Embedded Boundary AMR Elliptic Algorithm and Implementation

P. Colella
D. T. Graves
T. J. Ligocki
B. Van Straalen

Applied Numerical Algorithms Group
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA

May 29, 2003

Contents

Chapter 1

Introduction

1.1 Introduction

This document is to briefly explain the workings of the EBAMRElliptic implementation. First, the algorithm is explained. Here we extend the AMR multigrid algorithm of Martin and Cartwright [6] to embedded boundaries using the results of Johansen and Colella [4], [5]. Second, we document the implementation of the EBAMRElliptic software in detail, trying wherever possible to draw the connections from the software to the algorithm specification. We shall describe the algorithm in both two and three dimensions.

We are going to present two examples. For both, we are given a bounded domain Ω and a charge density distribution ρ which exists over all of Ω and we are given a boundary condition for the solution is given on $\partial\Omega$. We discretize the domain with a block-structured, adaptive mesh.

In the first example, we solve Poisson's equation

$$\nabla \cdot (\nabla \phi) = \rho \tag{1.1}$$

for ϕ. In the second example, we present

Chapter 2

Poisson's Equation

2.1 Poisson's equation

This section describes the method for solving the elliptic partial differential equation

$$L(\phi(\vec{x})) = \rho(\vec{x}) \tag{2.1}$$

on a Cartesian grid embedded boundary mesh, for the special case of Poisson's equation, in which

$$L(\phi(\vec{x})) = \nabla^2 \phi(\vec{x}) \tag{2.2}$$

is the Laplacian. This algorithm is largely an extension of that developed by Johansen and Colella [4] combined with the AMR multigrid algorithm of Martin and Cartwright [6].

2.1.1 Operator Discretization

2.1.1.1 Notation

To suppress the use (i,j,k) notation, we define: $v^+(f)$ to be the VoF on the high side of face f; $v^-(f)$ to be the VoF on the low side of face f; $f_d^+(v)$ to be the set of faces on the high side of VoF v; $f_d^-(v)$ to be the set of faces on the low side of VoF v, where $d \in \{x,y,z\}$ is a coordinate direction (the number of directions is $\mathbf{D}$). Also, we compose these operators to obtain the set of VoFs directly connected to a given VoF: $v_d^+(v) = v^+(f_d^+(v))$ and $v_d^-(v) = v^-(f_d^-(v))$.

Barred variables, such as $\bar{x}_v$ or $\bar{x}_f$, are distances from the center of the grid cell containing v or of the grid face containing f, respectively, that have been normalized by the grid spacing h. Typically, $-\frac{1}{2} \leq \bar{(\cdot)} \leq \frac{1}{2}$.

2.1.1.2 Interior Method

The Laplacian of ϕ is defined in three stages: compute the grid-centered gradient of ϕ, recenter the gradient, and compute the divergence of recentered gradient. The face-centered gradient of ϕ is defined as

$$\widetilde{g}_f^d = \frac{1}{h}\left(\phi_{v^+(f)} - \phi_{v^-(f)}\right) \tag{2.3}$$

The gradients at the irregular face centroids g_f^d are computed by interpolation using a modification of the Johansen-Colella method. Interpolation is done in the $\mathbf{D}-1$ dimensional, linear subspace which contains the irregular face. In 2D, this is a line and, in 3D, this is a plane. If possible, multilinear interpolation is done using the face-centered gradients whose locations bound the centroid of the irregular face. Multilinear interpolation is possible if all the face-centered gradients needed can be used (see below for the definition of "can be used" in this context). If multilinear interpolation is not possible then the $\widetilde{g}_f^d$ is used at the irregular face centroid, i.e., piecewise constant interpolation.

By the divergence theorem, the integral of the Laplacian of ϕ over a VoF is equal to the integral around the boundary of the VoF of the gradient of ϕ. Discretizing the integral with the midpoint rule yields the approximation

$$L_v(\phi) = \frac{1}{\kappa_v h}\left(\sum_{f \in f_d^+(v)} \alpha_f g_f^d - \sum_{f \in f_d^-(v)} \alpha_f g_f^d \; - \; \alpha_v^{EB}\left(\vec{g}_v^{EB} \cdot \widehat{n}_v^{EB}\right)\right) \tag{2.4}$$

where κ_v is the volume fraction of a VoF v, α_f is the area fraction of face f, and α_v^{EB} is the area fraction of the embedded boundary of the VoF. The superscript EB, in general, refers to quantities associated with the segment of the embedded boundary within a VoF. The calculation of $(\vec{g}_v^B \cdot \widehat{n}_v^B)$, the normal gradient of ϕ at the boundary, is described in section 2.1.1.3. In regions of the grid where all VoFs and faces involved are regular, no recentering of the gradient is required and there is no contribution from the embedded boundary. In this case equation 2.3 gives the gradient at the VoF face and this method reduces to the familiar star-shaped direction-split stencil for the Laplacian:

$$L_v(\phi) = \frac{1}{h^2}\left(\sum_{d=0}^{\mathbf{D}-1} \phi_{v_d^+(v)} - 2\phi_v + \phi_{v_d^-(v)}\right). \tag{2.5}$$

See figure 2.1 for a graphical version of the stencil in two dimensions.

Now we define when a face-centered gradient "can be used" in the context of computing the gradient at the centroid of an irregular face. For each direction $d' \neq d$, we define two sets of VoFs,

$$\begin{aligned} v_{d'}^- &= v^\pm(f_{d'}^\pm((v^-(f))) \\ v_{d'}^+ &= v^\pm(f_{d'}^\pm((v^+(f))) \end{aligned} \tag{2.6}$$

where the choice of sign is the sign of $\bar{x}_f^{d'}$, the normalized centroid of f in the d' direction. Basically, we take the VoF on each side of f (in the d direction), find all faces connected

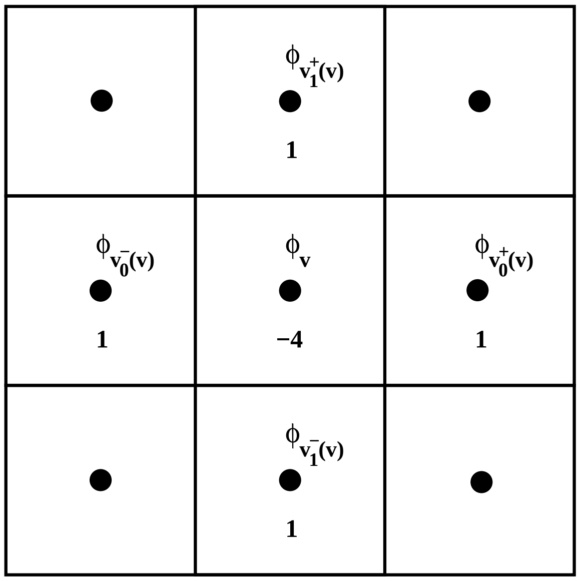

Figure 2.1: Illustration of the 5-point Laplacian stencil in two dimensions.

to that VoF in the low or high, $\pm$, d' direction, and then collect all the VoFs connected to the other side of these faces.

Now, construct the set of faces that are shared by a VoF in $v_{d'}^-$ and a VoF in $v_{d'}^+$. If there is one such face, it is $f'(d')$. If there are no faces or more than one face then $g_f^d = \widetilde{g}_f^d$, i.e., drop order.

