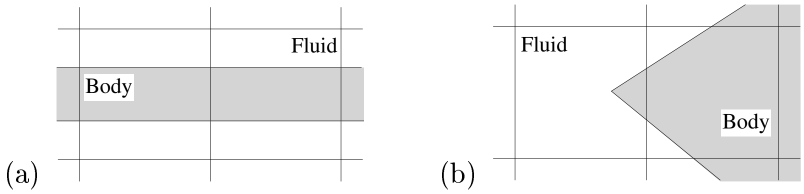


Figure 4: (a) An example "thin" body in the Embedded Boundary grid framework. Each of the two *mesh* cells shown contains multiple *partial* cells. (b) A "trailing edge" geometry, where a cell has more than one neighbor in a coordinate direction.

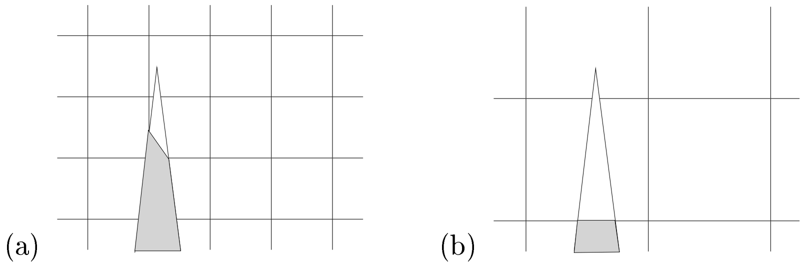


Figure 5: (a) The blunting procedure used in existing embedded boundary implementations unable to otherwise cope with the thin-body or trailing edge problems. (b) The same geometry represented on a coarser grid. The location of the "tip" will continue to creep with coarsening; the problem coarsened many times will no longer represent gross physical properties of the original geometry, and may lead to unphysical communication in the computed fields

Adaptive EB methods to date have employed a simple geometrical "blunting" technique (a schematic of this process is shown in Figure 5(a)). Blunting cuts off arbitrary portions of the embedded body that lead to multiple cell fragments at a single index. Geometric fidelity is preserved typically through concurrent use of adaptive mesh refinement (see [16], for example). However, blunting and mesh refinement alone have not been sufficient for large three-dimensional simulations. DeZeeuw[11] and Melton[14] have implemented "split-cell" schemes, allowing multiple discrete cells to exist at a given mesh cell location. By localizing the region of greatest refinement, they reduce the overall computational requirements, particularly in simulating complex three-dimensional machinery.

In initial implementations of the "split-cell" approach, the fixed-width tree-based data structures allow cell splitting only at the level most refined locally. However, for genuinely multiple-level algorithms (i.e. those requiring a reasonable representation of the state at all refinement levels), we must generalize the scheme. By definition, the extension requires an unstructured data format, but only near the embedded boundary. For data over the bulk of the domain, efficient structured array storage is sufficient and desirable.

2.2 A Formal Description of Embedded Boundaries

In the following, we present a dual-structure scheme that is general enough for arbitrary geometrical complexity, yet does not preclude an efficient implementation. We describe and manipulate the computational domain via a *connected graph*, $G^{tot} = \{V^{tot}, E^{tot}\}$, emphasizing the role of connectivity and communication through the domain. In G^{tot}, the nodes,

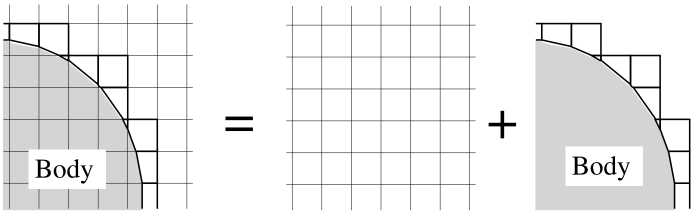


Figure 6: The dual-structure scheme stores the regular cells in a logically rectangular array, and the irregular cells as a generalized graph (detailed in text). The unstructured gridding used for the cut cells allows arbitrarily complex embedded structures in the domain, while the logically rectangular data structures for the remainder of the domain enable access to the inherent efficiencies of regular structured gridding.

V^{tot}, represent the set of finite-volume cells, and the edges, E^{tot}, represent the faces through which the cells communicate.

We divide G^{tot} into two sub-graphs, G and G^{full}, with $G = \{V, E\}$, containing all cells, V, adjacent to the embedded boundary. The set, E, contains all the edges between the nodes in V. We define $G^{full} = \left\{V^{full}, E^{full}\right\}$ similarly for the regions away from the embedded boundary. Data on V^{full} and E^{full} are stored and manipulated in logically rectangular arrays–incurring a small overhead; viz. unused locations occupied by partial and empty cells. Irregular data on V and E is maintained in a sparse representation which implements the nontrivial aspects of the connectivity implied by G^{tot} (see Figure 6). The interface between the two subgraphs is a small subset of edges, $E^{tot} - E^{full} - E$, and is maintained as an auxiliary set in the G^{full} data structures, since there is a natural location in the arrays.

Logically, an edge is specified by the two nodes that surround it. Let us define the subscript operator, "-" over edge, e, such that e_- returns the *node* object to the "low" side. Similarly, the "+" operator is defined such that e_+ returns the node object to the "high" side of e. Now, we may specify formally the defining properties of all edges in E:

$$\forall\, e \in E,\; e = (e_-, e_+)\,,\quad e_- \in V \text{ and } e_+ \in V$$

That is, both the high and low nodes are in the graph. Relating the graph back to the computational domain, we may associate a set of geometrical attributes with each node or edge, such as its index, $\mathbf{i} \in \mathcal{Z}^d$ (note that multiple nodes may have identical indices). If $\mathcal{K}(v)$ is the operator returning the index of node v, then:

$$\mathcal{K}(e_+) - \mathcal{K}(e_-) = u_k$$

for e belonging to the set of faces in the k^{th} coordinate (here, u_k is the k^{th} unit vector). That is, the nodes on either side of an edge are separated by a unit vector. The indices of the edge is:

$$\mathcal{K}(e) = \frac{\mathcal{K}(e_+) + \mathcal{K}(e_-)}{2}$$

Each cell in the domain has an associated *cell volume fraction*, $\Lambda(v) : V \mapsto [0, 1]$. The cell volume fraction is the ratio of the partial cell volume to that of the underlying mesh

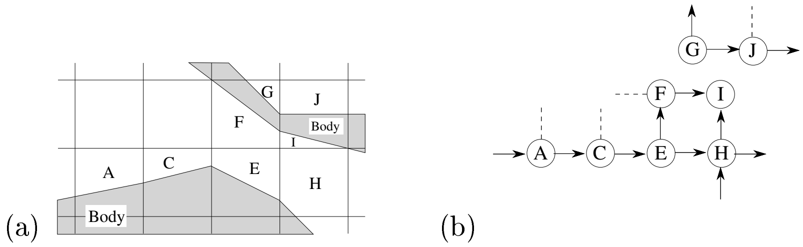


Figure 7: (a) A small region of a typical multiply-connected geometry. Cells A, C, F and J are bounded on a side by "interface" faces. (b) The graph representation of the partial cells labeled in (a). The dotted lines indicate interface faces, which are not strictly part of the graph. Information from the logically rectangular data communicates with the graph via the interface faces.

cell. Each cell face in the domain has an associated *face area aperture*, $\mathcal{A}(e) : E \mapsto [0,1]$. The face area aperture is the ratio of the partial face area to that of the underlying mesh cell face. For our uniform grid spacing, h, the mesh cell volume is h^d, and the mesh face area is h^{d-1}. For the nodes in G^{full}, $v^{full} \in V^{full}$ $\Lambda\left(v^{full}\right) = 1$, and for $e^{full} \in E^{full}$, $\mathcal{A}\left(e^{full}\right) = 1$.

The two data structures, regular and irregular, will communicate through the *interface* faces (see Figure 7). Interface faces have on one side, a full cell in G^{full}, represented in the block-structured dense data, and on the other side, a cell fragment represented in the graph, G. Formally, we define an interface edge:

$$e \,\text{an inteface if}\, e_{\pm} \in V \,\text{and}\, e_{\mp} \in V^{full}.$$

Also, we define the corresponding unit vector set, $\mathcal{L}(v) \subset \mathcal{O} = \{\pm u_1, \ldots, \pm u_d\}$, as the subset of orientations about the node, $v \in V$, which are bounded by interface faces. Using the notation of Figure 1, the set of indices of the interface faces about v are then $\mathcal{K}(v) + o/2, \forall o \in \mathcal{L}(v)$. In Figure 7, $\mathcal{L}(\mathrm{F}) = \{-u_1\}$, and there is an interface edge connected to cell F residing at index $\mathcal{K}(\mathrm{F}) - u_1/2$. Also, $\mathcal{K}(\mathrm{I}) = \mathcal{K}(\mathrm{J})$, and $\mathcal{K}(\mathrm{F}) = \mathcal{K}(\mathrm{G})$. This is an example of a "thin-body" geometry.

We are now in a position to define the appropriate generalization of Equation 5. For the partial cells, $v \in V$, the sum consists of contributions from the partial faces, $e \in E$ as well as from the interface faces. There is no flux contribution from the boundary, since we have assumed homogeneous Neumann boundary conditions. Let us introduce a flux function, $F = \left(F^R, F^I\right)$, which lives on cell faces in the regular and irregular parts of the domain. (i.e. $\forall o \in \mathcal{O}, \mathbf{i} \in \mathcal{Z}^d$, $F^R\left(\mathbf{i} + \frac{o}{2}\right) : \mathcal{Z}^d + \frac{\mathcal{O}}{2} \mapsto \Re$ and $\forall e \in E$, $F^I(e) : E \mapsto \Re$). We have,

$$\begin{aligned}\left(\nabla \cdot \vec{F}\right)_{v \in V} &= \frac{1}{\Lambda(v)\, h^d} \left\{ \sum_{e:e_-=v} F^I(e)\, \mathcal{A}(e)\, h^{d-1} - \sum_{e:e_+=v} F^I(e)\, \mathcal{A}(e)\, h^{d-1} \right. \\ &\left. + \sum_{o \in \mathcal{L}(v)} F^R\left(\mathcal{K}(v) + \frac{o}{2}\right) h^{d-1} \operatorname{sgn}(o) \right\} + \mathcal{O}\left(h^2\right) \qquad (6)\end{aligned}$$

where $\operatorname{sgn}(x) = 1$ if $x > 0$, otherwise $\operatorname{sgn}(x) = -1$. Note that the right side contains an implied sum over coordinate directions, and that this formulation treats correctly the cases where there are more, or less than a single face on a given side of a discrete cell. On the regular cells, $v \in V^{full}$, we simply apply Equation 3.

The flux functions, F^R and F^I, may be defined, according to the PDE, using a pair of cell-based data structures for the state, $\varphi = \left(\varphi^R, \varphi^I\right)$, where $\varphi^R : \mathcal{Z}^d \mapsto \Re$, and $\varphi^I : V \mapsto \Re$. It is useful to define F^R in two passes. For the Poisson equation, on the first pass, F^R may be defined using central differences on data exclusively from φ^R (as in Equation 5 for $d = 2$). On the second pass, the fluxes on the interface edges are overwritten with the central differences using data from φ^R on the full-cell side of the interface, and data from φ^I on the partial-cell side. Formally, the expression is

$$F^R\left(\mathcal{K}(v) + \frac{o}{2}\right) = \frac{1}{h}\left(\varphi^R(\mathcal{K}(v) + o) - \varphi^I(v)\right)\operatorname{sgn}(o) \tag{7}$$

The flux function, F^I, will be computed using the algorithm described in [33]. To carry this out in our generalized context, we require a set of *monotone* nodes, $\mathcal{M}(v, L_m)$, associated with node v, and built from $G^{tot} = (V^{tot}, E^{tot})$ (i.e. the nodes $\{v, \mathcal{M}(v, L_m)\} \subset V^{tot}$). The node, $u \in \mathcal{M}(v, L_m)$ is *reachable* from v via a monotone path of length L_m if there exists some $\vec{N} \in \mathcal{Z}^d$ such that

$$\mathcal{K}(u) = \mathcal{K}(v) + \sum_{k=1}^{d} N_k\, u_k, \text{ where } \sum_{k=1}^{d} |N_k| = L_m$$

That is, $\mathcal{M}(v, L_m)$ consists of a sequence of at most $L_m > 0$ movements to neighbor nodes, with the restriction that the movements along any single coordinate be all of the same sign. Note that no two cells in a path can be at the same index. Also note that the list of cells in a monotone path may include full cells from V^{full} as well as partial cells from V.

Geometrically, a monotone path may be used to restrict the neighborhood of a cell for the purposes of constructing interpolating profiles that do not span an embedded thin body (see Figure 8(a)). Monotone node sets may also be used to define an appropriate neighborhood for conservative flux redistribution (such as in the scheme described in [17]). In the present context, we will utilize the concept of monotonicity to help identify candidate neighbor faces to involve in the flux interpolation scheme of [33].

In order to carry out the flux interpolation for a cell face, e, we need to identify all the appropriate "other" edges, e', containing a cell-centered flux value we can use in the interpolant. To do this, search all edges in E^{tot} for e' where

$$e' = (v \in \mathcal{M}_-, u \in \mathcal{M}_+) \tag{8}$$

Here $\mathcal{M}_- = \mathcal{M}(e_-, 1)$ and $\mathcal{M}_+ = \mathcal{M}(e_+, 1)$ (see Figure 8(b)). In general, this search will return either zero, one or several candidate full or partial edges. Since we do not maintain cell *location* information, we cannot select the "best" from multiple candidates. We might further restrict the search to find only *full* cells for which Equation 8 held true, or to return only faces such that $\mathcal{A}(e') = 1$. Alternatively, we just average together the influence of all edges that qualify. If the condition in Equation 8 returns an empty list, we have only a single point on the edges to use for the interpolant, so we may construct only a piecewise constant flux interpolant.

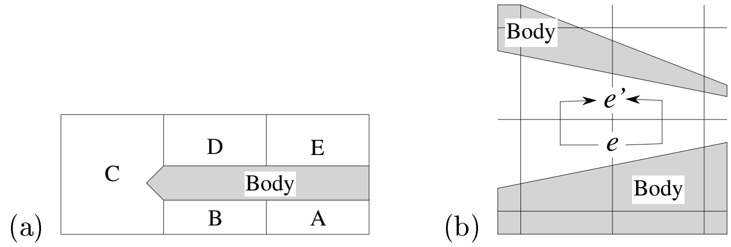


Figure 8: (a) A *monotone path* is a sequence of steps along a connected path in the cell fragment graph such that all steps in any single coordinate direction are of the same sign. In the two-dimensional example, there is no monotone path from B to D, or from A to E. However, the paths {A-B, A-B-C, C-D, C-D-E, D-E} are monotone. (b) The face, e', is found by constructing two 1-step monotone paths, as described in the text. Information above the top body is not used when constructing the interpolated flux at e.

Now, we compute the full edge-centered Poisson flux, F^f for a state, φ, on the node set, V

$$F^f(e) = \frac{\varphi^I(e_+) - \varphi^I(e_-)}{h} + \mathcal{O}\left(h^2\right) \tag{9}$$

and then compute the partial edge-centered flux, F, as the linear in 2D, bilinear in 3D, interpolation between $F^f(e)$ and the set $F^f(e')$. For example, if $d = 2$, and the search, Equation 8, returns a single candidate, then

$$F^I(e) = F^f(e') + \frac{1}{2}\left(\mathcal{A}(e') + \mathcal{A}(e)\right)\left(F^f(e) - F^f(e')\right) + \mathcal{O}\left(h^2\right) \tag{10}$$

As an aside, notice that since data is stored for the full cells, $v \in V^{full}$, in dense block-structured arrays, there are additional array positions corresponding to the mesh cells partially or completely covered by the solid (i.e. $\mathbf{i}$, where $\nexists v \in V^{tot}$ such that $\mathcal{K}(v) = \mathbf{i}$). The interface faces have the effect of preventing any direct communication across the regular edges into these covered cells. The covered cells thus become isolated from the computation, and are effectively wasted space allocated for the solution process. In general though, the computer resources spent on unused rectangular cells covered by the embedded structures is easily minimized to be a negligible overhead cost for the calculation.

