The Workshop Package for EBChombo Geometry Generation

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

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

Tue, Feb 4, 2003

Contents

1 Introduction

For some computations, it is possible to specify the domain of computation based upon functional description of the boundary interface. The workshop package provides a systematic algorithm and API that will produce an embedded boundary description of the

interface which converges to second order with grid refinement to the functional description of the interface. Gueyffier, et. al. [1] solve a similar problem in their volume-of-fluid method in three dimensions. We have used some of their ideas about renormalization in this work.

This document specifies the workshop algorithm and the associated API.

2 Algorithm

The workshop algorithm provides a numerically convergent way to convert a functional description of an interface to an embedded boundary description. Any given cell is either entirely inside the covered region (covered), entirely outside the covered region (regular), or intersected the cell by the interface (irregular). A workshop class must be able to identify cells as such. Once a cell is identified to be irregular, the workshop class then defines which direction is "up". This determines which coordinate direction of the surface is treated as the dependent variable in the cell ($y = \phi(x, z)$, for example). The workshop class must then provide the intersection of the surface with coordinate lines that vary only in the "up" direction.

2.1 Calculation of Edge Intersections

Given a surface, S, we have assumed that S is specified locally as a function. To fix notation, suppose that at a given n-dimensional cell,E, a coordinate direction x_i has been specified as well as a the local function $\phi : \mathbf{R}^{n-1} \rightarrow \mathbf{R}$, which describes S. For any $\mathbf{x} \in \partial E$, the intersection of S with the line in the x_i direction that passes through $\mathbf{x}$ may be calculated by evaluating $\phi(x_1, x_2, \cdots x_{i-1}, x_{i+1}, \cdots x_n)$.

For the edges of E that don't lie in the $\mathbf{e}_i$ direction an iterative method such as Brent's Algorithm may be used to calculate the intersection or observe the non-intersection of S with the edge. However for the special case of the discrete Divergence operator a simpler method suffices. We now address this special case.

Denote the endpoints of such an edge by $\mathbf{a}'$ and $\mathbf{c}'$. Let $\mathbf{a} = a_1, a_2, \cdots a_{i-1}, a_{i+1}, \cdots a_n)$. That is, $\mathbf{a}$ denotes the projection of $\mathbf{a}'$ onto the space spanned by the $n-1$ vectors $\mathbf{e}_j$ for $j \neq i$. Similarly, $\mathbf{c}$ denotes the projection of $\mathbf{c}'$ on the same space.

If $A = \phi(\mathbf{a}) > a_i$ and $C = \phi(\mathbf{c}) > c_i$ or $A = \phi(\mathbf{a}) < a_i$ and $C = \phi(\mathbf{c}) < c_i$, we conclude that S does not intersect the edge. If $A = \phi(\mathbf{a}) = a_i$ or $C = \phi(\mathbf{c}) = c_i$, then S intersects an endpoint of the segment. If $(A - a_i)(C - c_i) < 0$ then there is an intersection in the interior, which we estimate by linear interpolation.

2.2 Algorithm for d-dimensional cube sliced by a co-dimension 1 surface

Let Ω denote a polytope formed by the intersection of an d-dimensional cube with a codimension one surface. Let Θ denote the portion of the surface that forms one of the faces of Ω. Define B by $B = \partial\Omega - \Theta$. B comprises the faces of Ω that lie in coordinate directions.

We consider the problem of computing moments over Ω, assuming that the intersections of Θ with the one-dimensional edges of Ω have been calculated.

Let $\mathbf{k}$ denote a multiindex of whole numbers, $\mathbf{k} = k_1, \cdots k_n$. Let $\mathbf{x}^\mathbf{k}$ denote a monomial of order $|\mathbf{k}| = k_1 + \cdots + k_n$. That is,$\mathbf{x}^\mathbf{k} = x_1^{k_1} \cdots x_n^{k_n}$. Finally, let $\boldsymbol{e}_i$ denote a unit vector in the i direction.