2.1.1.3 Boundary Conditions

There are two distinct type of boundaries: faces which lie on the boundary of the solution domain, and embedded boundary segments which are contained within a VoF. See fig-

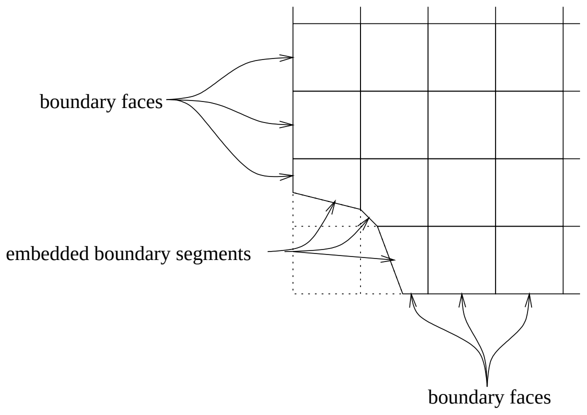

Figure 2.2: Boundary faces and embedded boundary segments.

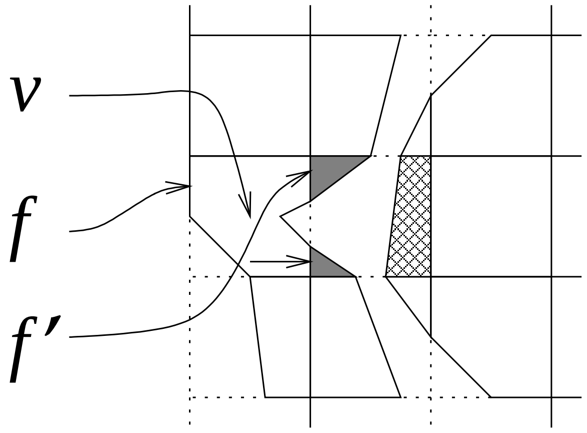

Figure 2.3: VoFs and faces for Dirichlet boundary condition at a boundary face.

ure 2.2. Discretization of homogeneous Dirichlet and Neumann boundary conditions are described for each type of boundary face. Homogeneous Neumann boundary conditions are defined by setting the appropriate gradient to zero. Homogeneous Dirichlet boundary conditions are more involved.

2.1.1.4 Homogeneous Dirichlet Boundary Condition at Faces

For a boundary face f normal to d, the normal gradient depends on whether the solution domain is on the high side $(+)$ of f or on the low side $(-)$ of f. The gradient is

$$\widetilde{g}_f^d = \pm\frac{1}{h}\left(3\phi_v - \phi_1/3\right) \tag{2.7}$$

where

$$v = v^{\pm}(f) \tag{2.8}$$

is the first VoF in the solution domain, and

$$\phi_1 = \frac{\displaystyle\sum_{f' \in f_d^{\pm}(v)} \ell_{f'} \phi_{v^{\pm}(f')}}{\displaystyle\sum_{f' \in f_d^{\pm}(v)} \ell_{f'}} \tag{2.9}$$

is the face-area-averaged value of the solution in the set of VoFs in the second cell inward in the solution domain that are directly connected to the VoF v. An example is shown in figure 2.3. In the figure, the solution domain is on the high side of the face f. The set of VoFs $v^+(f')$ is shaded gray. Note that the crosshatched VoF is in the same cell as $v^+(f')$, but does not participate in the average.

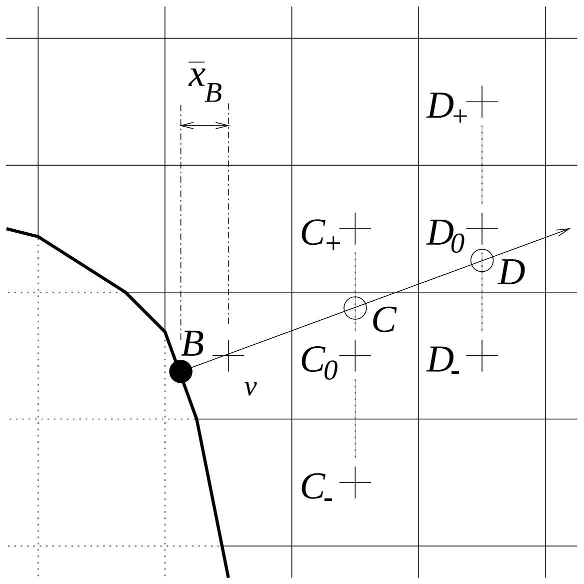

Figure 2.4: Dirichlet boundary condition at embedded boundary segment. Stencil for quadratic interpolation. Interpolate values from cell centers (+) to intersection points (○). Gradient at boundary segment centroid (●) is found by differencing values at the ○s.

2.1.1.5 Homogeneous Dirichlet Boundary Condition at Embedded Boundary Segments

For an embedded boundary segment, the gradient normal to the segment is found by casting a ray normal to the segment into the solution domain. See figure 2.4. The ray $\overrightarrow{BCD}$ is cast from the centroid of the embedded boundary face B in VoF v. Note that, in this example, $\widehat{n}^B_{v,x} \geq \widehat{n}^B_{v,y} \geq 0$. If this inequality does not hold, the problem is transformed into a different coordinate system in which it does hold by means of coordinate swaps and reflections. The direction that transforms to x is called the *major coordinate*. Unless otherwise specified, the rest of this discussion is in terms of the transformed coordinate system.

Planes are constructed normal to x through the centers of cells near v. We then find the intersection of the ray with the two planes that are closest to but do not intersect v. In the figure, the intersection points are C, the closer point, and D, the further point. We will first describe the method assuming all the cells necessary are regular except this one containing the embedded boundary. We locate the centers of the cells these intersections are within, C_0 and D_0, and the centers of the neighbor cells of each in the same intersection plane, C_+ and C_-, and D_+ and D_-, respectively (in three dimensions, there are eight neighbors each, C_{++}, C_{+0}, C_{+-}, *etc.*). Values at C and D are found by quadratic interpolation. In two dimensions,

$$\phi_C = N_+(\bar{y}_C)\phi_{C_+} + N_0(\bar{y}_C)\phi_{C_0} + N_-(\bar{y}_C)\phi_{C_-} \tag{2.10}$$

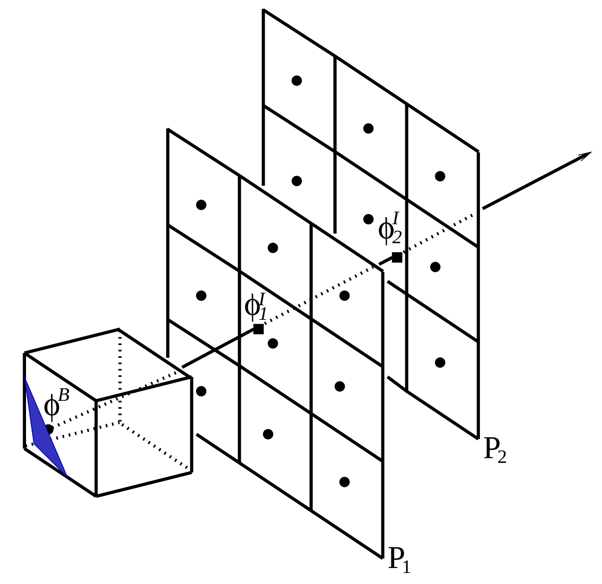

Figure 2.5: Quadratic approximation of C in three dimensions. We interpolate in each plane then interpolate along ray.

and similarly for ϕ_D, where

$$\bar{y}_C h = y_C - y_{C_0} \tag{2.11}$$

and the interpolation functions are

$$\begin{array}{rcl} N_+(\xi) & = & \frac{1}{2}\xi\,(\xi+1) \\ N_0(\xi) & = & 1-\xi^2 \\ N_-(\xi) & = & \frac{1}{2}\xi\,(\xi-1)\,. \end{array} \tag{2.12}$$

If the major coordinate were y, then y would be replaced by x in equation 2.10.