2.3 Dirichlet Boundaries

In this section, we generalize the Poisson problem of interest to include Dirichlet boundaries. Since we are using a cell-centered approach, Dirichlet conditions imposed along the domain boundary, $\partial\Omega$, result in nontrivial fluxes through the boundary faces. We present the methods we use to evaluate the fluxes based on the gradient of a multi-dimensional polynomial interpolant constructed using the boundary data, and the internal state.

The case where the Dirichlet boundary aligns with the grid index coordinates is depicted for $d = 2$ in Figure 9(a). The flux is to be evaluated at the midpoint of the cell face on the physical domain boundary ($\blacksquare$), using a parabola constructed with the boundary value (at $\blacksquare$), and internal state values (at $\circ$). The procedure extends directly to $d = 3$, since the interpolant is constructed only in the dimension normal to the boundary surface.

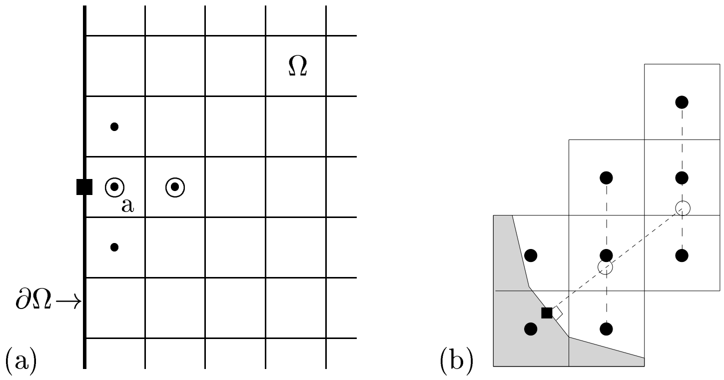


Figure 9: (a) Dirichlet boundary fluxes for cell "a" (for $d = 2$), computed by fitting a parabola through the value at the domain boundary, and two interior points (○). The resulting Laplacian stencil involves $2 \cdot d$ points (•). (b) Two dimensional quadratic interpolation for computing profile gradients normal to the embedded boundary.

The embedded boundary case is depicted in Figure 9(b). The embedded boundary is represented as a piecewise linear surface reconstruction between adjacent nodes on the irregular cell graph, as detailed in Appendix A. The Dirichlet boundary value and the resulting normal boundary gradient both live at the center of the cell's reconstructed surface. A quadratic interpolant is constructed between this location, and where the boundary normal intersects two adjacent grid lines (or planes, if $d = 3$) nearby; the intersection locations are marked in Figure 9(b) with ○'s. The procedure for carrying this out follows closely the one outlined in [33].

State values at the grid-line intersection locations are evaluated with a quadratic interpolant (parabolic for $d = 2$, bi-quadratic for $d = 3$). The interior state values used for constructing the multi-dimensional interpolating surface must be in a *monotone* path from the partial cell, as discussed in Section 2.2. This requirement prevents cases of unphysical communication, where parts of the embedded boundary lie in between the cells used for constructing the interpolant. The quadratic interpolant for embedded boundary fluxes that is constructed in this fashion remains well-conditioned for arbitrarily small partial cells adjacent to the boundary, as detailed in [33].

For sufficiently coarse geometries, a quadratic interpolant may be impossible to construct, simply for lack of sufficient candidate cells in a monotone path from the boundary location. Typically, this occurs when a complex geometry is underresolved, or when an embedded body is within $2h$ from the regular boundary. In these cases, we construct a bilinear interpolant in two dimensions (tri-linear in 3D), if possible, from adjacent cells. If there are no adjacent cells available, we set the flux at the Dirichlet boundary to zero, effectively using a piecewise-constant interpolant. In practice, when we are forced to reduce the order of the boundary interpolant on any cell at the finest level, our codes generate warning messages, since the resulting discretization becomes formally inconsistent. The remedy is usually to redefine the underlying rectangular grid so as to ensure sufficient grid points. Unless otherwise mentioned, none of the results presented in later sections required boundary interpolant order reduction.

3 Multigrid

Using Equation 6, and the dimension-dependent expression for the flux, such as Equation 10, we build a discretization for Equation 1 of the form

$$L(\varphi) = \rho \tag{11}$$

Equation 11 can be solved with using point relaxation with multigrid acceleration[35]. Typically, we employ simple "V-cycle" multigrid schedules in the relaxation, using piecewise constant prolongation, volume-weighted restriction, and a simple smoother of the Gauss-Seidel type. It is worth noting that our level-transfer operators fail the well-known requirement that

$$n_P + n_R > 2n \tag{12}$$

where n is the order of the differential operator, and n_P (n_R) is the maximum degree of exactly interpolated (coarsened) polynomials plus 1. For our choices, $n_P = n_R = 1$. In fact, inequality (12) is a heuristic for "optimal" multigrid performance, and is not strictly necessary; we demonstrate that the computational work in our algorithm scales nearly linearly with system size despite our low-order transfer functions.

Details of the multigrid V-cycle are presented in Section 3.1. The scheme has been tailored to solve Equation 11 in *correction form*, applicable to our *linear* problem (i.e. solve for $e : L(\varphi^0 + e) = \rho$, where φ^0 is some initial guess for φ). The boundary conditions for the correction, e, are simply the homogeneous form of those of the original problem for φ. Our multigrid scheme requires a hierarchy of grids, created by coarsening recursively the original geometry via a procedure we detail in Section 3.2. We detail the smoother and level-transfer operations in Section 3.3.

3.1 Multigrid V-cycle

We label the refinement levels of our problem domain with $m : 0 \leq m \leq m_{hi}$, where m_{hi} represents the original level, where we desire the problem solution. The multigrid iteration is initiated by invoking the multigrid level-relaxer (the "V-cycle") on m_{hi}. The level-relaxer applies some number of smoothing passes, and then constructs the next coarser problem using the smoothed residual. The coarse problem is relaxed with a recursive call to the level-relaxer. At the bottom of the cycle, the coarse equations are solved "exactly", and the resulting correction is interpolated back up to the next finer level. The interpolated corrections from the coarse grid are added to the next finer solution, which is then smoothed once again. A complete V-cycle terminates when the finest solution has been incremented with coarse corrections and smoothed. The V-cycle is invoked repeatedly until the magnitude of the residual, $R^m = \rho^m - L^m(\varphi^m)$, is acceptably small at $m = m_{hi}$.

Let $P(\Omega^{m+1})$ be the projection of the grid at multigrid level, $m+1$, onto the grid at level m. The recursive multigrid level relaxation is shown in Algorithm 1, for the case that $P(\Omega^{m+1}) = \Omega^m$ on all multigrid levels, $m < m_{hi}$ (all grids cover the entire domain). The level m_2 represents the finest grid, while m_1 is the level at the bottom of the V-cycle. Notice that m_1 is an input parameter to the scheme, and is not necessarily zero. If $m_1 > 0$, the "bottom" level is not solved "exactly", but rather just smoothed like all the other levels.

Algorithm 1 The multigrid V-cycle, for $\Omega^m = P(\Omega^{m+1})$.

Vcycle(m, m_1, m_2)
if $(m = m_2)$ **then**
  $R^m = \rho^m - L^m(\varphi^m)$
end if
if $(m > 0)$ **then**
  $e^m =$ **Smooth** $(e^m = 0, R^m)$
  $\varphi^m = \varphi^m + e^m$
  if $(m > m_1)$ **then**
    $R^{m-1} = \rho^{m-1} - L^{m-1}(\varphi^{m-1})$
    Vcycle($m-1$, m_1, m_2)
    $e^m = e^m +$ **Refine** (e^{m-1})
  end if
  $R^m = R^m - L^m(e^m)$
  $\delta^m =$ **Smooth** $(\delta^m = 0, R^m)$
  $\varphi^m = \varphi^m + \delta^m$
else
  Solve$L(e^m) = R^m$
end if

This feature is used later, when we extend our multigrid scheme to incorporate a limited form of adaptive mesh refinement.

3.2 Geometry Coarsening

In the following, we present an algorithm for coarsening a geometry specified according the definitions in Section 2.2. The coarsening procedure is recursive, in the sense that it takes an input fine graph, $G^f = \left(V^f, E^f\right)$, and its underlying index space, and generates a complete coarse graph, G^c. We assume a static geometry, so that the procedure need be carried out only once to generate the full hierarchy of irregular geometry graphs at the beginning of a computation. The refinement ratio, $r \in \mathcal{Z}^d$, is the refinement, by dimension, between G^c and G^f, with respect to the cell indices, $\mathcal{K}\left(V^f\right)$ and $\mathcal{K}(V^c)$. We restrict our implementation to the case, $r = 2^i(2,\ldots,2)$, where $i \in \mathcal{Z}$ and discuss only the case $r = (2,\ldots,2)$, since the rest of the set we allow can be generated by recursive application. Generally speaking, multigrid performs most efficiently when the levels are separated by a constant factor of 2, unless there are geometrical or physical effects driving anisotropic transport. The scheme is trivially extended to arbitrary r, including directionally biased refinement, but such details detract from the presentation.

The procedure for generating G^c from G^f consists of three basic steps. First, we augment the fine graph to include all full cells that will be merged into the new coarse map (see Figure 10). Next, for every coarse index, $\mathbf{i}^c$, we build lists of connected components from the fine nodes, v^f, such that $\mathcal{K}\left(v^f\right) = \mathbf{i}^f$, where $\mathbf{i}^c = \lfloor\frac{1}{2}\mathbf{i}^f\rfloor = \left(\lfloor\frac{\mathbf{i}^f_1}{2}\rfloor,\ldots,\lfloor\frac{\mathbf{i}^f_d}{2}\rfloor\right)$ (the operator, $\lfloor x\rfloor$, returns the largest integer d-tuple, such that each component is less than the

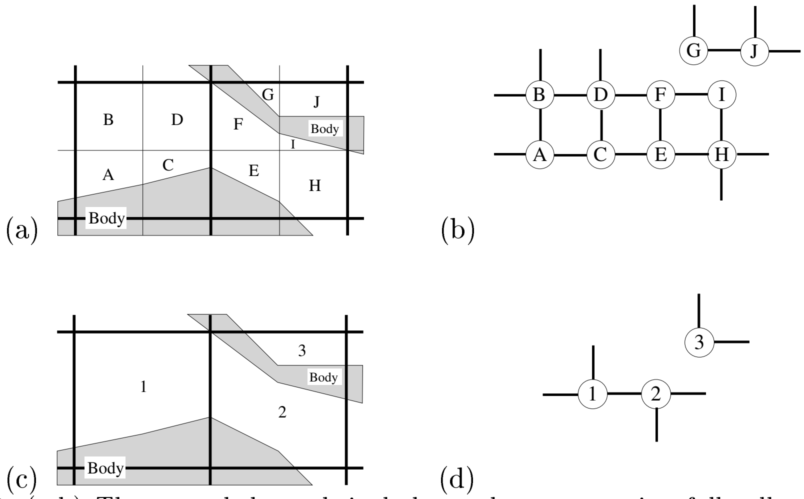


Figure 10: (a-b) The extended graph includes nodes representing full cells, such as B and D. Also, the extended graph includes edges between the full cells, such as B-D, as well as interface faces, such as A-B, C-D, D-F. (c-d) The coarsened geometry and graph, where the path A-B-C-D has becomes coarse node 1, E-F-H-I has become 2, and G-J has become 3. The edge 1-2 is created by the coarsening procedure.

corresponding component in x). Each connected component generates a new node in the coarse graph. Finally, the edge list is assembled to connect the new coarse nodes. Some auxiliary information needed by the algorithm is generated on the fly, as will be discussed below.

We can discuss each step in detail, after defining some useful notation.

- The index set of the cells in the fine graph, $\mathcal{I}^f = \left\{\mathcal{K}(v) : v \in V^f\right\}$.
- The fine-to-coarse projection, $\mathcal{P}^{fc} : \mathcal{Z}^d \mapsto \mathcal{Z}^d$, takes a fine index, and returns a coarse index, such that $\mathcal{P}^{fc}\left(\mathbf{i}^f\right) = \lfloor \frac{1}{2}\mathbf{i}^f \rfloor$.
- The coarse-to-fine projection takes a coarse index and returns a set of fine indices, $\mathcal{I}^{cf}(\mathbf{i}^c) = \left\{\mathbf{i}^f : \mathcal{P}^{fc}\left(\mathbf{i}^f\right) = \mathbf{i}^c\right\}$ of fine indices associated with a coarse index. For each $\mathbf{i}^f \in \mathcal{I}^{cf}(\mathbf{i}^c)$, if $\mathbf{i}^f \notin \mathcal{F}$, then $\mathbf{i}^f \in \mathcal{I}^f$ (i.e. the operator does not return indices fully covered by the embedded boundary).
- The index set spanned by the coarse graph, $\mathcal{I}^c$, is the union of fine indices created by coarsening, then refining, the index set $\mathcal{I}^f$,$\mathcal{I}^c = \bigcup_{v \in V^f} \mathcal{I}^{cf}\left(\mathcal{P}^{fc}\left(\mathbf{i}^f\right)\right), \forall\, \mathbf{i}^f \in \mathcal{I}^f$.
- The index set of full cells to add to the fine graph is therefore, $\mathcal{I}^{Ext} = \mathcal{I}^c - \mathcal{I}^f$
- The set of full cells, $\mathcal{N}(v) \subset \mathcal{Z}^d$, neighboring a node, v is defined from the index set $\mathcal{L}(v)$ as $\mathcal{N}(v) = \{\mathcal{K}(v) + o : \forall\, o \in \mathcal{L}(v)\}$. The set $\mathcal{L}(v)$ must be provided as input at the finest level; for coarser levels, the set is generated by the algorithm.
- The set of partial cells at the fine level to be associated with the coarse node, v, is $\mathcal{L}^{cf,I}(v)$. The corresponding set of full fine cells to be associated with v is $\mathcal{L}^{cf,R}(v)$.

These are built during the coarsening procedure, and are useful when transferring state data between refinement levels (see Section 3.3).

The extended graph, G^{aug}, is created by adding each index, $\mathbf{i} \in \mathcal{I}^{Ext}$ into the list of nodes, V^{aug}, removing $\mathbf{i}$ from all the lists, $\mathcal{N}(v), v \in V^{aug}$, creating new edges connecting this cell to the graph, and building a new $\mathcal{N}$ map entry for this cell. Algorithm 2 details this procedure. In Figure 10, cells labeled B and D are to be added to V^f, and the edges (A, B), (C, D), (B, D) and (D, F) are added to E^f to obtain G^{aug}.

Algorithm 2 Creating the extended graph, $G^{aug} = (V^{aug}, E^{aug})$.

```
Initialize (V^aug, E^aug) = G^f
for all i ∈ I^Ext do
  V^aug ← V^aug ⋃ {v^new}
  K(v^new) = i
  Λ(v^new) = 1
  N(v^new) = {}
  for k = 1,d do
    if ∃v : i ∈ N(v) then
      if i − K(v) = u_k then
        N(v) ← N(v) − {i}
        e = (v, v^new)
        A(e) = 1
        E^aug ← E^aug ⋃ {e}
      else if i − K(v) = −u_k then
        N(v) ← N(v) − {i}
        e = (v^new, v)
        A(e) = 1
        E^aug ← E^aug ⋃ {e}
      else
        N(v^new) ← N(v^new) ⋃ {i − u_k}
      end if
    end if
  end for
end for
```

The graph, G^c is created by coarsening the extended graph, G^{aug}, as detailed in Algorithm 3. Here, we build all the connected components at coarse index, $\mathbf{i}^c$, of an undirected subgraph of G^{aug}, using all the nodes, $v \in V^{aug}$ such that $\mathcal{P}^{fc}(\mathcal{K}(v)) = \mathbf{i}^c$. A new coarse cell is created for each of these connected paths, and the volume fraction of the new cell is such that its volume is the sum of the volume of its constituent full and partial cells. In Figure 10(a), the full coarse cell on the right contains two connected components, G-J, and E-H-I-F, which give rise to coarse cells 2 and 3 in Figure 10(c).