Fix a whole number $p > 0$. For each coordinate direction, $\boldsymbol{e}_i$, for each multi-index, $\mathbf{k}$, of order $p+1$, we construct a vector field, $\boldsymbol{F} = \boldsymbol{F}(\mathbf{k}, i) = \mathbf{x}^\mathbf{k}\boldsymbol{e}_i$. Note that the divergence of $\boldsymbol{F}$ is either zero or a monomial of order p. Furthermore, every monomial of order p may be obtained as the divergence of such an $\boldsymbol{F}$. For example, let $\mathbf{j}$ denote a multi-index of order p, for each i, $\boldsymbol{F} = \mathbf{x}^\mathbf{j} x_i \boldsymbol{e}_i$ satisfies the condition $\frac{1}{j_i+1}\nabla \cdot \boldsymbol{F} = \mathbf{x}^\mathbf{j}$.

Therefore, using the divergence theorem we may write:

$$C \int_\Omega \mathbf{x}^\mathbf{j} d\Omega = \int_\Omega \nabla \cdot \boldsymbol{F} d\Omega = \int_{\partial\Omega} \boldsymbol{F} \cdot \boldsymbol{n}\, d(\partial\Omega) \tag{1}$$

where $\boldsymbol{n}$ denotes the outward unit normal and $C = j_i + 1$ is a constant.

If we let μ_i^+ and μ_i^- denote the two faces of Ω with outward normal $\boldsymbol{e}_i$ and $-\boldsymbol{e}_i$, respectively, we can rearrange the terms of last integral to observe:

$$C \int_\Omega \mathbf{x}^\mathbf{j}\, d\Omega - \int_\Theta \mathbf{x}^\mathbf{j} x_i \boldsymbol{n}_i^\theta d\theta \tag{2}$$

$$= \sum_{\mu \in M} \int_\mu \boldsymbol{F} \cdot \boldsymbol{n}^\mu d\mu \tag{3}$$

$$= \int_{\mu_i^+} \mathbf{x}^\mathbf{k}\, d\mu_i^+ - \int_{\mu_i^-} \mathbf{x}^\mathbf{k}\, d\mu_i^-. \tag{4}$$

where $\boldsymbol{n}^\theta$ denotes the normal to Θ and similarly for $\boldsymbol{n}^\mu$. μ_i^+ and μ_i^- are polytopes of dimension $n-1$.

We make the approximation that $\boldsymbol{n}_i^\theta$ is constant.

$$\int_\Theta \mathbf{x}^\mathbf{j} x_i \boldsymbol{n}_i^\theta d\theta \approx \boldsymbol{n}_i^\theta \int_\Theta \mathbf{x}^\mathbf{j} x_i d\theta \tag{5}$$

Using this approximation, we write an approximate divergence theorem expression:

$$C \int_\Omega \mathbf{x}^\mathbf{j}\, d\Omega - \boldsymbol{n}_i^\theta \int_\Theta \mathbf{x}^\mathbf{j} x_i d\theta \tag{6}$$

$$\approx \sum_{\mu \in M} \int_\mu \boldsymbol{F} \cdot \boldsymbol{n}^\mu d\mu \tag{7}$$

$$= \int_{\mu_i^+} \mathbf{x}^\mathbf{k}\, d\mu_i^+ - \int_{\mu_i^-} \mathbf{x}^\mathbf{k}\, d\mu_i^-. \tag{8}$$

We consider the the terms on the left as unknowns, and for the moment we assume that the terms on the right are known as well as the normal to Θ, $\boldsymbol{n}^\theta$. We replace $\approx$ by $=$ and form the linear system of all such equations that can be obtained by varying i and $\mathbf{j}$ in the construction of $\boldsymbol{F}(\mathbf{j}, i)$. (Recall that $|\mathbf{j}| = p + 1$.) Each monomial of order p contributes $n + 1$ unknowns to the systems. While each monomial of order $p + 1$ contributes n equations to the system. Since the number of monmomials of order (p+1) is always at least n more than the number of monomials of order p, the system is formally overdetermined. In the event that the surface S is planar, our approximation is exact and the system is consistent. In the event that S is not planar there may still be a plane that passes through the edge intersections. Here again the system is consistent; the algorithm recovers this plane. However, whether the system is consistent or not, approximate or not, the column space of the associated matrix has full rank, and the system has a least squares solution which we find.