With the values ϕ_C and ϕ_D known, the gradient at B normal to the embedded boundary segment is

$$\left(\vec{g}_v^B \cdot \widehat{n}_v^B\right) = \frac{n_{v,x}^B}{h}\left(\frac{2-\bar{x}^B}{1-\bar{x}^B}\,\phi_C \;-\; \frac{1-\bar{x}^B}{2-\bar{x}^B}\,\phi_D\right) \tag{2.13}$$

where $\bar{x}^B h = x^B - i^B h$ is the x-component of the distance from the centroid of the embedded boundary face B to the center of the cell it is in. The terms $n_{v,x}^B$ and $\bar{x}^B$ are associated with the major coordinate.

Note that the value of the solution at a VoF v does not affect the value of the gradient at B, and therefore does not contribute to the Laplacian at v via this boundary condition.

The method just described can be extended to the case where the cells are not all regular. Define VoF_0 to be the VoF containing the current embedded boundary. Let VoF_1 be the set of VoFs connected to VoF_0 via the face the normal first intersects (what is done if the normal intersects an edge or corner is described below). Note, all the VoFs, VoF_1, will lie in one cell, $Cell_1$. Continuing in this fashion, let VoF_2 be the set

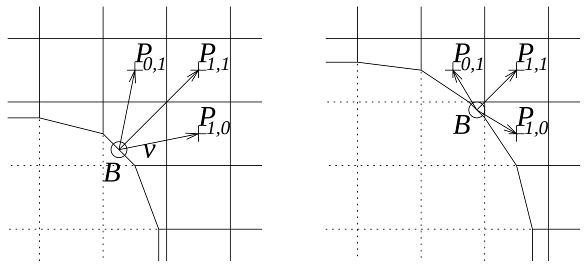

Figure 2.6: Dirichlet boundary condition at embedded boundary segment. Stencil for least-squares fit. Left:: typical situation. Right:: nearly degenerate situation.

of all the VoFs connected to a VoF in VoF_1 via the second face the normal intersects and they will all lie in $Cell_2$. Observe that one of the $Cell_i$ will be the cell that C lies in, C_0, and one will be the cell that D lies in, D_0. Call these cells $Cell_C$ and $Cell_D$, respectively. Now, the set of VoFs corresponding to these cells, VoF_C and VoF_D, may or may not be empty. If VoF_C is empty then we approximate the gradient at B normal to the embedded boundary by 0. If VoF_C is not empty but VoF_D is empty we will use linear interpolation to compute the value a C

If the cells containing C and D, described above, are not regular or are part of a coarse-fine interface, a different method is used to approximate the normal gradient. The gradient also can be found by a least-squares minimization method. See figure 2.6. Note that the components of the normal in this example are positive. If they are not, the problem is transformed via coordinate swaps into a coordinate system in which they are. We wish to find the normal gradient at B, the centroid of an embedded boundary face, in a VoF v. The details depend on the dimensionality of the problem. The two-dimensional case is described first.

In two dimensions, we select three cells adjacent to v's cell, two directly adjacent and one diagonally adjacent. The centers of these cells are the points $P_{1,0}$, $P_{0,1}$ and $P_{1,1}$. Note that the points $P_{1,0}$ *etc.* are always the centers of the cells, even if a cell is irregular. We then do a least-squares fit on the gradients in the directions $BP_{0,1}$, $BP_{1,0}$ and $BP_{1,1}$ (which are known) to determine the components of the full vector gradient (ϕ^B_x, ϕ^B_y).

Figure 2.6 also shows the need for a least-squares fit, rather than a coordinate transformation. On the left of the figure is the situation if the volume of v is not small. In this situation, the gradients in the directions $BP_{1,0}$ and $BP_{0,1}$ are linearly independent, so it is possible to compute the full gradient from them alone. On the right is the situation in the degenerate case in which the volume of v is small (and the normal $\widehat{n}^B_v$ is not aligned with the grid). The point B is almost at the corner of the grid cell, directly between $P_{1,0}$ and $P_{0,1}$. Thus, the gradients in the directions $BP_{1,0}$ and $BP_{0,1}$ are not linearly independent, and a full gradient cannot be computed from them alone.

The overdetermined system we need to approximate is

$$\begin{aligned}\phi_{1,0}-\phi^B &= \left(x_{1,0}-x^B\right)\phi_x^B+\left(y_{1,0}-y^B\right)\phi_y^B \\ \phi_{0,1}-\phi^B &= \left(x_{0,1}-x^B\right)\phi_x^B+\left(y_{0,1}-y^B\right)\phi_y^B \\ \phi_{1,1}-\phi^B &= \left(x_{1,1}-x^B\right)\phi_x^B+\left(y_{1,1}-y^B\right)\phi_y^B \end{aligned} \tag{2.14}$$

or, with $\bar{x}h = x - i^B h$, and replacing $\bar{x}_{1,0}$ *etc.* with the actual values (which are always either unity or zero because $P_{1,0}$ *etc.* are cell centers),

$$\mathbf{A}\, g = \Delta\Phi \tag{2.15}$$

where

$$\Delta\Phi = \left\{ \begin{array}{c} \phi_{1,0}-\phi^B \\ \phi_{0,1}-\phi^B \\ \phi_{1,1}-\phi^B \end{array} \right\}, \quad \mathbf{A} = \left[\begin{array}{cc} 1-\bar{x}^B & -\bar{y}^B \\ -\bar{x}^B & 1-\bar{y}^B \\ 1-\bar{x}^B & 1-\bar{y}^B \end{array} \right] h, \quad g = \left\{ \begin{array}{c} \phi_x^B \\ \phi_y^B \end{array} \right\}. \tag{2.16}$$

Note that $\phi^B = 0$. The least-squares approximation to equation 2.15 is the solution g to

$$\mathbf{M}\, g = b \tag{2.17}$$

where

$$\mathbf{M} = \mathbf{A^T A}, \quad b = \mathbf{A^T}\Delta\Phi. \tag{2.18}$$

The normal gradient is then

$$\left(\vec{g}_v^B \cdot \widehat{n}_v^B\right) \; = \; \widehat{n}_v^B \cdot g \; = \; n_{v,x}^B \phi_x^B \; + \; n_{v,y}^B \phi_y^B \tag{2.19}$$

If any of the cells containing a point $P_{1,0}$ *etc.* are irregular and contain multiple VoFs, we use for the value ϕ at that cell's center the volume-weighted average of the values of ϕ in the set of VoFs in that cell which are correctly connected to the VoF v. Note that the appropriate set of VoFs for the diagonal neighbor $P_{1,1}$ is the set of VoFs in that cell that are connected to the appropriate VoFs in both the cells of $P_{1,0}$ and $P_{0,1}$. See figure 2.7. In the figure, the VoFs v_0, v_1 and v_2 are in the stencil because they are directly connected to v. The VoF v_3 is in an adjacent cell, but is not connected to v. We call the set of VoFs $\{v_0, v_1\}$ the *x-neighbors* of v and the set of VoFs $\{v_2\}$ the *y-neighbors* of v. Of the VoFs in the diagonally adjacent cell, the VoF v_4 is in the stencil because it is connected both to an x-neighbor of v (namely v_0), and to a y-neighbor of v (namely v_2); the VoF v_5 is excluded because it is connected neither to an x-neighbor nor a y-neighbor of v; and the VoF v_6 is excluded because, although it is connected to an x-neighbor of v (namely v_1), is is not connected to any y-neighbor.