For each new coarse cell created, a subset of fine edges, $E^{k,\pm}_{vc} \subset E^{aug}$ is identified that connects the subgraph in the coordinate direction, k, to the remainder of the grid. The $\pm$ symbol indicates whether this set of edges is on the low ($-$), or high ($+$) side of the new

Algorithm 3 Creating nodes, V^c, of the coarsened graph, $G^c = (V^c, E^c)$, from the augmented graph, G^{aug}.

Initialize $V^c = \{\}$
for all $\mathbf{i}^c \in \mathcal{I}^c$ **do**
  $V = \left\{v \in V^{aug} : \mathcal{P}^{fc}\left(\mathcal{K}\left(v\right)\right) = \mathbf{i}^c\right\}$
  $E = \{e \in E^{aug} : e_- \in V \wedge e_+ \in V\}$
  for all connected components, (V_α, E_α) of (V, E) **do**
    $V^c \leftarrow V^c \bigcup \{v^{new}\}$
    $\mathcal{K}\left(v^{new}\right) = \mathbf{i}^c$
    $\Lambda\left(v^{new}\right) = \frac{1}{2^d} \sum_{v^f \in V_\alpha} \Lambda\left(v^f\right)$
    $\mathcal{L}^{cf,R}\left(v^{new}\right) = \mathcal{L}^{cf,I}\left(v^{new}\right) = \{\}$
    for all $v \in V_\alpha$ **do**
      if $\mathcal{K}\left(v\right) \in \mathcal{I}^{Ext}$ **then**
        $\mathcal{L}^{cf,R}\left(v\right) = \mathcal{L}^{cf,R}\left(v\right) \bigcup \{\mathbf{i}\}$
      else
        $\mathcal{L}^{cf,I}\left(v\right) = \mathcal{L}^{cf,I}\left(v\right) \bigcup \{v^{new}\}$
      end if
    end for
  end for
  for $k = 1, d$ **do**
    $E^{k,\pm}_{v^c} = \{e_\mp \in V_\alpha, e_\pm \notin V_\alpha, \mathcal{K}\left(e_+\right) - \mathcal{K}\left(e_-\right) = u_k\}$
  end for
end for

coarse cell. Each unique edge subset, $E_{v^c}^{k,\pm}$, generates a new coarse edge in E^c, as detailed in Algorithm 4. The new edge may be defined once we find two cells pointing to the identical fine-edge subset. The aperture of the new edge is such that the surface area of the coarse edge is equal to the sum of its constituent fine edges. In Figure 10, the fine edge subset, $E_2^{0,-}$, associated with coarse cell 2 in the 0-direction on the low side is $\{(D,F),(C,E)\}$. This is identical to $E_1^{0,+}$, so a new edge, $(1,2)$, is added to the coarse graph.

Algorithm 4 Creating edges, E^c, of the coarsened graph, $G^c = (V^c, E^c)$ from the augmented graph, G^{aug}.

Initialize $E^c = \{\}$
for all $v^c \in V^c$ **do**
  for $k = 1, d$ **do**
    for all $v^{C'} : \mathcal{K}(v^c) - u_k = \mathcal{K}\left(v^{C'}\right)$ **do**
      $E = E_{-,v^c}^{B,k} \bigcap E_{+,v^{C'}}^{B,k}$
      if $E \neq \{\}$ **then**
        $e^{new} = \left(v^{C'}, v^c\right)$
        $E^c \leftarrow E^c \bigcup \{e^{new}\}$
        $\mathcal{A}(e^{new}) = \frac{1}{2^{d-1}} \sum_{e \in E} \mathcal{A}(e)$
      end if
    end for
  end for
end for

The coarsening strategy is trivial for the grid completely in the regular part of the domain (i.e. at $\mathbf{i}^c : \mathbf{i}^f \in \mathcal{F}, \forall\, \mathbf{i}^f \in \mathcal{I}^{cf}(\mathbf{i}^c)$). Finally, the coarse full-cell map, $\mathcal{F}^c$, is created using the existing fine full-cell map, $\mathcal{F}^f$, according to the following criteria:

$$\mathbf{i}^c \in \mathcal{F}^c \text{if } \nexists\, \mathbf{i}^f \in \mathcal{I}^{cf}(\mathbf{i}^c) : \mathbf{i}^f \in \mathcal{F}^f \tag{13}$$

Notice that within our coarsening strategy, floating-point data, such as apertures and volumes, are not used explicitly to determine the merging process. The procedure we have outlined can be used to coarsen an input geometry to $G^c = (V^c, E^c)$, where $\mathcal{K}(v^c) = (0, \ldots, 0), \forall\, v^c \in V^c$, and accommodates multiple dimensions and arbitrary complexity. Since we are concerned only with the aspects of the geometry that appear in Equation 3 (cell areas, volumes and connectivity), we do not require the ability to reconstruct the embedded surface. In particular, no "blunting" is necessary, and we retain maximal geometric fidelity.

Also, notice that we have not designed our coarsening strategy to construct connected paths of solid; there is no determined way to distinguish parts of the solid in a coarse cell that were derived from specific regions of the fine description. This would be an issue for applying inhomogeneous boundary conditions, except that we solve in correction form to avoid requiring such information—the boundary conditions for the correction problem are homogeneous). It follows then that our scheme cannot easily be extended to Full-Approximation-Storage versions of multigrid, for example (useful for nonlinear elliptic problems).

3.3 Smoothing, Coarsening, Refining the State

Point relaxation for Equation 11 iterates on the expression, $\varphi^{n+1} = \varphi^n + \lambda\left(L\left(\varphi^n\right) - \rho\right)$, where λ is a relaxation parameter, n is an iteration counter and φ^n is an approximation to the solution, φ. For each cell, we choose λ such that the expression for φ^{n+1} does not contain φ^n at that cell.

The relaxation parameter, λ_v, on the irregular cells, $v \in V$, is obtained by summing the derivative of each term in Equation 6 with respect to φ_v^I, the value of the state in the irregular data structure at node v. We can generalize Equation 10 so that the derivative of the flux with respect to the cell-centered state is

$$\frac{\partial F^I\left(e\right)}{\partial \varphi_v^I} = \pm\frac{w_v^e}{h}, \text{ for } e : e_{\mp} = v$$

The weighting, w_v^e, depends on the dimension, d, of the problem. For $d = 2$, $w_v^e = \frac{1}{2}\left(\mathcal{A}\left(e\right) + \mathcal{A}\left(e'\right)\right)$, where $\mathcal{A}\left(e'\right)$ is the aperture of other edge involved in the flux interpolation (or, the average of the apertures, if there are more than one). The relaxation parameter becomes

$$\lambda_v = \left[\frac{1}{\Lambda\left(v\right)h^2}\left(\sum_{e:e_{\pm}=v} w_v^e \mathcal{A}\left(e\right) + \sum_{o\in\mathcal{L}\left(v\right)} 1\right)\right]^{-1} \tag{14}$$

whereas on the regular cells, $\mathbf{i} \in \mathcal{F}$, the expression reduces simply to

$$\lambda_{\mathbf{i}} = \frac{h^2}{2d} \tag{15}$$

Over the regular cells, we order the pointwise updates with a multi-coloring scheme (red-black "checker-boarding" for $d = 2$) based on cell index for vectorization efficiency. We update all the irregular cells simultaneously between each colored sweep over the regular cells. The combination (red sweep, irregular update, black sweep, irregular update) counts as a single "smoothing" pass of the point relaxation operator.

The **Smooth** operations in Algorithm 1 consist of two or more iterations of the above sequence, while the **Solve** operation iterates the sequence to numerical convergence (the number of iterations required is on the order of the number of unknowns at that level). Typically, the **Solve** operation is carried out only on the coarsest multigrid level.

The level transfer operations, **Coarsen** and **Refine** are defined using the cell-to-cell-subset maps defined in Section 3.2. A volume-weighted averaging **Coarsen** operation for irregular data at node v^c is

$$\varphi_{v^c}^I = 2^{-d}\left(\sum_{v^f\in\mathcal{L}^{cf,I}\left(v^c\right)} \varphi_{v^f}^I \Lambda\left(v^f\right) + \sum_{\mathbf{i}^f\in\mathcal{L}^{cf,R}\left(v^c\right)} \varphi_{\mathbf{i}^f}^R\right) \tag{16}$$

where $\mathcal{L}^{cf,R}\left(v^c\right)$ and $\mathcal{L}^{cf,I}\left(v^c\right)$ are, respectively, the regular and irregular fine nodes that coarsen into v^c, as defined in Algorithm 4. For regular data at index $\mathbf{i}^c$,

$$\varphi_{\mathbf{i}^c}^R = \sum_{\mathbf{i}^f:\mathbf{i}^f\in\mathcal{I}^{cf}\left(\mathbf{i}^c\right)\bigwedge\mathbf{i}^f\in\mathcal{F}} 2^{-d} \varphi_{\mathbf{i}^f}^R \tag{17}$$

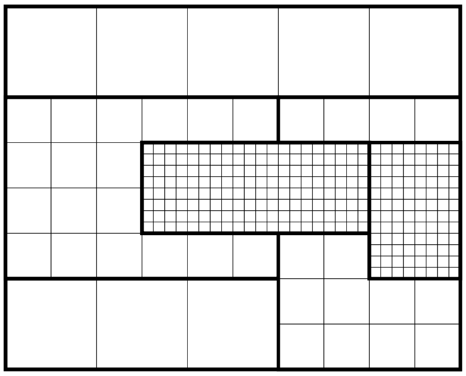


Figure 11: Properly nested unions of rectangular grid patches for cell-based data in 2D. The refinement ratio between AMR levels is $2^n, n$ is a small positive integer. The refined patches at any level may touch the boundary of the computational domain, but coarse-fine boundaries are buffered with at least one layer of cells at the next coarser level.

A piecewise-constant **Refine** operation for irregular data is constructed with the cell-to-cell-subset maps as

$$\varphi^I_{v^f \in \mathcal{L}^{cf,I}(v^c)} = \varphi^I_{v^c} \tag{18}$$

The corresponding piecewise-constant refine for regular data is

$$\varphi^R_{\mathbf{i}^f \in \mathcal{F}^f} = \begin{cases} \varphi^I_{v^c} & \text{if} \quad \mathbf{i}^f \in \mathcal{L}^{cf,R}(v^c) \\ \varphi^R_{\mathbf{i}^c} & \text{if} \quad \mathbf{i}^f \in \mathcal{I}^{cf}(\mathbf{i}^c) \bigwedge \mathbf{i}^f \notin \mathcal{L}^{cf,R}(v^c) \ \forall v^c \in V^c \end{cases} \tag{19}$$

Note that $\varphi^R_{\mathbf{i}^f}$ is not defined for $\mathbf{i}^f \notin \mathcal{F}^f$, since those full cells are either cut or covered by the embedded solid.

4 Adaptive Mesh Refinement

The regular component of the geometry description in Section 2.2 was built on rectangular patches of uniform gridding over the large portion of Ω that is not adjacent to the embedded boundary. This aspect, and the structure of the coarsening machinery used to generate the multigrid mesh hierarchy, make it straightforward to extend our scheme to incorporate block-structured adaptive mesh refinement (AMR) over the regular parts of the domain. The scheme is related closely to that described in [34].

The AMR rectangular grid hierarchy is composed of different levels, ℓ, of refinement, ranging from coarsest, at $\ell = 0$, to the finest, at $\ell = \ell_{hi} \geq 0$. These levels will correspond to a subset of the multigrid levels previously discussed. The domain at each AMR level, Ω^ℓ, is represented as a union of rectangular grid patches of a given resolution, accompanied by a graph of the irregular cells. The rectangular grids are properly nested, in the sense that the union of the grid patches at level $\ell + 1$ are contained in the union of grids at level ℓ for $0 \leq \ell < \ell_{hi}$ (see Figure 11). Furthermore, except at physical boundaries, the union of level ℓ grids is large enough to guarantee that there is a border at least one level ℓ cell wide surrounding each $\ell + 1$ grid. Grids at all levels are allowed to extend to the

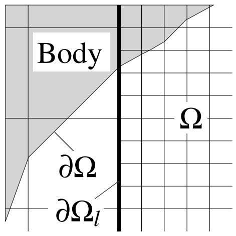


Figure 12: The case where the embedded boundary, $\partial\Omega$, intersects the coarse-fine boundary, $\partial\Omega^{\ell}$, between AMR levels, ℓ and $\ell - 1$. The AMR implementation presented here does not allow for this condition.

regular physical boundaries. We restrict this implementation under the condition that the irregular geometry, $G^{\ell} = \left(V^{\ell}, E^{\ell}\right)$, at level ℓ, be completely contained within the union of rectangular patches at level ℓ (see Figure 12). Thus, $\mathcal{K}\left(v^{\ell}\right)$ falls within the bounds of the patches for every $v^{\ell} \in V^{\ell}$. In short, this restriction specifies that the embedded boundary will be discretized at the finest grid level.

The extent of the rectangular patches of regular gridding may be fixed throughout the calculation, or modified as the calculation proceeds so as to focus computational resources where resolution is required. In the latter solution-adaptive applications, error estimation techniques, such as Richardson extrapolation, are used to tag cells where the local error is above a given tolerance. The tagged cells are grouped into rectangular patches using the clustering algorithm given in [36], and refined to form the grids at the next level. The process is repeated until either the error tolerance criteria are satisfied, or a specified maximum refinement level is reached. Upon entering the iterative solver, the initial guess data may be used to create the grids at level 0 through ℓ_{hi}. As the guessed state is relaxed toward the solution, a re-gridding algorithm may be called periodically. When new grids are created at level $\ell + 1$, the data on these new grids are copied from the previous grids at level $\ell + 1$, if possible, otherwise interpolated in space from the underlying level ℓ grids. In all cases, the newly generated fine-level grids must be properly nested.

4.1 Multi-level V-cycle

In order to extend our embedded boundary multigrid Poisson solver to this limited AMR framework, we augment our discretization and V-cycle to incorporate that $P\left(\Omega^{\ell}\right) \subset \Omega^{\ell-1}$. We begin with the initial set of AMR levels on which we want the solution, and construct *intermediate multigrid levels* between and below the AMR levels so that adjacent pairs of levels are related by a refinement ratio of 2. These new levels are for use by the multigrid solver alone, and are discarded when the solution is complete. Each new multigrid level is created by coarsening the next finer level above, and does not communicate with coarser AMR levels below. Let $m = m\left(\ell\right)$ be the multigrid level corresponding to a given AMR level ℓ. (Note that $m\left(\ell_{hi}\right) = m_{hi}$.) For all intermediate multigrid levels, $m\left(\ell\right) < m < m\left(\ell + 1\right)$, $\Omega^{m} = P(\Omega^{m+1})$, i.e. the coarsened domain covers the same region of the physical domain as does the source fine domain.

The multi-level residual, $R \equiv \rho - L\left(\varphi\right)$, is defined everywhere to be the residual on the finest grid available. For every level, $\ell < \ell_{hi}$, the residual for the region covered by $P\left(\Omega^{\ell+1}\right)$ is ignored. The multigrid relaxation is initiated by invoking the recursive V-cycle smoother on the finest level, ℓ_{hi}, which in turn calls a V-cycle smoother on the next level. Note that the next level may be an AMR level, or it may be simply a multigrid level.