Still assuming that the right hand side is known, $\boldsymbol{n}^\theta$ may be approximated by taking $\boldsymbol{F}$ to be a constant vector field $(p = 0)$. In this case the linear system is exactly determined and we solve it exactly. Therefore the algorithm will be complete provided the right hand sides are known. However, the right hand sides are moments over $n - 1$ dimensional cubes sliced by codimension 1 surfaces. Hence after $(n - 2)$ further iterations of the algorithm we may express the right hand sides as $(p + n - 1)$th moments over a set of one dimensional line segments, which may be conveniently calculated directly.

3 API

3.1 Design

`Workshop` is a class which inherits from `GeometryService` and implements the `GeometryService` interface using the workshop algorithm. `Workshop` contains a `BaseLocalGeometry` class. This base class is an interface which encapsulates the steps to the workshop algorithm of reducing a surface that is locally defined as a function to an embedded boundary description. The `Workshop` class is defined by its only constructor.

- `Workshop(const BaseLocalGeometry& localGeometry)`

  Define the Workshop with the input local geometry description.

A BaseLocalGeometry-derived class must be able to answer the following questions:

- Is the box in question regular, or covered (or neither)? A regular box in this context is a cell which is entirely outside the covered region. A covered box is entirely inside the covered region.
- If the cell is irregular, which signed coordinate direction is "up" in the cell? The up direction corresponds to the coordinate direction that corresponds to the largest component of the normal to the boundary at this cell.
- For an irregular cell, given the independent variables, return the dependent variable. Consider again figure **??**. In this example, the y direction is the "up" direction and the workshop-derived class must be able to return values of y in the cell given x and z.

Given this, the BaseLocalGeometry class has the following pure virtual functions in its interface:

- ```
  virtual bool isRegular(const Box& region, const Box& domain,
                         const RealVect& origin, const Real& dx)
                         const =0;
  virtual bool isCovered(const Box& region, const Box& domain,
                         const RealVect& origin, const Real& dx)
                         const =0;
  ```

  Return true if *every* cell in the input region is regular or covered. Argument region is the subset of the domain. The domain argument specifies is the span of the solution index space. The origin argument specifies the location of the lower-left corner (the zero node) of the solution domain and the dx argument specifies the grid spacing.

- ```
  virtual
  int upDirection(const IntVect& iv,
                  const Box& domain,
                  const RealVect& a_origin,
                  const Real& a_dx)  const = 0;
  ```

  This returns the signed integer which most closely represents the normal direction of the interface at an irregular cell (which coordinate direction has the largest normal component). This will only be called if the cell is irregular.

- ```
  virtual
  Real localFuncValue(const RealVect& independentCoords,
                      const int& upDirection,
                      const IntVect& a_iv,
                      const Box& a_domain,
                      const RealVect& a_origin,
  ```
  ```

```
const Real& a_dx) const = 0;
```

Return the value at the dependent coordinate given the independent coordinates. The `upDirection` argument defines which coordinate is the dependent one.

- `virtual`
  `BaseLocalGeometry* new_baseLocalGeometry() const = 0;`

  Return a newly allocated derived class. The responsibility for deleting the memory is left to the calling function.

3.2 Example

The `Workshop` class has the following usage pattern. The local geometry description is defined and used to define the `Workshop` class. The `Workshop` is then used to define the global `EBIndexSpace`.

```
class MyGeometry: public BaseLocalGeometry
{
....
};

void defineMyEBIS(const Box& domain,
                  const RealVect& origin,
                  const Real& dx)
{
  MyGeometry myLocalGeom;
  Workshop myWorkshop(myLocalGeom);
  EBIndexSpace* ebisPtr = Chombo_EBIS::instance();
  ebisPtr->define(domain, origin, dx, myWorkshop);
}
```

References

[1] Denis Gueyffier, Jie Li, Ali Nadim, Ruben Scardovelli, and Stephane Zaleski. Volume-of-fluid interface tracking with smooth surface stress methods for three dimensional flows. *J. Comput. Phys.*, 152:423–456, 1999.