In three dimensions, we need four equations to form an overdetermined system for three components of the full vector gradient. We use the directional gradients from B to each of four cell centers. The cells are the three directly adjacent cells and one cell that is diagonally adjacent in the plane through v's cell that is normal to the direction

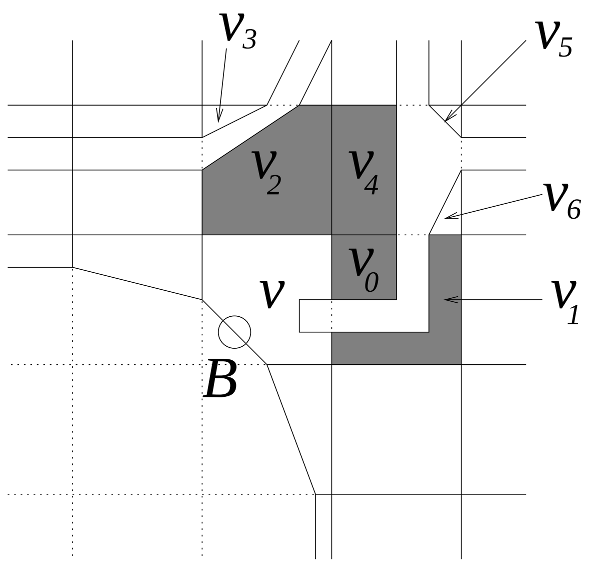

Figure 2.7: VoFs in the least-squares stencil. The VoFs v_0, v_1, v_2 and v_4 are in the stencil for the gradient at B.

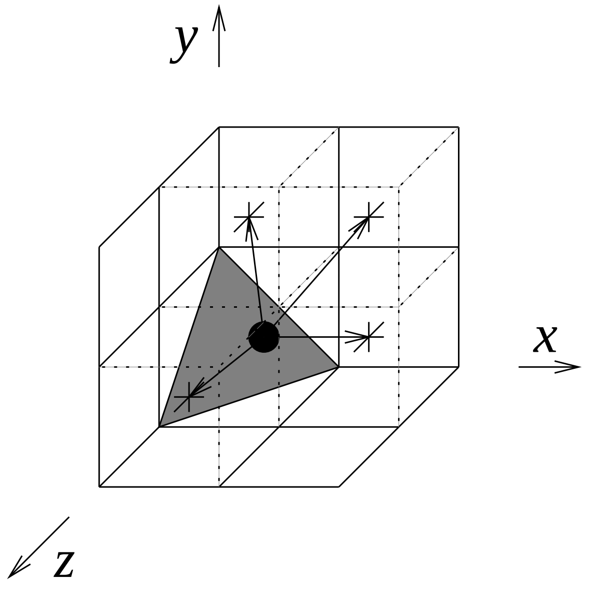

Figure 2.8: Least-squares stencil for three dimensions. The centroid of the embedded boundary face is the •. The centers of the neighbor cells for the least-squares approximation are the $+$s.

in which the component of the normal $\widehat{n}_v^B$ is the least. See figure 2.8. In the figure, $n_{v,x}^B > n_{v,y}^B > n_{v,z}^B$, so that the fourth point is $P_{1,1,0}$, the center of the cell in the same xy-plane as the center of v's cell. We now have

$$\mathbf{A}\, g = \Delta\Phi \tag{2.20}$$

where

$$\Delta\Phi = \left\{ \begin{array}{c} \phi_{1,0,0} - \phi^B \\ \phi_{0,1,0} - \phi^B \\ \phi_{0,0,1} - \phi^B \\ \phi_{1,1,0} - \phi^B \end{array} \right\}, \quad \mathbf{A} = \left[\begin{array}{ccc} 1 - \bar{x}^B & -\bar{y}^B & -\bar{z}^B \\ -\bar{x}^B & 1 - \bar{y}^B & -\bar{z}^B \\ -\bar{x}^B & -\bar{y}^B & 1 - \bar{z}^B \\ 1 - \bar{x}^B & 1 - \bar{y}^B & -\bar{z}^B \end{array} \right] h, \quad g = \left\{ \begin{array}{c} \phi_x^B \\ \phi_y^B \\ \phi_z^B \end{array} \right\} \tag{2.21}$$

which is approximated similarly to the two-dimensional case.

Again, the value of the solution at a VoF v does not affect the value of the gradient at B, and therefore does not contribute to the Laplacian at v via this boundary condition.

2.1.2 Relaxation Method

The relaxation method is based on Gauss-Seidel iteration with red-black ordering (GSRB) []. The VoFs are divided into three sets: irregular VoFs, and two sets of regular VoFs, red and black, such that every black VoF is adjacent only to red and irregular VoFs, and every red VoF is adjacent only to black and irregular VoFs. One iteration consists of the following steps:

- update the solution at the black VoFs,
- update the solution at the red VoFs,
- update the solution at the irregular VoFs N_{irreg} times, where $N_{\text{irreg}} > 0$ is an adjustable parameter.

The solution at a VoF is updated by incrementing it with

$$\delta\phi_v = \alpha_v \left(\rho_v - L_v\left(\phi\right)\right) \tag{2.22}$$

where α_v is a relaxation coefficient designed to annihilate the diagonal terms of the differential operator $L_v(\phi)$. It is constructed by applying the operator to a delta function and taking the inverse,

$$\frac{1}{\alpha_v} = L_v\left(\delta_v\right) \tag{2.23}$$

where

$$\delta_v\left(\vec{x}\right) = \left\{ \begin{array}{ll} 1 & \text{if } \vec{x} \text{ is within } v \\ 0 & \text{otherwise} \end{array} \right. \tag{2.24}$$

The operator L_v must include the face boundary conditions (see section 2.1.1.4), but does not require the embedded boundary segment boundary conditions (see section 2.1.1.5) because in the latter the contribution to the operator at the VoF v does not depend on the value of the solution at v.

2.1.3 Multigrid Algorithm

Multigrid is a method for the acceleration of convergence of iterative schemes for elliptic and parabolic equations. It involves the creation of a sequence of coarser grids on which a coarser problem is solved. A procedure is also specified for transferring solution data between grids of different resolution. For solving a linear problem, we use the residual-correction form of multigrid. One multigrid cycle consists of the following sequence of steps:

- perform N_{pre} iterations of the relaxation procedure
- restrict residual from this grid to the next coarser grid:
$$\rho^{2h} = I_h^{2h}\left(\rho^h - L^h(\phi^h)\right) \tag{2.25}$$
- perform N_{cycles} multigrid cycles on the coarser grid to solve
$$L^{2h}(\phi^{2h}) = \rho^{2h} \tag{2.26}$$
- interpolate correction from the next coarser grid to this grid:
$$e^h = I_{2h}^h\left(\phi^{2h}\right) \tag{2.27}$$
- increment solution on this grid with the correction e^h
- perform N_{post} iterations of the relaxation procedure

If $N_{cycles} = 1$, the method is called a V-cycle; if $N_{cycles} = 2$, the method is called a W-cycle; other values of N_{cycle} are unusual.