The solution at level ℓ sees the coarse solution through the interface, $\partial\Omega^{\ell}$, between Ω^{ℓ} and $\Omega^{\ell-1}$ (excludes the physical boundary). Additionally, if $\ell < \ell_{hi}$, the solution on Ω^{ℓ} sees also the finer data through the interface $\partial\Omega^{\ell+1}$. We define the full three-level discrete Laplacian operator, $L^{\ell}\left(\varphi^{\ell+1}, \varphi^{\ell}, \varphi^{\ell-1}\right)$ to incorporate the fine fluxes at $\partial\Omega^{\ell+1}$, and and coarse data at level $\ell-1$ along $\partial\Omega^{\ell}$, as discussed in Section 4.2. We also define a "no-fine" operator, $L^{\ell,nf}\left(\varphi^{\ell}, \varphi^{\ell-1}\right)$, which uses the coarser data at $\partial\Omega^{\ell}$, but ignores level $\ell+1$ data, and applies a homogeneous boundary condition on all physical boundaries. In order to use the no-fine operator, we construct the level ℓ correction problem in the region $P\left(\Omega^{\ell+1}\right)$ by coarsening the level $\ell+1$ residual. In this way, the level ℓ correction is "aware" of the progress made on level $\ell+1$ without ever requiring the full three-level operator, except to compute the initial residual at each level. The complete AMR V-cycle appears

Algorithm 5 The multigrid V-cycle, for $P\left(\Omega^{\ell}\right) \subset \Omega^{\ell-1}$.

AmrVcycle(ℓ)
if $(\ell = \ell_{hi})$ **then**
  $R^{\ell} = \rho^{\ell} - L^{\ell}\left(\varphi^{\ell}, \varphi^{\ell-1}\right)$
  $e^{\ell} = 0$
end if
if $(\ell > 0)$ **then**
  $\varphi^{\ell}_{save} = \varphi^{\ell}$
  $e^{\ell} = \mathbf{Smooth}\left(e^{\ell}, R^{\ell}\right)$
  $\varphi^{\ell} = \varphi^{\ell} + e^{\ell}$
  $e^{\ell-1} = 0$
  $$R^{\ell-1} = \begin{cases} \mathbf{Coarsen}\left(R^{\ell} - L^{\ell,nf}\left(e^{\ell}, e^{\ell-1}\right)\right) & \mathbf{On}\ P\left(\Omega^{\ell}\right) \\ \rho^{\ell-1} - L^{\ell-1}\left(\varphi^{\ell}, \varphi^{\ell-1}, \varphi^{\ell-2}\right) & \mathbf{On}\ \Omega^{\ell-1} - P\left(\Omega^{\ell}\right) \end{cases}$$
  AmrVcycle($\ell-1$)
  $e^{\ell} = e^{\ell} + \mathbf{Refine}\left(e^{\ell-1}\right)$ **On** $P\left(\Omega^{\ell}\right)$
  $R^{\ell} = R^{\ell} - L^{\ell,nf}\left(e^{\ell}, e^{\ell-1}\right)$
  $\delta^{\ell} = \mathbf{Smooth}\left(\delta^{\ell} = 0, R^{\ell}\right)$
  $e^{\ell} = e^{\ell} + \delta^{\ell}$
  $\varphi^{\ell} = \varphi^{\ell}_{save} + e^{\ell}$
else
  Solve $L\left(e^{\ell}\right) = R^{\ell}$
  $\varphi^{\ell} = \varphi^{\ell} + e^{\ell}$
end if

in Algorithm 5. If $\exists m : m(\ell-1) < m < m(\ell)$, then we augment the multigrid cycle to

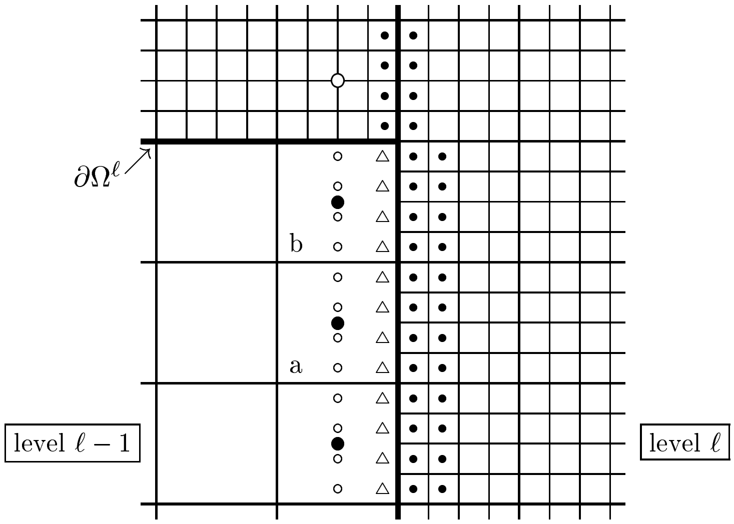


Figure 13: A typical coarse-fine interface, $\partial\Omega^\ell$, for $d = 2, r = 4$. The heaviest lines indicate fine grid boundaries. Locations are shown for coarse grid boundary data (●), tangentially interpolated values (∘), fine grid cell-centers (•) and perpendicularly interpolated ghost cell values (△). Interpolated coarse grid boundary data (○) is computed using Equation 20.

smooth on the multigrid levels between the AMR levels, and bypass the bottom solve. This modification is effected by replacing the recursive call to **AmrVcycle**$(\ell-1)$ in Algorithm 5, with a call to **Vcycle**$(\ m(\ell)-1, m(\ell-1), m(\ell)\ )$.

4.2 Coarse-Fine Matching

A coarse-fine interface, $\partial\Omega^\ell$, separates the regions $P\left(\Omega^\ell\right)$ and $\Omega^{\ell-1} - P\left(\Omega^\ell\right)$. The fine grid solution is connected to the coarse data through this interface so that it can properly "feel" the boundary conditions on $\partial\Omega$, using a procedure closely related to that presented in [37]. The fine grid feels the coarse solution via Dirichlet boundary conditions by interpolating the coarse data adjacent to $\partial\Omega^\ell$. The coarse grid likewise feels the fine solution through a procedure that replaces the coarse flux on $\partial\Omega^\ell$ with the appropriate sum of constituent face fine fluxes. The following two sections discuss each of these operations in detail.

4.2.1 Fine Grid Boundary Conditions

At each level of the multigrid V-cycle (i.e. each multigrid level m), colored sweeps of the point relaxation are performed on rectangular grids sequentially, with the boundary conditions effectively imposed once per sweep. For convenience, the coarse-fine boundary conditions are represented in the operator as Dirichlet values in ghost cells immediately outside the fine grids (to locations represented by triangles in the two-dimensional example shown in Figure 13). For a given fine grid, each ghost cell value is copied from another fine grid, or interpolated using the coarse grid data. Once the ghost cell values have been filled, the Laplacian operator may be computed as specified in Equations 3 and 5 for all fine cells in the rectangular grid patch.

The interpolation (for $d = \{2, 3\}$) is performed in two separate steps. First, a quadratic interpolation tangential to the face of the fine grid gives values at the locations identified in the example, Figure 13, by small open circles. Next, a quadratic interpolant is constructed normal the interface, using the cell-centered fine grid data (small solid circles), and tangentially interpolated data, to fill in data in the ghost cell locations. The multi-dimensional interpolation must be updated after each time the fine or coarse data is modified, since the ghost cell value is affected by both profiles.

Sufficient coarse data exists to easily compute a parabola through the coarse data for the tangentially interpolated values that lay in the coarse cell labeled "a" in Figure 13. For cell "b" however, the upper point is covered by fine grid, and therefore contains invalid data. In [37], a one-side linear tangential interpolation was constructed in this case using only the valid coarse data. We improve on that concept by generating an accurate coarse value in the covered coarse cells (large open circle in Figure 13), so that the parabolic interpolant may be constructed as before. The generated coarse cell data is based on the covering fine data, using a third-order interpolant:

$$\varphi_{\bigcirc}^{\ell-1} = \sum_{j \in nbhd(\circ)} \varphi_j^{\ell} + \frac{1}{2}\nabla^2 \varphi_{\bigcirc}^{\ell} + \mathcal{O}\left(\left(h^{\ell}\right)^3\right) \tag{20}$$

where the sum is taken over the fine grid cells adjacent to location marked $(\circ)$, and the Laplacian correction term, $\nabla^2\varphi_{\bigcirc}^{\ell}$, is computed as the average of the simple $(2d+1)$-pt numerical Laplacian computed on the 2^d fine cells surrounding the point marked $(\circ)$.

4.2.2 Level $\ell - 1$ fluxes along $\partial\Omega^{\ell}$

Local conservation is preserved along the coarse-fine interface, $\partial\Omega^{\ell}$, by ensuring that the same flux computed to enter the fine grid is counted to leave the coarse grid. The procedure for carrying this out can be specified after defining some additional notation. The coarse index, $\mathbf{i}^c$, at level $\ell - 1$, is *uncovered* if $\mathbf{i}^c \in \left(\Omega^{\ell-1} - P\left(\Omega^{\ell}\right)\right)$. Further, the uncovered index, $\mathbf{i}^c \in \mathcal{I}^{k,+}$, lays adjacent to $\partial\Omega^{\ell}$ in the k^{th}-direction if $\partial\Omega^{\ell}$ borders the cell at $\mathbf{i}^c$ on its high-side. The cell at $\mathbf{i}^c \in \mathcal{I}^{k,-}$ lays adjacent to the coarse-fine boundary if $\partial\Omega^{\ell}$ borders its low-side. For example, in Figure 13, each coarse cell marked with a large bullet, $(\bullet)$, is a member of the set, $\mathcal{I}^{0,+}$ at level $\ell - 1$. For every $\mathbf{i}^c \in \mathcal{I}^{k,\pm}$, there is a face set, $\mathcal{S}\left(\mathbf{i}^c\right)$ at level ℓ such that the sum of the faces, $s \in \mathcal{S}\left(\mathbf{i}^c\right)$, covers entirely the coarse face at $\mathbf{i}^c \pm {}^1\!/_2 u_k$.

We incorporate the fine fluxes into the coarse discretization at level $\ell - 1$ by building the conservation sum on the coarse cell in two passes. In the first pass, the coarse fluxes are computed and summed as if the level ℓ fine grid were not present. For the correction pass, we compute the fine fluxes along $\partial\Omega^{\ell}$ according to the prescription in Section 4.2.1. Then, we use the following expression to overwrite coarse fluxes at level $\ell - 1$ on $\partial\Omega^{\ell}$ for each coordinate direction, k:

$$
\begin{aligned}
\left(\nabla\cdot\vec{F}\right)_{\mathbf{i}^c\in\mathcal{I}^{k,\pm}} &= \left(\nabla\cdot\vec{F}\right)_{\mathbf{i}^c} + \\
&\frac{1}{\left(h^{\ell-1}\right)^d}\left(\sum_{s'\in\mathcal{S}(\mathbf{i}^c)}\left(\vec{F}\cdot\hat{n}\right)_{s'}\left(h^{\ell}\right)^{d-1} - \left(\vec{F}\cdot\hat{n}\right)_{\mathbf{i}^c\pm u_k}\left(h^{\ell-1}\right)^{d-1}\right)
\end{aligned} \tag{21}
$$

That is, we remove the extensive contribution from the underlying coarse edge, and replace it with the sum of extensive fluxes on the contributing fine edges. In this operation, the cells in $\Omega^{\ell-1} - P\left(\Omega^{\ell}\right)$ become effectively isolated from the cells in $P\left(\Omega^{\ell}\right)$.

5 Implementation and Geometrical Requirements

The fundamental irregular data representation, the graph, $G = (V, E)$ of irregular cells, is implemented in our codes as two lists, one for the cell fragments, and one for the edge fragments. These lists are produced by a "geometry generator" module, according to requirements of the algorithms presented in earlier sections of this paper. The geometry generator is discussed in Section 5.1. In short, for every partial cell, v, in the domain, we must store the following information:

- The partial cell volume fraction, $\Lambda(v)$,
- The set of full cells, $\mathcal{N}(v)$, neighboring v, specified as a list of integer d-tuples,
- The index, $\mathcal{K}(v)$, of the full mesh cell containing v, specified as a d-tuple

For each partial edge, e, in the domain, we need the following information:

- The partial edge area fraction, $\mathcal{A}(e)$
- The coordinate direction of the unit vector normal to the edge
- An identifier of the cell on either side of the edge

Additionally, for the nodes, v_i in all levels but the coarsest, we require the lists, $\mathcal{L}^{cf,I}(v_i)$ and $\mathcal{L}^{cf,R}(v_i)$, as discussed in Algorithm 3. Finally, we require a method of testing whether an index, $\mathbf{i}$, lies within the set of full fluid cells, i.e. if $\mathbf{i}\in\mathcal{F}$. For most problems, the total number of solid and partial cell in the domain is much smaller than the number of full cells. Instead of maintaining a list of all the full cells, we generate a list of the solid cells, $\mathcal{B} : \mathbf{i}\in\mathcal{F}$ iff $(\nexists v\in V : \mathcal{K}(v) = \mathbf{i})$ and $(\nexists\mathbf{i}\in\mathcal{B})$. Generation of $\mathcal{B}$, is described in the Section 5.1.4. The *regular* data is stored in block-structured arrays on a union of rectangles for each refinement level using the BoxLib [BoxLib96] software library. The two distinct data structures communicate via the "interface faces" described in Section 2.2.

5.1 Geometry Generation

In general, the procedure for generating Embedded Boundary geometries consists of the following steps: intersect the surface description with the background uniform Cartesian

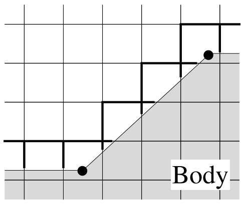
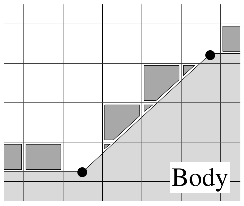
(a)  (b) 

Figure 14: Creation of the cell fragments in two dimensions. (a) Along each line segment, created by connecting successive nodes in the specified list, edge fragments are generated from the grid line intersections. (b) Once the edges are known, cell fragments can be generated. Cell fragments surrounding the nodes are created in a subsequent step of the algorithm

mesh; compute partial cell areas and cell fragment volumes; and establish connectivity of the cells to each other and to the full cells. Although a general implementation of this procedure has been presented for complex three dimensional geometries[14], we introduce a simpler scheme for two dimensions which requires considerably less effort to implement, yet is sufficient for our purposes.

Our procedure is similar to a two-dimensional scheme[16] presented for constructing an Embedded Boundary representation of bodies which are specified as unions of Bezier curves. We extend this idea by allowing for a nearly arbitrary collection of two-dimensional polygons (vertex lists with an assumed orientation). The vertices of these polygons may be generated, for example, by evaluating the parameterized Bezier curves and line segments used in PostScript-compatible computer drawing software, or any other user-specifiable function. The only constraint is that no mesh cell in the background fine-level Cartesian grid may contain more than one of these vertices totally within it.

By convention, as the input vertex list is traversed, the body lies to the left of the segments connecting successive nodes. Except in the case of a "polyline" (discussed in Section 5.1.3), the polygon is closed by connecting the first point in the list to the last. Each vertex is specified by location, and whether the point lies exactly on any grid line or at a coordinate line intersection. The latter avoids difficulties associated with exact arithmetic on a finite-precision machine.