Something about bottom solves.

We use the sequence of coarse grids produced by `EBIndexSpace`'s coarsening algorithm. A grid has half the resolution of the finer grid from which it was created, and the volumes of VoFs and areas of faces are "conserved,"

$$\begin{aligned} \Lambda_v &= \frac{1}{2^D} \sum_{v' \in \mathsf{refine}(v)} \Lambda_{v'} \\ \ell_f &= \frac{1}{2^{D-1}} \sum_{f' \in \mathsf{refine}(f)} \Lambda_{f'}. \end{aligned} \tag{2.28}$$

Data is transferred to a coarser grid by a volume-weighted average restriction operator I_h^{2h}, defined as

$$\phi_v = \frac{1}{2^D \Lambda_v} \sum_{v' \in \mathsf{refine}(v)} \Lambda_{v'} \phi_{v'}. \tag{2.29}$$

Data is transferred to a finer grid by piecewise-constant interpolation operator I_{2h}^h.

Because we are using the residual-correction form of multigrid, all boundary conditions on the coarser grids are homogeneous.

2.1.4 Viscous operator discretization

In this section, we define a Helmholtz operator and how we solve it. We are solving

$$(I + \mu L)\phi = rho \tag{2.30}$$

where μ is a constant. Just as we did for the MAC projection, we discretize $L \equiv DG^{mac}$ (see equation **??**). In this context, however, since the irregular boundary is a no-slip boundary, we must solve (2.30) with Dirichlet boundary conditions $\phi = 0$ on the irregular boundary. To do this, we must compute

$$F^B = \frac{\partial \phi}{\partial \hat{n}}$$

at the embedded boundary. We follow Schwartz, et. al [7] and compute this gradient by casting a ray into space, interpolating ϕ to points along the ray, and computing the normal gradient of phi by differencing the result. We cast a ray along the normal of the VoF from the centroid of area of the irregular face C. We find the closest points B and C where the ray intersects the planes formed by cell centered points. The axes of these planes d_1, d_2 will be the directions not equal to the largest direction of the normal. We use biquadratic interpolation to interpolate data from the nearest cell centers to the intersection points B and C. In two dimensions, we find the nearest lines of cell centers (instead of planes) and the interpolation is quadratic. We then use this interpolated data to compute a $O(h^2$ approximation of $\frac{\partial \phi}{\partial \hat{n}}$. In the case where there are not enough cells to cast this ray, we use a least-squares approximation to $\frac{\partial \phi}{\partial \hat{n}}$ which is $O(h)$. As shown in [3], the modified equation analysis shows that, for Dirichlet boundary conditions, it is sufficient to have $O(1)$ boundary conditions to achieve second order solution error convergence for elliptic equations.

2.1.5 Slope Calculation

The notation

$$CC = A \mid B \mid C$$

means that the 3-point formula A is used for CC if all cell-centered values it uses are available, the 2-point formula B is used if the cell to the right (i.e. the high side) of the current cell is covered, and the 2-point formula C is used if the cell to the left (i.e. the low side) current cell is covered.

To compute the limited differences in the first step on the algorithm, we use the second-order slope calculation [1] with van Leer limiting.

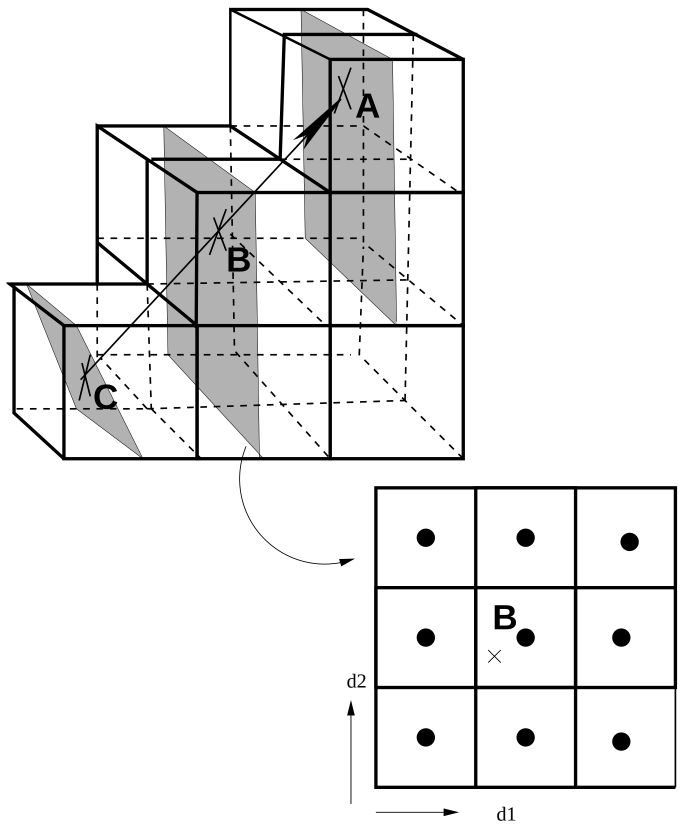

Figure 2.9: Ray casting to get fluxes for Dirichlet boundary conditions at the irregular boundary. A ray is cast along the normal from the centroid of the irregular area C and the points A and B are the places where this ray intersects the planes formed by cell centers. Data is interpolated in these planes to get values at the intersection points. That data is used to compute a normal gradient of the solution.
.

$$\begin{aligned}
\Delta_2^d W_{\boldsymbol{i}} = &\ \Delta^{vL}(\Delta^C W_{\boldsymbol{i}}, \Delta^L W_{\boldsymbol{i}}, \Delta^R W_{\boldsymbol{i}}) \mid \Delta^{VLL} W_{\boldsymbol{i}} \mid \Delta^{VLR} W_{\boldsymbol{i}} \\
\Delta^B W_{\boldsymbol{i}} = &\ \tfrac{2}{3}((W - \tfrac{1}{4}\Delta_2^d W)_{\boldsymbol{i}+\boldsymbol{e}^d} - (W + \tfrac{1}{4}\Delta_2^d W)_{\boldsymbol{i}-\boldsymbol{e}^d}) \\
\Delta^C W_{\boldsymbol{i}} = &\ \tfrac{1}{2}(W^n_{\boldsymbol{i}+\boldsymbol{e}^d} - W^n_{\boldsymbol{i}-\boldsymbol{e}^d}) \\
\Delta^L W_{\boldsymbol{i}} = &\ W^n_{\boldsymbol{i}} - W^n_{\boldsymbol{i}-\boldsymbol{e}^d} \\
\Delta^R W_{\boldsymbol{i}} = &\ W^n_{\boldsymbol{i}+\boldsymbol{e}^d} - W^n_{\boldsymbol{i}} \\
\Delta^{3L} W_{\boldsymbol{i}} = &\ \tfrac{1}{2}(3W^n_{\boldsymbol{i}} - 4W^n_{\boldsymbol{i}-\boldsymbol{e}^d} + W^n_{\boldsymbol{i}-2\boldsymbol{e}^d}) \\
\Delta^{3R} W_{\boldsymbol{i}} = &\ \tfrac{1}{2}(-3W^n_{\boldsymbol{i}} + 4W^n_{\boldsymbol{i}+\boldsymbol{e}^d} - W^n_{\boldsymbol{i}+2\boldsymbol{e}^d})
\end{aligned}$$