A list of mesh-line intersections is computed between each successive pair of vertices. The segment joining each successive pair of these new intersections will represent a portion of the embedded boundary, and will become the irregular boundary of a new cut cell. The grid-aligned partial edges of each of these new cut cells are easily constructed, and added to a master list. It is a simple matter in this setting to then determine which partial edges in the master list border the new cell fragment. Once the involved partial edges have been identified, the cell volume is computed using the scheme outlined in Appendix A. For each partial edge between newly created cell fragments, there is now enough information to complete the specification, including in particular the identity of the surrounding partial cells. After the input vertex list has been traversed, cell and edge fragments will exist that completely surround the polygon, except within and bordering mesh cells that contain original vertices (see Figure 14). The procedure for adding these final cell fragments and edges into the master list depends on whether the specified vertex is concave or convex. We

(a) (b)

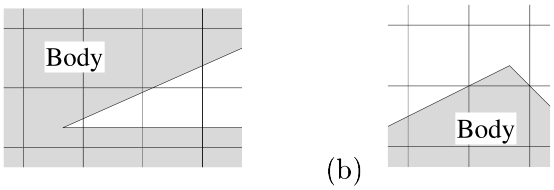


Figure 15: Classification of nodes along the polygon that specifies the embedded body. (a) A concave node, and (b) a convex node. Convex nodes are significantly easier to handle in the geometry generation procedures.

(a) (b)

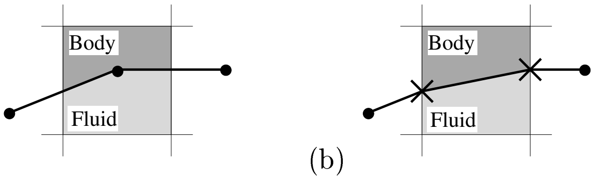


Figure 16: The two methods used in this paper for computing cell fragments. (a) The "natural" method: Compute the cell fragment according the the specified polygon, exactly. (b) The "blunted" method: Compute the cell fragment using the grid line intersection locations. The blunted method only builds cells that are exactly discretized by the finite-volume conservation sum. The piecewise-linear boundary interfaces allowed by the natural method are only approximated by the flux sum.

discuss the simpler convex case first.

5.1.1 Convex Polygons

If the i^{th} node, v_i, of the specified polygons is convex (i.e. if $\overline{v_{i-1}v_i} \times \overline{v_iv_{i+1}} \leq 0$, see Figure 15), there are at least two methods for computing the volume of the surrounding cell fragment: the volume may be defined explicitly by the polygon segments, or by the nearest grid line intersections (locally blunting the boundary shape–see Figure 16). The first option was implemented as the default in our scheme. The second option improved some of the convergence results, as detailed in Section 6, but it places severe limits on the generality of our scheme with respect to geometries containing fine scale surface concavity, as discussed in Section 5.1.2. We refer to the former option as the "natural" method, and the latter as the "blunted" method. Cell fragments encompassing a convex node may be constructed by generating the appropriate partial edge areas and computing the cell volume using one of the two methods shown in Figure 16. The edge fragments are simply added into the edge list as well, since all the necessary information (face area, neighboring cells) already exists.

5.1.2 Concave Nodes

If the node, v_i, is concave, the situation is a little more complex, as there is the possibility that one or more of the cell fragments defined in the first pass actually conflict with one another (see Figure 17). Since each was created without regard for the other, the two will

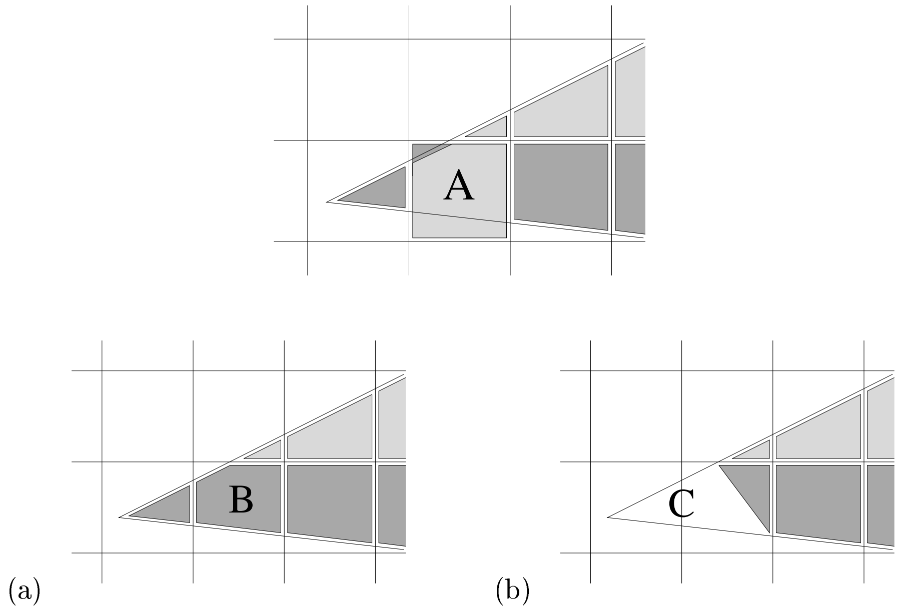


Figure 17: Resolving cell fragment conflicts that may arise near concave nodes, depending on whether the natural or blunted generation scheme is being used. (a) A pair of conflicting cell fragments exist at the mesh index marked “A”. (b) In the natural scheme, one of the cell fragments is removed, and the remaining one is trimmed away appropriately, leaving the cell fragment marked “B”. A new cell fragment is added at the apex of the node. (c) In the blunted scheme, cell and edge fragments are removed from the geometry until a cell fragment with non-zero volume can be constructed from a linear boundary segment.

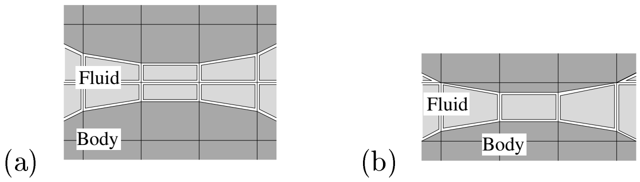


Figure 18: A thin region of fluid between two sections of the embedded solid. The sections may or may not be part of the same body. (a) No cell fragment conflicts arise with our present generation schemes. (b) The cell fragments between the body sections can be properly generated only after a global search procedure. We have not implemented such a search, and currently flag this condition as an input error.

overlap in space, and each will protrude through the irregular boundary of the other. Given the local node layout, we simply resolve the conflicting cell and edge fragment definitions based on the location of edge intersections near the concave node. In particular, we march away from the concave node by interval along the segment, $\overline{v_i v_{i+1}}$, searching for a mesh index containing more than one cell fragment. If two are found, we remove the fragment associated with the segment $\overline{v_i v_{i+1}}$, in favor of the one associated with $\overline{v_{i-1} v_i}$. We additionally update the cell pointers of the adjacent edges, and reduce the affected edge fragment apertures, and cell fragment volume. Finally, we add in the cell fragment at the apex of the node.

If we are to build the geometry according to the blunted method, the procedure above is modified. Firstly, if the segments, $\overline{v_{i-1} v_i}$ and $\overline{v_i v_{i+1}}$ intersect the same face of the mesh cell containing the concave node, then there can be no cell fragment surrounding that node. If this is the case, we must traverse the segments, $\overline{v_{i-1} v_i}$ and $\overline{v_i v_{i+1}}$, removing edge and cell fragments until we can properly construct a cell fragment with non-zero volume according the blunted method. This process will minimize grid-scale concave features of the body, and could be a strong function of exactly where the body is placed on the mesh grid.

As presently implemented, our gridding scheme can resolve conflicting cell fragment definitions coming only from adjacent line segments in the polygon description. This limits large-scale convexity to cases where non-adjacent segments of the polygon remain separated by at least one mesh grid line. For the same reason, multiple bodies in the same calculation must remain separated by a grid line as well (see Figure 18). This limitation is easily removed by expanding the search for conflicting cell fragments to include the entire set, but the work of such a search would scale poorly with problem size, and cell and edge conflict resolution would become considerably more complex.

5.1.3 Infinitely Thin Shells

A special case easily allowed by our procedure is the "infinitely thin" body having its outline specified by an "open" polygon, or polyline. This is effected via the same procedures as above, except that after we construct the cell fragments along each line segment in the polygon, we reverse the point list, and repeat the procedure to generate cell fragments along the other side of the line. We truncate the polyline at the last intersection with the grid to avoid creating cell fragments around the first and last nodes of the polyline. Figure 19 illustrates such a situation. This is a "thin" body condition, as discussed in Section 2.1.1,

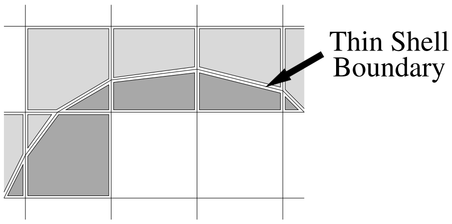


Figure 19: A infinitely thin shell geometry. Cell fragments can be generated along both sides of a a polyline; our schemes support multiple cell fragments at a given mesh cell index.

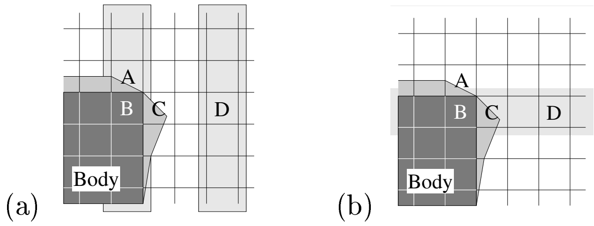


Figure 20: Determining the set of solid cells, $\mathcal{B}$, from the layout of cell and edge fragments. (a) The vertical strips for the x-direction sweep. The cell fragment at A has non-zero aperture on its high y-side, and zero aperture on its low y-side. Therefore, B, and all the indices below it, must be solid cells. There are no cell fragments in the vertical strip containing D, so no solid cells can be identified. (b) The horizontal strips for the y-direction sweep. The cell fragments in C identify D as a fluid cell.

and is accommodated naturally in our framework; in Section 6.2, we present results for one such geometry.

5.1.4 Set of Solid Cells

The simple procedure we use for identifying the solid cells is similar in spirit to that outlined in [16], except that we must allow for thin bodies (see Figure 20). We proceed after generating all the cell and edge fragments, by sweeping in one-dimensional strips. The figure illustrates the process in two-dimensions, though the scheme is valid in three dimensions as well. We begin with a vertical strip at the left side of the domain, and the assumption that all non-partial cells are full (non-solid), though we cannot determine *a priori* whether the bottom of the strip is inside the solid or the fluid until we reach the first index which contains cell fragments. We use the general logic that if one cell fragment in the set at that index has non-zero aperture on its low x face, then there can be no solid cell immediately below. Likewise, if a cell in the set has non-zero aperture on its high x face, then no solid cell can be immediately above. If, at the first index containing cell fragments, there are none in the set with non-zero aperture on their low x face, we add all mesh cells below that one to the solid cell list. We continue upward until finding an index with cell fragments where none have non-zero aperture on their high x face. All cells between that location, and the next with all cell fragments having zero low x aperture, are added to the solid set.

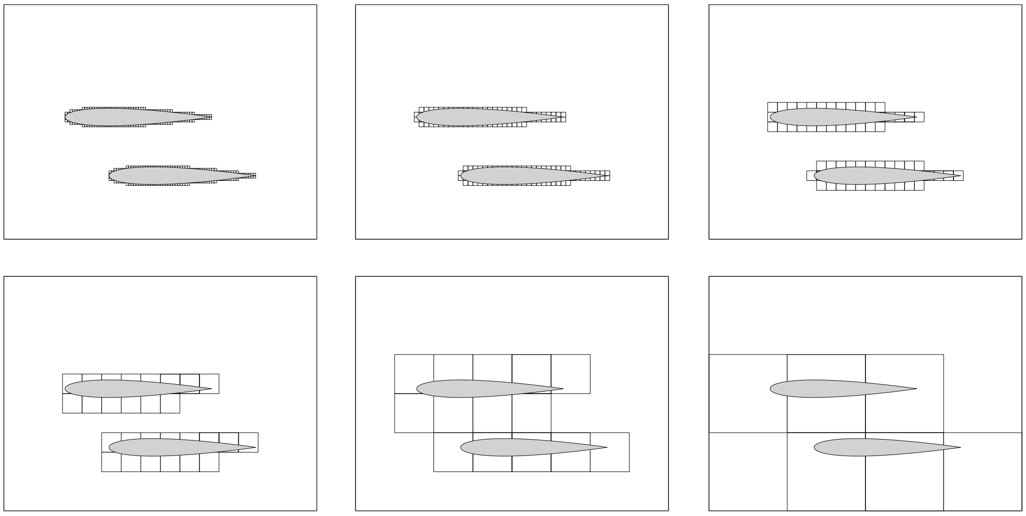


Figure 21: Example geometry coarsening, taking a 128×96 grid progressively down to a 4×3. Note that the volume and area fractions are consistent across levels.

Note that since we may traverse the entire strip without encountering a cell fragment, this single pass system may fail to identify solid cells which populate the entire strip. We now proceed with similar logic in y-strips, and if in three dimensions, finish with z-strips.

Algorithm 6 Ensuring solid cells are "properly bordered". Here, $\mathcal{O}$ is the list of possible orientation unit vectors.

$\mathcal{I} = \{\mathcal{K}(v) : v \in V\}$
while $\exists\, \mathbf{i} \in \Omega \,/\, \mathcal{B} \bigcup \mathcal{I} : \mathbf{i} + o \in \mathcal{B}$, for any $o \in \mathcal{O}$ **do**
  $\mathcal{B} = \mathcal{B} \bigcup \mathbf{i}$
end while

A final pass is required to eliminate any remaining ambiguities. We search for mesh cells not containing a cell fragment, and not marked for solid, but which are adjacent to solid. Since solid cells cannot be adjacent to fluid, if any such cells are found, they are tagged solid as well. Details of the scheme appear in Algorithm 6. This procedure is continued until there are no more mesh cells satisfying the condition. The algorithm is particularly inefficient, but only required if the strip passes reveal any solid cells adjacent to the rectangular computational domain boundary.

Once the complete geometry at the finest level has been generated, we may apply recursively the coarsening strategy defined in Section 3.2 to generate the coarser geometry descriptions required for the multigrid/AMR refinement levels. In Figure 21, we show an example two-dimensional geometry, as it is coarsened by our scheme. In the figure, the embedded body is shaded in, and the individual cell fragments are drawn. Note that the body shape is drawn in at the resolution of the finest grid; the volume and edges of the coarse cell fragments are consistent with this picture. We use the reconstruction algorithm detailed in Appendix A only for estimating a position to apply the Dirichlet boundary condition, as described in Section 2.3

6 Results

We present a variety of test cases which exercise different components of our adaptive multigrid linear elliptic solution scheme. In all cases, the domain is two-dimensional. The first sets of results are used to verify the consistency and accuracy of the discretization. Since the method is essentially identical to that presented in [33], we observe identical convergence behavior. Next, we look at the residual reduction performance of our multigrid scheme, using a variety of embedded boundary shapes and boundary conditions. We conclude with a demonstration and assessment of the adaptive aspects of the solver.

6.1 Convergence Verification

For the following cases, the embedded boundary is defined by the curve,

$$r = 0.30 + 0.15 \cos 6\theta$$

where r is radius, and θ is azimuthal angle about the origin, measured from the positive x-axis. The computational domain for these cases lies between this curve, and the unit box, centered at $(0, 0)$. Equation 1 is solved for the potential, φ, given a Poisson source

$$\rho = 7r^2 \cos 3\theta$$

The exact solution for this system is $\varphi^e(r, \theta) = r^4 \cos 3\theta$. The error field, $\xi(\vec{x}) = \varphi(\vec{x}) - \varphi^e(\vec{x})$ is used to monitor the convergence of the discrete solution to the correct continuum solution. The exact solution resides at the full cell centers, as discussed in Section 2.1. The truncation error field, $\tau(\vec{x})$, is the difference between the analytic Laplacian operator, and the numerically computed operator, $L(\varphi^e)$, defined in Section 3. The truncation field, as well as the Poisson source resides at each cell's center of mass.