$$\begin{aligned}
\Delta^{VLL} W_{\boldsymbol{i}} = &\ \min(\Delta^{3L} W_{\boldsymbol{i}}, \Delta^L W_{\boldsymbol{i}}) && \text{if } \Delta^{3L} W_{\boldsymbol{i}} \cdot \Delta^L W_{\boldsymbol{i}} > 0 \\
\Delta^{VLL} W_{\boldsymbol{i}} = &\ 0 && \text{otherwise} \\
\Delta^{VLR} W_{\boldsymbol{i}} = &\ \min(\Delta^{3R} W_{\boldsymbol{i}}, \Delta^R W_{\boldsymbol{i}}) && \text{if } \Delta^{3R} W_{\boldsymbol{i}} \cdot \Delta^R W_{\boldsymbol{i}} > 0 \\
\Delta^{VLR} W_{\boldsymbol{i}} = &\ 0 && \text{otherwise}
\end{aligned}$$

We apply the van Leer limiter component-wise to the differences.

Chapter 3

Viscous Tensor Equation

3.1 Viscous Tensor Equation

This section describes the method for solving the elliptic partial differential equation

$$\kappa L\vec{v} = \kappa\alpha\vec{v} + \beta\nabla\cdot F = \kappa\rho.$$

α is a constant and $\beta = \beta(\vec{x})$. F is given by

$$F = \eta(\nabla\vec{v} + \nabla\vec{v}^T) + \lambda(I\nabla\cdot\vec{v}) \tag{3.1}$$

where I is the identity matrix, $\eta = \eta(\vec{x})$, and $\lambda = \lambda(\vec{x})$.

3.1.1 Discretization

We discretize normal components of the face-centered gradient using an average of cell-centered gradients for tangential components and a centered-difference approximation to the normal gradient.

$$(\nabla\vec{v})^{d'}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \begin{pmatrix} \frac{1}{h}(\vec{v}_{\boldsymbol{i}+\boldsymbol{e}^d} - \vec{v}_{\boldsymbol{i}}) \text{ if } d = d' \\ \frac{1}{2}((\nabla\vec{v})^{d'}_{\boldsymbol{i}+\boldsymbol{e}^d} + (\nabla\vec{v})^{d'}_{\boldsymbol{i}}) \text{ if } d \neq d' \end{pmatrix}$$

where

$$(\nabla\vec{v})^d_{\boldsymbol{i}} = \frac{1}{2h}(\vec{v}_{\boldsymbol{i}+\boldsymbol{e}^d} - \vec{v}_{\boldsymbol{i}-\boldsymbol{e}^d}).$$

We discretize the divergence as follows

$$(\kappa\nabla\cdot F)_{\boldsymbol{i}} = \sum_{d'=1}^{D}(\alpha\nabla F)^{d'}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + \alpha_B F^B$$

where κ and α are the volume and area fractions.

We use equation 3.1 get the flux at cell face centers. We then interpolate the flux to face centroids. In two dimensions, this interpolation takes the form

$$\widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}} = F_{\boldsymbol{f}}^{n+\frac{1}{2}} + |\bar{x}|(F_{\boldsymbol{f} \ll sign(\bar{x})\boldsymbol{e}^d}^{n+\frac{1}{2}} - F_{\boldsymbol{f}}^{n+\frac{1}{2}}) \tag{3.2}$$

where $\bar{x}$ is the centroid in the direction d perpendicular to the face normal. In three dimensions, define $(\bar{x}, \bar{y})$ to be the coordinates of the centroid in the plane (d^1, d^2) perpendicular to the face normal.

$$\widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}} = F_{\boldsymbol{f}}^{n+\frac{1}{2}}(1 - \bar{x}\bar{y} + |\bar{x}\bar{y}|) + \tag{3.3}$$

$$F_{\boldsymbol{f} \ll sign(\bar{x})\boldsymbol{e}^{d^1}}^{n+\frac{1}{2}}(|\bar{x}| - |\bar{x}\bar{y}|) + \tag{3.4}$$

$$F_{\boldsymbol{f} \ll sign(\bar{x})\boldsymbol{e}^{d^2}}^{n+\frac{1}{2}}(|\bar{y}| - |\bar{x}\bar{y}|) + \tag{3.5}$$

$$F_{\boldsymbol{f} << sign(\bar{x})\boldsymbol{e}^{d^1} << sign(\bar{x})\boldsymbol{e}^{d^2}}^{n+\frac{1}{2}}(|\bar{x}\bar{y}|) \tag{3.6}$$

Centroids in any dimension are normalized by Δx and centered at the cell center. This interpolation is only done if the shifts that are used in the interpolation are uniquely-defined and single-valued. We use a conservative discretization for the flux divergence.

$$\kappa_{\boldsymbol{v}} \nabla \cdot \vec{F} \equiv (D \cdot \vec{F}) = ((\sum_{d=0}^{\mathbf{D}-1} \sum_{\pm=+,-} \sum_{\boldsymbol{f} \in \mathcal{F}_{\boldsymbol{v}}^{d,\pm}} \pm \alpha_{\boldsymbol{f}} \widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}}) + \alpha_{\boldsymbol{v}}^B F_{\boldsymbol{v}}^{B,n+\frac{1}{2}}) \tag{3.7}$$

where where F^B is the flux at the irregular boundary, wherein lies most of the difficulty in this operator.

3.1.2 Flux at the Boundary

In all cases, we construct the gradient at the boundary and use equation 3.1 to construct the flux.

For Neumann boundary conditions, the gradient of the solution is specified at the boundary.

For Dirichlet boundary conditions, the gradient normal to the boundary is determined using the value at the boundary. The gradients tangential to the boundary are specified. For irregular boundaries, the procedure for calculating the gradient normal to the boundary is given in section 2.1.1.5. For domain boundaries, we construct a quadratic function with the value at the boundary and the two adjacent points along the normal to construct the gradient. For example, say we are at the low side of the domain with a value ϕ_0 at the boundary. The normal gradient is given by. that means that normal gradient is given by

$$(\nabla \phi)^x_{-\frac{1}{2},j,k} = \frac{9(\phi_{0,j,k} - \phi_0) - (\phi_{1,j,k} - \phi_0)}{3\Delta x}$$

Chapter 4

Conductivity Operator

4.1 Conductivity Equation

This section describes the method for solving the elliptic partial differential equation

$$\kappa L\phi = \kappa\alpha a\phi + \beta\nabla\cdot F = \kappa\rho.$$

α and $beta$ are constants, a is a function of space and F is given by

$$F = b\nabla\phi \tag{4.1}$$

The conductivity b is a function of space.