We define the volume-weighted norm of a variable e:

$$\|\Lambda e\|_p = \left(\sum_{i \in \Omega} |e_i|^p \Lambda_i h^d \middle/ \sum_{i \in \Omega} \Lambda_i h^d \right) \tag{22}$$

where Ω is the computational domain. An ∞-norm, $\|e\|_\infty$, is the maximum over all the domain of the absolute value of the elements of e. The rate of convergence in a given norm, p, between two errors fields, e_1 and e_2, computed with two different background mesh spacings, $h_1 > h_2$, is

$$R_p = \log\left(\frac{\|e_1\|_p}{\|e_2\|_p}\right) \Big/ \log\left(\frac{h_1}{h_2}\right) \tag{23}$$

The convergence rate, $R_p = n$ indicates n^{th}-order accuracy, i.e. the leading term in the truncation error scales as $\mathcal{O}(h^n)$.

6.1.1 Problem 1: Dirichlet Embedded Boundary Conditions

We enforce inhomogeneous Dirichlet boundary conditions, as described in Section 2.3, by setting the value at the center of the reconstructed interface, $\vec{x}_{bc}$, equal to the exact solution

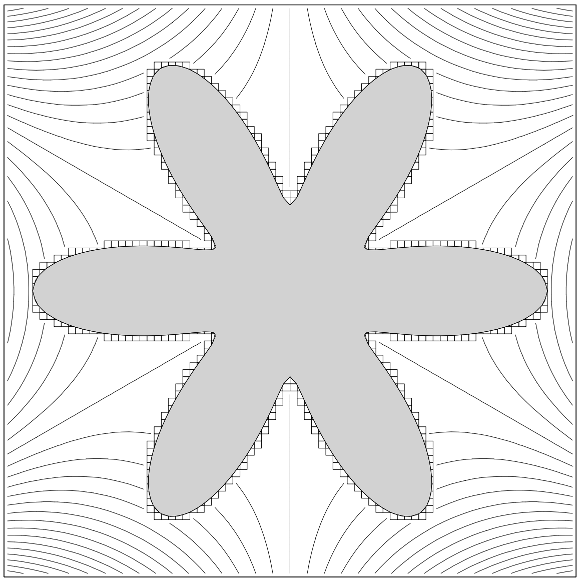


Figure 22: Contours of the exact solution for Problem 1, plotted over a grid with $h = 1/80$. The shaded region represents the embedded body, and is excluded from the computational domain. Contours are not extended into the cell fragments, which are drawn in around the embedded solid.

value, $\varphi^e(\vec{x}_{bc})$. This fixed value results in a non-trivial extensive Dirichlet boundary flux, $\vec{F}_{EB} \cdot \vec{A}_{EB}$, to be added to the conservative sum, Equation 6 on the cell fragments. Here, $\vec{A}_{EB} = A_{EB}\hat{n}_{EB}$, where $\hat{n}_{EB}$ is the unit normal on the embedded boundary evaluated at $\vec{x}_{bc}$, and A_{EB} is the magnitude of area of contact of the cell fragment with the solid, computed using the interface reconstruction scheme described in Appendix A. The flux, $\vec{F}_{EB}$, is computed according to the specification in Section 2.3.

In the first set of cases, the embedded boundary geometry is constructed using the natural method, as discussed in Section 5.1, and we discretize the domain on uniform mesh of N^2 cells, where $N = \{40, 80, 160, 320, 640, 1280\}$. For these cases, the finest grid covers the entire domain, i.e. $P(\Omega^{m+1}) = \Omega^m, \forall\, m : 0 \leq m < m_{hi}$. We initialize the state with the exact solution, and relax the system via a multigrid V-cycle using the level-transfer and smoother operations defined in Section 3.3. At the coarsest level, there is insufficient data in the domain to compute the full embedded boundary interpolants at 16 of the partial cell boundaries (generally, at the concave nodes of the geometry). For those cells, the planar interpolation functions are used, resulting in a scheme that is formally inconsistent. At the refinement level where $h = 1/80$, there are just four points where this occurs–at the concave nodes along the central vertical axis. For $h < 1/80$, the full interpolants could be computed for all partial cells in the domain. A contour plot of the solution for this test problem is shown in Figure 22. In the figure, we also draw in the cell fragments resulting from discretizing the domain on a grid with $h = 1/80$, and shade in the embedded body, which is excluded from the calculation.

Tables 1 and 2 show the convergence rates of the norm of the volume-weighted truncation, and the error, respectively with decreasing $h = 1/N$. The large initial rates are due to

N	$\|\Lambda\tau\|_\infty$	R_∞	$\|\Lambda\tau\|_1$	R_1	$\|\Lambda\tau\|_2$	R_2
0040	0.483204	-	0.0148148	-	0.15888	-
0080	0.0616952	3.0	0.00311469	2.2	0.0141678	3.5
0160	0.0330336	0.90	0.00078422	2.0	0.00491301	1.5
0320	0.0191487	0.79	0.000198956	2.0	0.00180314	1.4
0640	0.00907973	1.1	4.93392e-05	2.0	0.000628069	1.5
1280	0.0059005	0.62	1.23034e-05	2.0	0.000225618	1.5

Table 1: Convergence of the volume-weighted truncation error of the numerical Laplacian operator for the Dirichlet case, Problem 1, using the natural cell fragment construction process, as detailed in Section 5.1. The largest errors are consistently on the cell fragments. The ∞-norm converges roughly at first order in h, while the 1 and 2-norms converge at 2 and 1.5 respectively.

N	$\|\xi\|_\infty$	R_∞	$\|\xi\|_1$	R_1	$\|\xi\|_2$	R_2
40	3.47043e-05	-	8.63834e-06	-	1.35548e-05	-
80	5.15269e-06	2.8	1.27141e-06	2.8	1.83661e-06	2.9
160	7.38936e-07	2.8	2.3757e-07	2.4	3.42284e-07	2.4
320	1.32241e-07	2.5	5.1197e-08	2.1	7.67556e-08	2.2
640	3.26955e-08	2.0	1.19695e-08	2.1	1.85318e-08	2.1
1280	8.18485e-09	2.0	2.93154e-09	2.0	4.61567e-09	2.0

Table 2: Convergence of the error, $\xi = \varphi - \varphi^e$, of the computed solution to the Dirichlet case, Problem 1, using the natural cell fragment construction process, as detailed in Section 5.1. The largest error is on the cell fragments. The initial fast convergence was explained in [33]; the convergence rates approach two asymptotically, as expected.

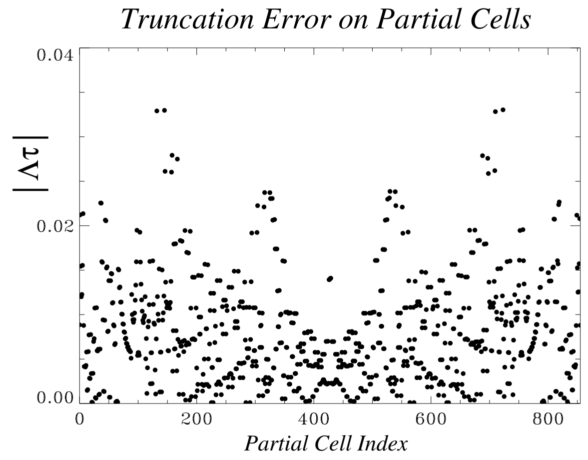


Figure 23: Scatter plot of the magnitude of the volume-weighted truncation error, $|\Lambda\tau|$, of the operator in Problem 1, over the cell fragments generated for the grid, $h = 1/160$.

the low-order boundary interpolants, and the erratic convergence rates for $\|\Lambda\tau\|_\infty$ will be explained shortly. The 1 and 2 norm convergence rates for the truncation are as expected for centered differences with boundary fluxes computed using parabolic interpolants. As demonstrated in [33], the initial rapid convergence of the error, ξ, due to errors in approximating the flux at embedded Dirichlet boundaries. These errors, which are large on coarse grids, generate contributions to ξ which converge to third order in h for all the norms. Eventually however, this component of ξ becomes small enough that it is dominated by the second-order truncation terms generated by the central differences in the bulk of the computational domain. Our results appear to confirm that argument.

The erratic convergence behavior of $\|\Lambda\tau\|_\infty$ can be understood with the aid of Figure 23, where we plot the magnitude of $\Lambda\tau$ as a function of cell fragment index number for Problem 1 setup, with $h = 1/160$. It is evident that the ∞-norm is determined (to a factor of two, or so) by a small number of cells in the domain. All the cells represented in the figure where $|\Lambda\tau| > 0.02$ share the feature that boundary curvature effects introduce $\mathcal{O}(h)$ errors into the numerical operator via the approximation of A_{EB}, i.e. the surface integral,

$$\oint_{\partial\Delta_{\mathbf{i}}} \hat{n} \cdot d\vec{S}$$

is incorrect to $\mathcal{O}(h)$, whereas for normal cell fragments, this error is only $\mathcal{O}(h^2)$. Such a situation arises only for triangular reconstructed cells with high aspect-ratio near regions on the boundary with significant curvature (see Figure 24). In this case, the reconstruction will position the boundary segment correctly to $\mathcal{O}(h^2)$, measured along $\hat{n}_{EB}$, which results in an $\mathcal{O}(h)$ error at the intersections with the grid line. As an alternative, the boundary area, $\vec{A}_{EB}$, may be defined so that the discrete area integral is exact, but then Λ computed for this cell would no longer consistent with these cell faces. The observation motivated us to develop the blunted approach to cell construction, described in Section 5.1, where both the cell apertures and volumes are constructed consistent with the piecewise linear representation. For this case, the boundary interface reconstruction procedure will give

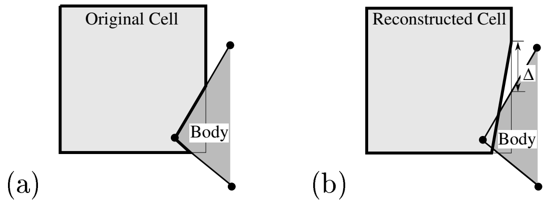


Figure 24: Cell reconstruction in 2D. The error, Δ, in the computed boundary area, may be $\mathcal{O}(h)$ for this special case. The figure exaggerates the curvature of the boundary to illustrate the issue.

N	$\|\Lambda\tau\|_\infty$	R_∞	$\|\Lambda\tau\|_1$	R_1	$\|\Lambda\tau\|_2$	R_2
40	0.484045	-	0.0150905	-	0.155971	-
80	0.065068	2.90	0.00313342	2.3	0.014329	3.4
160	0.035117	0.89	0.000787199	2.0	0.0049367	1.5
320	0.0198966	0.82	0.000199542	2.0	0.00179268	1.5
640	0.0104768	0.93	4.94945e-05	2.0	0.000618629	1.5
1280	0.00578773	0.86	1.23339e-05	2.0	0.000220398	1.5

Table 3: Convergence of the volume-weighted truncation error of the numerical Laplacian operator for the Dirichlet case, Problem 1, using the blunted cell fragment construction process, as detailed in Section 5.1. The ∞-norm convergence is slightly better behaved in this case, compared to the results generated from the natural cell generation method. Here again, the ∞-norm converges roughly at first order in h, while the 1 and 2-norms converge at 2 and 1.5 respectively.

the "exact" boundary point location. Now, the geometry-induced errors arise only from the piecewise-linear representation of the smooth boundary, and these errors are smoother and better behaved. This phenomenon was not observed in [33], where the cell fragments generated were automatically blunted (according to our terminology in Section 5.1).

We re-computed the geometry using the blunted version of the scheme, as detailed in Section 5.1. Tables 3 and 4 show the convergence behavior for Problem 1, using the blunted cell fragments. Clearly, $\|\Lambda\tau\|_\infty$ behaves as expected, and indeed how it was reported to behave in [33]. Note that the other norms remain effectively unchanged, as one might expect. As an aside, we note that a plot for this blunted case that corresponds to Figure 23 would show the same general features as before, i.e. only a few cells were within a factor of 2 of $\|\xi\|_\infty$, while the rest were at roughly half that level. However, in this case, since these errors decay smoothly as h decreases, the outliers do not adversely impact the overall error norm.

These results might indicate that the blunted scheme is superior to the natural scheme, were it not for the undue restrictions that the blunted scheme places on geometries. Aside from being unable to represent concave features on the scale of the mesh grid, the blunted method generates geometries that may depend strongly on the position of the underlying grid lines. Since we are developing these numerical schemes for arbitrary geometries, the latter is not a desirable feature. Also, it is worth noting that since the large errors in this scheme are due to an $\mathcal{O}(1)$ number of points, and since these cells generally have very small

N	$\|\xi\|_\infty$	R_∞	$\|\xi\|_1$	R_1	$\|\xi\|_2$	R_2
40	3.62205e-05	-	8.74556e-06	-	1.36764e-05	-
80	5.21425e-06	2.8	1.2735e-06	2.8	1.84214e-06	2.9
160	7.92214e-07	2.7	2.38048e-07	2.4	3.43014e-07	2.4
320	1.32427e-07	2.6	5.12732e-08	2.2	7.68536e-08	2.2
640	3.27236e-08	2.0	1.19778e-08	2.1	1.8544e-08	2.1
1280	8.18749e-09	2.0	2.93368e-09	2.0	4.61752e-09	2.0

Table 4: Convergence of the error, $\xi = \varphi - \varphi^e$, of the computed solution to the Dirichlet case, Problem 1, using the blunted cell fragment construction process, as detailed in Section 5.1. These results are quite similar to those presented for the natural cell fragment method

volume, they will have minimal impact on the solution over the rest of the domain–this was evident in the results presented above, since the truncation and solution errors converged at the expected rates in the 1 and 2-norm regardless of the convergence behavior of the ∞-norm.

6.1.2 Problem 2: Neumann Embedded Boundary Conditions

To test the discretization with Neumann boundary conditions, we set up a case identical to Problem 1, except that the inhomogeneous extensive flux, $\vec{F}_{EB} \cdot \hat{n}_{EB}$, added to the conservative flux sum is computed explicitly from the known solution. The components of $\vec{F}_{EB} = \nabla\varphi^e$, are

$$\nabla_x \varphi\,(x, y) = \frac{4x^4 - 3x^2y^2 - 3y^4}{r}$$
$$\nabla_y \varphi\,(x, y) = -\frac{-xy\,(5x^2 + 9y^2)}{r}$$

where $r^2 = x^2 + y^2$. The local normal was computed from the cell's edge fragment apertures, according to the procedures outlined in Appendix A. The natural cell construction procedures were used for this case. The convergence results are presented in Tables 5 and 6. Note that convergence behavior in these cases is much more uniform. The Neumann case appears to be somewhat less sensitive to details of the boundary treatment, as expected. Since there is no contribution to conservative flux sum from terms along the boundary interface, the scheme is insensitive to the details of the boundary surface reconstruction procedure.

6.2 Multigrid Assessment

In this section, we evaluate the effectiveness of our simple multigrid scheme for solving the Poisson equation on a variety of two-dimensional geometries. First, we assess the performance of our schemes for the case that $\ell_{hi} = 0$, and $\Omega^m = P(\Omega^{m+1})$, i. e. the finest grid covers the entire domain completely. We use the simple V-cycle described in this paper, with low-order level transfer functions and a point-relaxation smoother. In all cases, the coarsest level in the multigrid V-cycle is $h = 1/2$, and the "exact" solve at the bottom of the

N	$\|\Lambda\tau\|_\infty$	R_∞	$\|\Lambda\tau\|_1$	R_1	$\|\Lambda\tau\|_2$	R_2
40	0.0498369	-	0.00691271	-	0.019298	-
80	0.0249605	1.0	0.00178174	2.0	0.00678532	1.5
160	0.0124902	1.0	0.000451207	2.0	0.00242888	1.5
320	0.00624756	1.0	0.000114123	1.9	0.000871247	1.5
640	0.00312439	1.0	2.87537e-05	2.0	0.000308245	1.5
1280	0.00156234	1.0	7.23863e-06	2.0	0.000110257	1.5

Table 5: Convergence of the volume-weighted truncation error of the numerical Laplacian operator for the Neumann case, Problem 2, using the natural cell fragment construction process, as detailed in Section 5.1. Here again, the ∞-norm converges at first order in h, while the 1 and 2-norms converge at 2 and 1.5 respectively.