4.1.1 Discretization

We discretize the face-centered gradient for the flux using a centered-difference approximation.

$$(\nabla\phi)^d_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} = \frac{1}{h}(phi_{\boldsymbol{i}+\boldsymbol{e}^d} - \phi_{\boldsymbol{i}})$$

We discretize the divergence as follows

$$(\kappa\nabla\cdot F)_{\boldsymbol{i}} = \sum_{d'=1}^{D}(\alpha\nabla F)^{d'}_{\boldsymbol{i}+\frac{1}{2}\boldsymbol{e}^d} + \alpha_B F^B$$

where κ and α are the volume and area fractions. We use equation 4.1 get the flux at cell face centers. We then interpolate the flux to face centroids. In two dimensions, this interpolation takes the form

$$\widetilde{F}^{n+\frac{1}{2}}_{\boldsymbol{f}} = F^{n+\frac{1}{2}}_{\boldsymbol{f}} + |\bar{x}|(F^{n+\frac{1}{2}}_{\boldsymbol{f}\ll sign(\bar{x})\boldsymbol{e}^d} - F^{n+\frac{1}{2}}_{\boldsymbol{f}}) \tag{4.2}$$

where $\bar{x}$ is the centroid in the direction d perpendicular to the face normal. In three dimensions, define $(\bar{x}, \bar{y})$ to be the coordinates of the centroid in the plane (d^1, d^2) perpendicular to the face normal.

$$\widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}} = F_{\boldsymbol{f}}^{n+\frac{1}{2}}(1 - \bar{x}\bar{y} + |\bar{x}\bar{y}|) + \tag{4.3}$$

$$F_{\boldsymbol{f} \ll sign(\bar{x})\boldsymbol{e}^{d^1}}^{n+\frac{1}{2}}(|\bar{x}| - |\bar{x}\bar{y}|) + \tag{4.4}$$

$$F_{\boldsymbol{f} \ll sign(\bar{x})\boldsymbol{e}^{d^2}}^{n+\frac{1}{2}}(|\bar{y}| - |\bar{x}\bar{y}|) + \tag{4.5}$$

$$F_{\boldsymbol{f} << sign(\bar{x})\boldsymbol{e}^{d^1} << sign(\bar{x})\boldsymbol{e}^{d^2}}^{n+\frac{1}{2}}(|\bar{x}\bar{y}|) \tag{4.6}$$

Centroids in any dimension are normalized by Δx and centered at the cell center. This interpolation is only done if the shifts that are used in the interpolation are uniquely-defined and single-valued. We use a conservative discretization for the flux divergence.

$$\kappa_{\boldsymbol{v}} \nabla \cdot \vec{F} \equiv (D \cdot \vec{F}) = ((\sum_{d=0}^{\mathbf{D}-1} \sum_{\pm=+,-} \sum_{\boldsymbol{f} \in \mathcal{F}_{\boldsymbol{v}}^{d,\pm}} \pm \alpha_{\boldsymbol{f}} \widetilde{F}_{\boldsymbol{f}}^{n+\frac{1}{2}}) + \alpha_{\boldsymbol{v}}^{B} F_{\boldsymbol{v}}^{B,n+\frac{1}{2}}) \tag{4.7}$$

where where F^B is the flux at the irregular boundary, wherein lies most of the difficulty in this operator.

4.1.2 Flux at the Boundary

In all cases, we construct the gradient at the boundary and use equation 4.1 to construct the flux.

For Neumann boundary conditions, the gradient of the solution is specified at the boundary.

For Dirichlet boundary conditions, the gradient normal to the boundary is determined using the value at the boundary. The gradients tangential to the boundary are specified. For irregular boundaries, the procedure for calculating the gradient normal to the boundary is given in section 2.1.1.5. For domain boundaries, we construct a quadratic function with the value at the boundary and the two adjacent points along the normal to construct the gradient. For example, say we are at the low side of the domain with a value ϕ_0 at the boundary. The normal gradient is given by. that means that normal gradient is given by

$$(\nabla\phi)^{x}_{-\frac{1}{2},j,k} = \frac{9(\phi_{0,j,k} - \phi_0) - (\phi_{1,j,k} - \phi_0)}{3\Delta x}$$

Chapter 5

EBAMRElliptic Interface

5.1 Overview

The principal `EBAMRElliptic` classes are:

- `EBPoissonOp` conforms to the `MGLevelOp` interface and is used to solve Poisson's equation over a single level.
- `EBAMRPoissonOp` conforms to the `AMRLevelOp` interface and is used to solve Poisson's (or Helmholtz's) equation over an AMR hierarchy with constant coefficients.
- `EBConductivityOp` conforms to the `AMRLevelOp` interface and is used to solve Poisson's (or Helmholtz's) equation over an AMR hierarchy with variable coefficients.
- `EBViscousTensorOp` conforms to the `AMRLevelOp` interface and is used to solve the viscous tensor equation over an AMR hierarchy with variable coefficients.
- `EBAMRTGA` advances a solution of the heat equation one step using the TGA [8] algorithm.

The first two, since their interface is well described in the Chombo Design document [2] will only be described through their factories, since the factories are the parts of the interface that the user actually has to use in order to use the class.

5.2 `EBPoissonOpFactory`

Factory to generate an operator to solve $(\alpha + \beta L)\phi = \rho$. This follows the MGLevelOp interface.

```
EBPoissonOpFactory(const EBLevelGrid&                              eblgs,
                   const RealVect&                                 dx,
                   const RealVect&                              origin,
                   const int&                                   orderEB,
                   const int&                                   numPreCondIters,
                   const int&                                   relaxType,
                   RefCountedPtr<BaseDomainBCFactory>           domainBCFactory,
                   RefCountedPtr<BaseEBBCFactory>               ebBcFactory,
                   const Real&                                  alpha,
                   const Real&                                  beta,
                   const IntVect&                               ghostCellsPhi,
                   const IntVect&                               ghostCellsRhs);
```

- `eblgs` : layout of the level
- `domainFactory` : domain boundary conditions
- `ebBCFactory`: eb boundary conditions
- `dxCoarse`: grid spacing at coarsest level
- `origin`: offset to lowest corner of the domain
- `refRatio`: refinement ratios. refRatio[i] is between levels i and i+1
- `preCondIters`: number of iterations to do for pre-conditioning
- `relaxType`: 0 means point Jacobi, 1 is Gauss-Seidel, 2 is line solver.
- `orderEB`: 0 to not do flux interpolation at cut faces.
- `alpha`: coefficient of identity
- `beta`: coefficient of Laplacian.
- `ghostCellsPhi`: Number of ghost cells in phi, correction (typically one)
- `ghostCellsRhs`: Number of ghost cells in RHS, residual, lphi (typically zero) Ghost cell arguments are there for caching reasons. Once you set them, an error is thrown if you send in data that does not match.

5.3 `EBAMRPoissonOpFactory`

Factory to generate an operator to solve $(\alpha + \beta L)\phi = \rho$. This follows the AMRLevelOp interface.

```
EBAMRPoissonOpFactory(const Vector<EBLevelGrid>&                        eblgs,
                      const Vector<int>&                                refRatio,
                      const Vector<RefCountedPtr<EBQuadCFInterp> >& quadCFI,
                      const RealVect&                                   dxCoarse,
                      const RealVect&                                   origin,
                      const int&                                        orderEB,
                      const int&                                        numPreCondIters,
                      const int&                                        relaxType,
                      RefCountedPtr<BaseDomainBCFactory>                domainBCFactory,
                      RefCountedPtr<BaseEBBCFactory>                    ebBcFactory,
                      const Real&                                       alpha,
                      const Real&                                       beta,
                      const Real&                                       time,
                      const IntVect&                                    ghostCellsPhi,
                      const IntVect&                                    ghostCellsRhs,
                      int numLevels = -1);
```

- `eblgs` : layouts at each AMR level
- `domainFactory` : domain boundary conditions
- `ebBCFactory`: eb boundary conditions
- `dxCoarse`: grid spacing at coarsest level
- `origin`: offset to lowest corner of the domain
- `refRatio`: refinement ratios. refRatio[i] is between levels i and i+1
- `preCondIters`: number of iterations to do for pre-conditioning
- `relaxType`: 0 means point Jacobi, 1 is Gauss-Seidel, 2 is line solver.
- `orderEB`: 0 to not do flux interpolation at cut faces.
- `alpha`: coefficient of identity
- `beta`: coefficient of Laplacian.
- `time`: time for boundary conditions
- `ghostCellsPhi`: Number of ghost cells in phi, correction (typically one)
- `ghostCellsRhs`: Number of ghost cells in RHS, residual, lphi (typically zero) Ghost cell arguments are there for caching reasons. Once you set them, an error is thrown if you send in data that does not match. Use numlevels = -1 if you want to use the size of the vectors for numlevels.