N	$\|\xi\|_\infty$	R_∞	$\|\xi\|_1$	R_1	$\|\xi\|_2$	R_2
40	6.12207e-05	-	1.77911e-05	-	3.01133e-05	-
80	1.72152e-05	1.8	4.8787e-06	1.9	8.34709e-06	1.9
160	4.57722e-06	1.9	1.29383e-06	1.9	2.21562e-06	1.9
320	1.18702e-06	1.9	3.35947e-07	1.9	5.73883e-07	1.9
640	3.02077e-07	2.0	8.53856e-08	2.0	1.45699e-07	2.0
1280	7.61676e-08	2.0	2.16465e-08	2.0	3.68528e-08	2.0

Table 6: Convergence of the error, $\xi = \varphi - \varphi^e$, of the computed solution to the Neumann case, Problem 2, using the natural cell fragment construction process, as detailed in Section 5.1. We clearly observe second order in all norm measures.

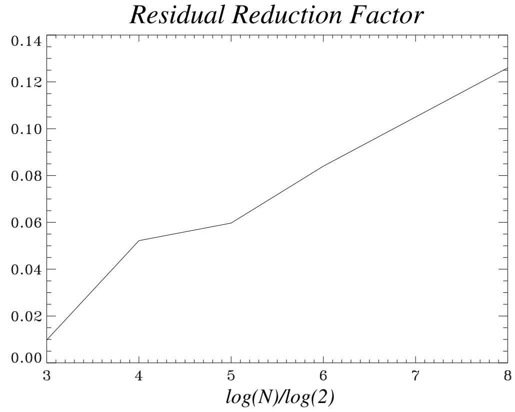


Figure 25: Plot of the residual reduction factor, f, as a function of system size, N, for Problem 3. The reduction factor scales linearly with $\log N$, so that the computer work to solve this linear system scales as $\mathcal{O}(N \log N)$.

V-cycle (on the 2×2 system) consists of 8 passes of the smoother operation. The measure of performance is the averaged residual reduction factor,

$$f = \left(\frac{\|\Lambda R_0\|_\infty}{\|\Lambda R_N\|_\infty} \right)^{\frac{1}{N}} \tag{24}$$

where the average is taken over the total number, N, of complete multigrid iterations taken during that solve. As in Section 3.1, $R = \rho - L(\varphi)$, and the subscript indicates iteration number, with 0 representing the residual computed with the initial guess. For all cases, the initial guess was a flat profile ($\varphi = 0$), and the system was relaxed until $\|\Lambda R\|_\infty$ was reduced by 10 orders of magnitude.

6.2.1 Problem 3: Solver Scaling with System Size

Problem 3 is designed to illustrate how our simple multigrid scheme scales with system size. The problem setup is identical to that of Problem 1, except that we build a single geometry, based on $h = 1/256$, and construct a series of sub-problems, at decreasing levels of refinement. We do this simply by starting the multigrid V-cycle at successively higher (coarser) levels. Figure 25 plots the residual reduction factor, f as a function $\log N$, where N is the number of cells on a side ($= 1h$). The reduction factor increases linearly with $\log N$, which verifies that we are achieving the expected performance of classic multigrid schemes, where the work scales as $\mathcal{O}(N \log N)$.

6.2.2 Problems 4 and 5

Problems 4 and 5 are designed to test the scheme's ability to handle a wide variety of geometrical shapes embedded in the grid. Statistics for the six geometries we tried appear in Table 7. In all cases, the background uniform mesh is 256×256 over the region in two-dimensional real space, $[0, 0] \times [1, 1]$. The bodies were described as a set of node lists, and the natural cell fragment construction procedure was used. The cases are:

Case	Desc.	$N_{\mathcal{B}}$	N_V^{256}	N	$t^{256}\,(s)$	N_V^{1024}	$t_{1024}\,(s)$	f_N	f_D
A	Line	0	192	65336	0.17	768	0.43	0.141	0.146
B	Boxes	8192	512	57344	0.35	2048	1.4	0.103	0.0767
C	Ellipses	10240	912	55296	0.9	3792	5.0	0.0407	0.0557
D	Ellipse	42764	700	22772	0.63	2812	7.5	0.0607	0.0760
E	Naca	2311	536	63225	0.5	2144	1.8	0.118	0.0902
F	Arc	0	764	65918	0.7	3068	3.0	0.186	0.135

Table 7: The six geometries used for Problems 4 and 5. Here, $N_{\mathcal{B}}$ is the number of solid cells , N_V^{256} is the number of cell fragments, N is the total number of uncovered cells (including full and partial cells), t^{256} is the CPU time (in seconds) to generate the geometry on the 256^2 domain, N_V^{1024} is the number of cell fragments in the same geometry generated for a 1024^2 mesh, and t_{1024} is the CPU time it took to generate the larger geometry. f_N and f_D are the residual reduction factors for the Neumann and Dirichlet problems, Problems 4 and 5, respectively.

A. An infinitely thin line boundary from $\left(\frac{5}{16}, \frac{5}{16}\right)$ to $\left(\frac{5}{16}, \frac{11}{16}\right)$.

B. Two boxes, measuring $\frac{1}{4}$ on a side, and placed at $\left(\frac{1}{2}, \frac{1}{2} \pm \frac{1}{4}\right)$.

C. Four ellipses, centered at $\left(\frac{1}{2} \pm \frac{1}{4}, \frac{1}{2} \pm \frac{1}{4}\right)$ measuring $\frac{5}{16} \times \frac{5}{32}$.

D. One ellipse, centered at $\left(\frac{1}{2}, \frac{1}{2}\right)$, measuring $\frac{3}{4} \times \frac{5}{8}$. For this case, the domain is *inside* the elliptical surface (so that there are no rectangular boundaries for this example),

E. Two NACA 0012 airfoils, uniformly scaled to have length 0.468, and placed so that the leading edges are at $(.336, .625)$ and $(.195, .375)$.

F. An arc, sweeping out $\theta = [\frac{\pi}{4}, -\frac{\pi}{4}]$, with center at $\left(\frac{3}{4}, 0\right)$ and radius $= \frac{1}{4}$.

For Problem 4, we computed solutions to the Poisson equation on the six geometries, imposing a homogeneous Neumann boundary condition at the embedded boundaries. A homogeneous Dirichlet condition was imposed all along the rectangular boundary (if one exists in the problem). To make the solution non-trivial and non-singular, we placed equal, but opposing Gaussian source terms,

$$\rho = \sum_i C_1^i \exp\left(C_2^i \left|\vec{x} - \vec{x}_i\right|^2\right) \tag{25}$$

where $\vec{x}_i = \left(\pm\frac{1}{2}, \frac{1}{2}\right)$, $C_1^i = \pm 1$ and $C_2^i = 0.01$ (actually we omitted the left source in Case A to obtain a solution which more clearly demonstrates the abilities of the code). Contour plots of the solutions for all six cases are presented in Figure 26. Here, we plot 31 equally spaced levels between the extreme values of the solution, and shade in the embedded bodies that have been excluded for each case. The contours clearly intersect normal to the embedded surface, and are tangent to the rectangular boundary, as expected. The multigrid residual reduction factors for this case appear in Table 7, column 9.

For Problem 5, we enforce Dirichlet conditions on all boundaries. The embedded boundaries were set to zero, while the rectangular boundaries where set to unity (if they exist in

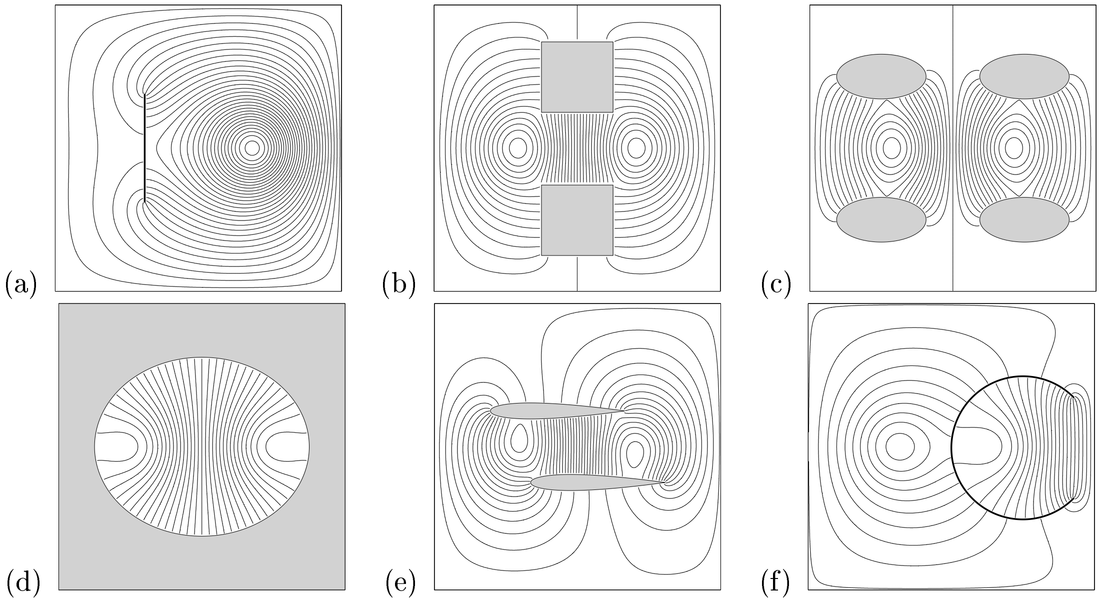


Figure 26: Contour plots for Problem 4. Homogeneous Neumann conditions are imposed on the embedded body, while homogeneous Dirichlet conditions are applied to the rectangular boundaries (except in (d)).

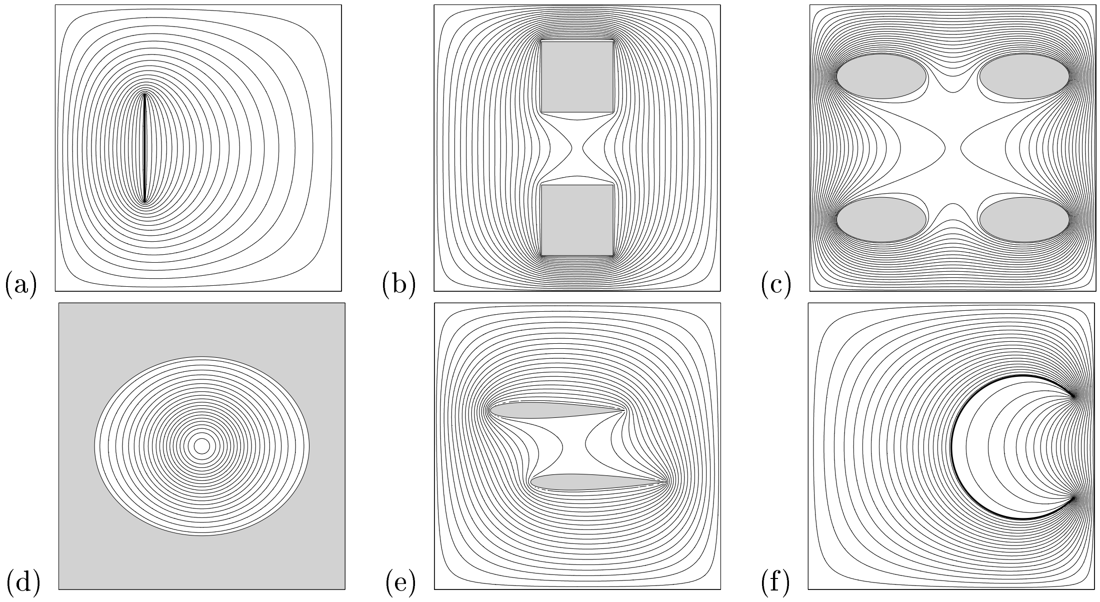


Figure 27: Contour plots for Problem 5. Homogeneous Dirichlet conditions are imposed on all boundaries of the domain, including rectangular and embedded surfaces.

the problem). In Case D, this would result in a trivial solution; we added a single source of the form of Equation 25 at the center of the domain, with where $\vec{x}_1 = \left(\frac{1}{2}, \frac{1}{2}\right)$, $C_1^1 = 1$ and $C_2^1 = 0.01$. Contour plots of the solutions for all six cases are presented in Figure 27. Here, we plot 21 equally spaced levels between 0 and 1, inclusive. The contours are clearly tangent to all boundaries in the problems, and show the correct general characteristics expected of the Poisson solution. The multigrid residual reduction factors for this case appear in Table 7, column 10.

6.2.3 Problem 6: Adaptive Multi-level Solve

Problem 6 demonstrates the AMR component of our solver. For this case, we chose the geometry labeled "F" in Table 7, and run the system setup for Problem 3 (i.e. the Poisson equation, with two opposing sources). We apply homogeneous Neumann conditions to the embedded boundaries, and along the left and right sides of the domain. We apply homogeneous Dirichlet conditions to the upper and lower boundaries. The final solution presented has four AMR levels, with a uniform factor of two separating each. The finest grid has $h = 1/512$, while the coarsest uncovered level has $h = 1/64$.

First, we solve our problem over the entire domain with $h = 1/64$ (this will involve 7 multigrid levels, with $h = 1/2^n, n = [1, 6]$). Richardson extrapolation is used to estimate the local truncation error, τ, as described in [38]. All rectangular cells with $\tau > .1h^2$ are "tagged" for refinement, according to the procedure detailed in [38]. We also tag every mesh cell containing at least one partial cell (in this way, we can ensure that the embedded boundary is always gridded to the finest level). A set of rectangular grid patches are generated for the level with $h = 1/128$. The fine grid solution is initialized by by interpolating the coarse grid values using our piecewise-constant level transfer scheme, and the system is relaxed via the multi-level multigrid scheme given in Algorithm 5. After the solve, the error tagging procedure is applied again to adjust the grids at $h = 1/128$, if necessary. The solve at this level continues until the grid layout remains constant. The next AMR level is then added using a similar process, and the three-level scheme is iterated to convergence in the same way. We terminate execution after four AMR levels have been converged, both in terms of grid placement and in terms of reducing the ∞-norm of the residual at each level at least eight orders of magnitude from that of the initial guess.

Figure 28 shows the converged, adapted solution. The boxes overlaid on the contours indicate the extent of block-structured logically rectangular grids at each level (for clarity, we've shaded them according to level). Due to limitations in our graphics, contours were not drawn in the partial cells–this is most noticeable near the body in the first solution plot. In the final arrangement, levels 3-0 cover $6.96, 24.3.46.8$ and 100 percent of the computational domain, respectively. The average residual reduction factor for the entire calculation was approximately 0.08. Note that the solution is resolved well enough that grid refinement is not triggered near the location of the sources. The minor profile adjustments with grid refinement appear to result from the improved resolution of the curved boundary.

We note this example was the largest of the linear solves presented in this paper, consuming approximately 10 minutes of CPU time on a DEC Alpha 300 MHz machine. Although such performance is unacceptable for for typical high-performance computing applications, the encouraging convergence performance warrants another implementation pass to stream-

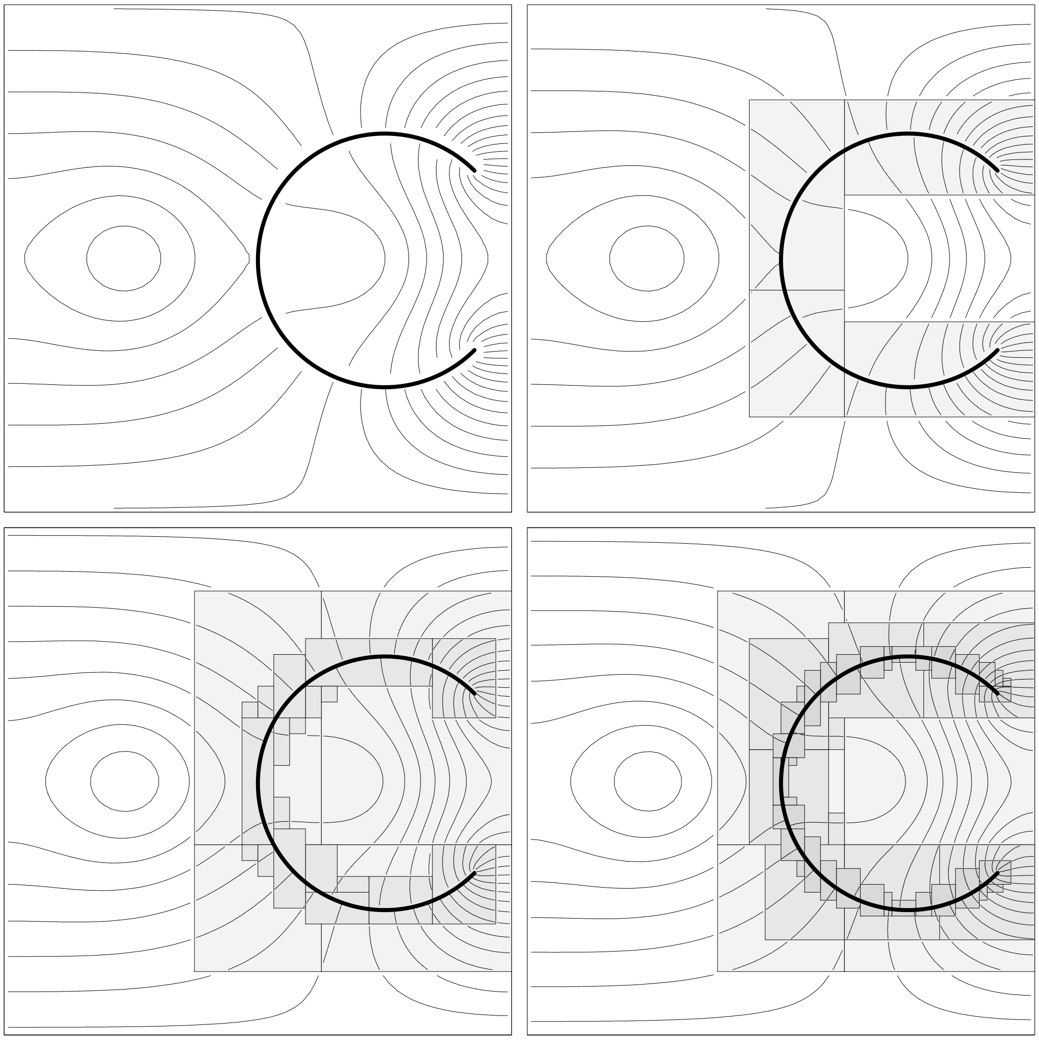


Figure 28: Adaptive solution to the Poisson equation on the "Arc" geometry, an infinitely thin embedded surface. Left-to-right, top-to-bottom, the figures depict the converged solution with one, two, three and four AMR levels, respectively. The coarsest solution is on a 64×64 grid, and each AMR level is refined a constant factor of two from the one next coarser.

line data access and minimize unnecessary calculation.

7 Conclusions

In this paper, we have presented a graph-based algorithm for representing irregular bodies embedded in a block-structured, logically rectangular Cartesian grid. We detailed a recursive geometrical coarsening strategy valid for arbitrarily complicated domain shapes. The strategy carries enough geometrical information along to allow finite-volume type conservation-law discretizations to be constructed on every coarse level generated. It appropriately handles "thin-bodies" and "trailing-edges" at every level, and extends directly to three dimensions.

Based on our data representation and coarsening strategy, we constructed a simple multigrid scheme for solving the Poisson equation in the presence of arbitrarily complex geometries. We studied the behavior of our scheme, both in terms of convergence rates to the continuum solution of Poisson's equation, and in terms of the residual reduction rates. By testing over a wide range of geometries, we found that the complexity and position of the embedded shapes seems to have some impact on our scheme's residual reduction properties, but that the scheme was nonetheless generally quite robust—remarkable since we implemented only the simplest of possible strategies for the various components of multigrid (such as level transfer and smoothing operations).

The encouraging results presented in this paper indicate that our scheme is suitable for extension to other conservation-law PDE systems. Our EB formalism extends naturally to the high-resolution Godunov scheme presented in [17], and to the approximate projection scheme presented in [27]). The adaptive solver can also be used as a starting point for extending the variable-density adaptive projection schemes in [37, 38].

A Piecewise Linear Boundary Reconstruction

At the most refined grid level, the embedded boundary is represented as a C_0 piecewise linear interface, specified by the apertures of the edges in connected paths of the irregular geometry graph. The location of this interface within the cells is required, for example, when computing the fluxes induced by Dirichlet conditions along the embedded boundary. The boundary surface can be reconstructed with *volume-of-fluid*-type methods. We compute the surface normal, $\vec{n} = \{n_1, \ldots, n_d\}$, for irregular cell v using the partial cell apertures:

$$n_k = \sum_{e:e_-=v} \mathcal{A}(e) - \sum_{e:e_+=v} \mathcal{A}(e) \tag{26}$$

for all edges, e, in the k-direction. Now, the surface normal, $\vec{n}$, and the cell volume, $\Lambda(v)$, specify a unique location for a *linear* boundary intersection surface. In two dimensions, for example (see Figure 29), take the case that $|n_2| > |n_1| > 0$. Define a slope, $m = \frac{n_1}{n_2}$, and $v = \Lambda(v)$. The shape of the cell can identified:

$$v \text{ is a} \begin{cases} \text{quadrilateral} & \text{if } v \geq b \text{ and } v \leq 1-b \\ \text{triangle} & \text{if } v \leq b \text{ and } v \leq 1-b \\ \text{pentagon} & \text{if } v \geq b \text{ and } v \geq 1-b \end{cases}$$

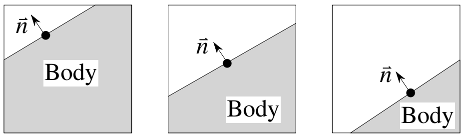


Figure 29: Linear reconstruction of the embedded boundary surface, based on the surface normal, $\vec{n}$, and cell volume. In two-dimensions, the fluid cell may be triangular, trapezoidal or pentagonal.

where $b = |m|/2$. In the case of the triangle, one node is on the unit square, the others are at $\left(0, 1-\sqrt{4bv}\right)$ and $\left(\sqrt{v/b}, 1\right)$. In the case of a quadrilateral, two nodes are on the unit square, and the others are at $(0, 1-v \pm b)$. Finally, in the case of the pentagon, three of the nodes are on the unit square, while the other two are at $\left(1-\sqrt{(1-v)/b}, 0\right)$ and $\left(1, \sqrt{4b(1-v)}\right)$. These formula are then rotated based on the signs of n_1 and n_2, in order to keep a well-behaved slope, m. Similar, but slightly more complex formula can be used to find the plane intersecting the boundary in three dimensions.

We use these formulas also to define an approximate boundary location for geometries coarsened using the techniques described in Section 3.2. However, there is no guarantee that the position will accurately reflect the sub-grid scale boundary shape. And since we are modeling arbitrary boundary shape, higher order reconstruction methods do not necessarily represent an improvement in location accuracy. For example, on a very coarse grid with $d = 2$, one may compute $n_k = 0, \forall k \in \{1, d\}$ and $\Lambda(v) < 1$, for some v in the domain. This implies that the embedded structure lays completely within the coarse cell; this case cannot be represented by a linear boundary segment. We handle this case in the computations by merely setting the slope, $m = 0$, and continuing on to the next cell. Such ill-defined cases arise typically at the coarsest levels generated automatically for a multigrid solver grid hierarchy, and usually represent a refinement level constructed solely as a temporary aid for the linear solution. If such a procedure ends up degrading the performance of the solver, we remove that level from the multigrid hierarchy so that the coarsest problem is fine enough to prevent these ambiguous cases.

References

[1] T.A. Reyhner, *Cartesian mesh solution for axisymmetric transonic potential flow around inlets. AIAA J.*, **15**(5) p. 624–631 (1977).

[2] J.W. Purvis and J.E. Burkhalter, *Prediction of critical mach number for store configurations. AIAA J.*, **17**(11) p. 1170–1177 (1979).

[3] B. Wedan and J. South, *A method for solving the transonic full-potential equations for general configurations.* in “Proceedings, AIAA 6th Computational Fluid Dynamics Conference”, Danvers, MA (1993). AIAA Paper 83-1889.

[4] D.P. Young, R.G. Melvin, M.B. Bieterman, F.T. Johnson, S.S. Samant, and J.E. Bussoletti, *A locally refined rectangular grid finite element method: Application to com-*

putational fluid dynamics and computational physics. J. Comput. Phys., **92** p. 1–66 (1991).

[5] D.K. Clarke, M.D. Salas, and H.A. Hassan, *Euler calculations for multielement airfoils using Cartesian grids. AIAA J.*, **24**(3) p. 353–358 (1986).

[6] R.L. Gaffney, H.A. Hassan, and M.D. Salas, *Euler calculations for wings using Cartesian grids.* in "Proceedings of the AIAA 25th Aerospaces Meeting", Reno, NV (1987). AIAA Paper 87-0356.

[7] B. Epstein, A.L. Luntz, and A. Nachson, *Cartesian Euler methods for arbitrary aircraft configurations. AIAA J.*, **30**(3) p. 679–687 (1992).

[8] K. Morinishi, *A finite difference solution of the Euler equations on non-body fitted grids. Comp. Fluids*, **21**(3) p. 331–344 (1992).

[9] Y-L. Chiang, B. van Leer, and K.G. Powell, *Simulation of unsteady inviscid flow on an adaptively refined Cartesian grid.* AIAA Paper 92-0443-CP (1992).

[10] W.J. Coirier, *An Adaptively Refined, Cartesian, Cell-Based Scheme for the Euler and Navier-Stokes Equations.* PhD thesis, University of Michigan (1994). Also appeared as NASA TM-106754.

[11] D.L. DeZeeuw, *A Quadtree-Based Adaptively Refined Cartesian-Grid Algorithm for Solution of the Euler Equations.* PhD thesis, University of Michigan (1993).

[12] D.L. DeZeeuw and K.G. Powell, *An adaptively refined Cartesian mesh solver for the Euler equations. J. Comput. Phys.*, **104** p. 56–68 (1993).

[13] C.F. Gooch, *Solution of the Navier-Stokes Equations on Locally-Refined Cartesian meshes using State-Vector Splitting.* PhD thesis, Stanford University (1993).

[14] J.E. Melton, *Automated Three-Dimensional Cartesian Grid Generation and Euler Flow Solutions for Arbitrary Geometries.* PhD thesis, University of California, Davis (1996).

[15] W.F. Noh, *CEL: A time-dependent two-space-dimensional, coupled Eulerian-Lagrange code.* in "Fundamental Methods of Hydrodynamics, Methods of Computational Physics", Vol. 3. Academic Press, New York/London (1964).

[16] J.J. Quirk, *An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies. Comp. Fluids*, **23**(1) p. 125–142 (1994).

[17] R.B. Pember, J.B. Bell, P. Colella, W.Y. Crutchfield, and M.L. Welcome, *An adaptive Cartesian grid method for unsteady compressible flow in irregular regions. J. Comput. Phys.*, **120** p. 278–304 (1995).

[18] G. Yang, D.M. Causon, and D.M. Ingram, *Calculation if 3-D compressible flows around moving bodies.* in "Twenty-first International Symposium on Shock Waves", Great Keppel Island, Australia (1997). Paper 1780.

[19] R.J. LeVeque, *A large time step generalization of Godunov's method for systems of conservation laws. SIAM J. Num. Anal.*, **22** p. 1051–1073 (1985).

[20] R.J. LeVeque, *High resolution finite volume methods on arbitrary grids via wave propagation. J. Comput. Phys.*, **78**(1) p. 36–63 (1988).

[21] M.J. Berger and R.J. LeVeque, *Stable boundary conditions for Cartesian grid calculations. Comp. Sys. Eng.*, **1** p. 305–311 (1990).

[22] M.J. Berger and R.J. LeVeque, *An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries.* in "Proceedings, AIAA 9th Computational Fluid Dynamics Conference", pp. 1–7, Buffalo, New York (1989). Paper 89-1930-CP.

[23] C.S. Peskin, *Flow patterns around heart valves. J. Comput. Phys.*, **10** p. 252–271 (1972).

[24] R.J. LeVeque, *High-resolution conservative algorithms for advection in incompressible flow. SIAM J. Num. Anal.*, **33** p. 627–665 (1996).

[25] M.F. Tome and S. McKee, *GENSMAC: a computational marker and cell method for free surface flows in general domains. J. Comput. Phys.*, **110** p. 171–186 (1994).

[26] E.Y. Tau, *A second-order projection method for the incompressible Navier-Stokes equations in arbitrary domains. J. Comput. Phys.*, **115** p. 147–152 (1994).

[27] A. Almgren, J.B. Bell, P. Colella, and T. Marthaler, *A Cartesian grid method for incompressible Euler equations in complex geomtries. SIAM J. Sci. Comput.*, **18**(5) p. 1289–1309 (1997).

[28] A. Chorin, *Numerical solution of the Navier-Stokes equations. Math. Comp.*, **22** p. 745–762 (1969).

[29] R.J. LeVeque and Z. Li, *The immersed interface method for elliptic equations with discontinuous coefficients and singular sources. SIAM J. Num. Anal.*, **31** p. 1019–1044 (1994).

[30] Z. Yang, *A Cartesian Grid Method for Elliptic Boundary Value Problems in Irregular Regions.* PhD thesis, University of Washington (1996).

[31] L. Adams, *A multigrid algorithm for immersed interface problems,* in "Seventh Copper Mountain Conference on Multigrid Methods". NASA Conference Publication 3339, NASA, p. 1–14 (1996).

[32] D.W. Hewitt, *The embedded curved boundary method for othogonal simulation meshes. J. Comput. Phys.*, **138** p. 585 (1997).

[33] H. Johansen and P. Colella, *A Cartesian grid embedded boundary method for Poisson's equation on irregular domains.* accepted for publication *J. Comput. Phys.* Also appears as LBNL-39908, Lawrence Berkeley National Laboratory, January, 1997.

[34] M.J. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations. J. Comput. Phys.*, **53** p. 484–512 (1984).

[35] W.L. Briggs. *A Multigrid Tutorial.* SIAM, Philadelphia, PA (1987).

[36] M.J. Berger and I Rigoutsos, *An algorithm for point clustering and grid generation. IEEE Trans. Sys. Man Cybernet*, **21** p. 1278–1286 (1991).

[37] A. Almgren, J.B. Bell, P. Colella, L.H. Howell, and M.L. Welcome, *A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. J. Comput. Phys.*, **141** (1998). Also appears as LBNL-39075, Lawrence Berkeley National Laboratory, July, 1996.

[38] D. Martin, *An Adaptive Cell-centered Projection Method for the Incompressible Euler Equations.* PhD thesis, UC Berkeley (1998).