5.4 EBAMRTGA

EBAMR implementation of the TGA algorithm to solve the heat equation.

```
  EBAMRTGA(const Vector<EBLevelGrid>&                      eblg,
           const Vector<int>&                              refRatio,
           const Vector<RefCountedPtr<EBQuadCFInterp> >& quadCFI,
           const RealVect&                                 dxCoar,
           const RefCountedPtr<BaseDomainBCFactory>&     domainBCFactory,
           const RefCountedPtr<BaseEBBCFactory>&         ebBCFactory,
           const int&                                      numlevels,
           const RealVect&                                 origin,
           const Real&                                     diffusionConst,
           const IntVect&                                  ghostCellsPhi,
           const IntVect&                                  ghostCellsRHS,
           const int&                                      numSmooths,
           const int&                                      iterMax,
           const int&                                      ODESolver,
           const int&                                      numMGCycles,
           const int&                                      numPreCondIters,
           const int&                                      relaxType,
           const int&                                      verbocity);
```

5.5 Example

5.6 Snippet to solve Poisson's equation

```
void solve(const PoissonParameters&  a_params,
  Vector<LevelData<EBCellFAB>* >& phi,
  Vector<LevelData<EBCellFAB>* >& rhs,
  Vector<DisjointBoxLayout>&      grids,
  Vector<EBISLayout>&             ebisl)
  )
{
  int nvar = 1;
  //create the solver
  AMRMultiGrid<LevelData<EBCellFAB> > solver;
  pout() << "defining  solver" << endl;

  BiCGStabSolver<LevelData<EBCellFAB> > newBottomSolver;
  newBottomSolver.verbosity = 0;
  defineSolver(solver, grids, ebisl, newBottomSolver, a_params,1.e99);
  pout() << "solving " << endl;
  //solve the equation
  solver.solve(phi, rhs, a_params.maxLevel, 0);
}
```

5.7 Snippet to Project a Cell-Centered Velocity Field

```
void projectVel(
              const Vector< DisjointBoxLayout >&   a_grids,
              const Vector< EBISLayout >&          a_ebisl,
              const PoissonParameters&             a_params)
              const int&                           a_dofileout,
              const bool&                          a_isFine)
              Vector<LevelData<EBCellFAB>* >& velo,
              Vector<LevelData<EBCellFAB>* >& gphi)
{
  int nlevels = a_params.numLevels;

  RealVect dxLevCoarsest = RealVect::Unit;
  dxLevCoarsest *=a_params.coarsestDx;
  ProblemDomain domLevCoarsest(a_params.coarsestDomain);

  RealVect dxLev = dxLevCoarsest;

  Real domVal = 0.0;
  NeumannPoissonDomainBCFactory* domBCPhi = new NeumannPoissonDomainBCFactory();
  RefCountedPtr<BaseDomainBCFactory> baseDomainBCPhi = domBCPhi;
  domBCPhi->setValue(domVal);
  DirichletPoissonDomainBCFactory* domBCVel = new DirichletPoissonDomainBCFactory();
  RefCountedPtr<BaseDomainBCFactory> baseDomainBCVel = domBCVel;
  domBCVel->setValue(domVal);

  NeumannPoissonEBBCFactory*      ebBCPhi = new NeumannPoissonEBBCFactory();
  ebBCPhi->setValue(domVal);
  RefCountedPtr<BaseEBBCFactory>     baseEBBCPhi     = ebBCPhi;


  Vector<LevelData<EBCellFAB>*> rhoinv;
  const int bottomSolverType = 1;

  Vector<EBLevelGrid>                        eblg   (a_grids.size());
  Vector<RefCountedPtr<EBQuadCFInterp> >   quadCFI(a_grids.size(), NULL);
  domLev = domLevCoarsest;
  for(int ilev = 0; ilev < a_grids.size(); ilev++)
```

```
    {
      int nvar = 1;
      int nref = a_params.refRatio[ilev];
      eblg[ilev] = EBLevelGrid(a_grids[ilev], a_ebisl[ilev], domLev);
      if(ilev > 0)
        {
          int nrefOld = a_params.refRatio[ilev-1];
          ProblemDomain domLevCoar = coarsen(domLev, nrefOld);
          quadCFI[ilev] = new EBQuadCFInterp(a_grids[ilev  ],
                                             a_grids[ilev-1],
                                             a_ebisl[ilev  ],
                                             a_ebisl[ilev-1],
                                             domLevCoar,
                                             nrefOld, nvar,
                                             *(eblg[ilev].getCFIVS()));

        }
      domLev.refine(nref);
    }


  EBCompositeCCProjector projectinator(eblg,  a_params.refRatio, quadCFI,
                                       a_params.coarsestDx,
                                       RealVect::Zero,
                                       baseDomainBCVel,
                                       baseDomainBCPhi,
                                       baseEBBCPhi,
                                       rhoinv, false, true, -1, 3 ,40,1.e99, 1,
  projectinator.project(velo, gphi);

}
```

Bibliography

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

[2] P. Colella, D. T. Graves, N.D. Keen, T. J. Ligocki, D. F. Martin, P.W. McCorquodale, D. Modiano, P.O. Schwartz, T.D. Sternberg, and B. Van Straalen. Chombo Software Package for AMR Applications - Design Document. unpublished, 2000.

[3] H. Johansen and P. Colella. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *J. Comput. Phys.*, 147(2):60–85, December 1998.

[4] Hans Johansen and Phillip Colella. A cartesian grid embedded boundary method for Poisson's equation on irregular domains. *J. Comput. Phys.*, 1998.

[5] Hans Svend Johansen. *Cartesian Grid embedded Boundary Finite Difference Methods for Elliptic and Parabolic Partial Differential Equations on Irregular Domains*. PhD thesis, University of California, Berkeley, 1997.

[6] D. F. Martin and K. L. Cartwright. Solving Poisson's equation using adaptive mesh refinement. *Technical Report UCB/ERI M96/66 UC Berkeley*, 1996.

[7] P. Schwartz, M. Barad, P. Colella, and T. Ligocki. A Cartesian grid embedded boundary method for the heat equation and Poisson's equation in three dimensions. *J. Comput. Phys.*, 211(2):531–550, January 2006.

[8] E.H. Twizell, A.B. Gumel, and M.A. Arigu. Second-order, l_0-stable methods for the heat equation with time-dependent boundary conditions. *Advances in Computational Mathematics*, 6:333–352, 1996.