Smooth Interface Reconstruction from Volume Fraction Data Using Variational Techniques and Level Set Methods

by

Michaeel Majeed Kazi

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor James A. Sethian, Chair
Professor John Strain
Professor Ömer Savaş

Spring 2013

Smooth Interface Reconstruction from Volume Fraction Data Using Variational Techniques and Level Set Methods

Abstract

Smooth Interface Reconstruction from Volume Fraction Data Using Variational Techniques and Level Set Methods

by

Michaeel Majeed Kazi

Doctor of Philosophy in Mathematics

University of California, Berkeley

Professor James A. Sethian, Chair

The volume fraction data structure describes the location of a given material within space. It partitions a domain into a set of discrete cells, and for each cell stores the ratio of material contained within it to the total volume of the cell. The task of transforming a grid of volume fractions into a surface representation of the boundary is called the Material Interface Reconstruction Problem. We investigate a variational formulation to this problem, implemented via a level set method, that produces a surface representation of the boundary. We start with an initial guess and iteratively refine the surface by computing a surface-area minimizing curvature flow, followed by an approximate L^2 projection of that flow onto a volume-preserving space. We find that the method yields satisfactory results for both two-phase and multiphase data. In the cases where there are C^0 but not C^1 boundaries between only two materials, the exact solution produces oscillations with period equal to the volume fraction grid size, and amplitude exponentially decreasing away from the location of the loss of smoothness. To reduce oscillations, an alternative method is proposed where in place of the volume fraction constraint, we minimize a weighted sum of the total surface area, plus the sum of squared error in reconstruction. We discuss the results and compare with the projection algorithm.

To Valentina

Contents

Acknowledgments

I want to thank James Sethian for his generous support, guidance, understanding, and nurturing of a great environment (LBNL Mathematics Group) to not only work but to spend enjoyable days. I also want to thank Alexandre Chorin, who first set up this group.

I couldn't have done this without Robert Saye or Jeff Donatelli, who provided valuable discussions of numerical methods, mathematical techniques, and being great friends. Thanks also to the members of the LBNL Mathematics group, who I've had many fun and/or enlightening conversations with over lunch or just around the office: Trevor Potter, Jue Chen, Bradley Froehle, Danielle Maddix, Ben Preskill, August Johansson, Chris Rycroft, and Matti Morzfeld.

I want to thank my professors: Jamie Sethian, again, for an excellent and entertaining Numerical PDE's class, and for not going easy on me; John Strain, whose numerical ODE's class taught me so much and reiterated to me how much I enjoyed applied math; Craig Evans, who introduced me to some pretty cool mathematical tools; Ömer Savaş, a humble and masterful fluid mechanician; Grigory Barenblatt, who entertained us day after day with intellectual conversation and tea.

Special thanks to Ken Joy at UC Davis, who, through a conversation with Jamie, started this project.

Thanks to Karen for her support and understanding, and getting me to stop browsing the internet and get to work! She provides me with a reason to put all of my knowledge to use.

And finally, thanks to my parents for always being there as sounding boards, and helping come up with solutions to life's other dilemmas.

Chapter 1

Introduction

The goal of this thesis is to propose a new algorithm for the Volume Fraction Reconstruction problem. In this chapter, we first define the volume fraction data structure. Then, we define the reconstruction problem, and briefly discuss several existing techniques. Finally, we propose the new method and outline the structure of the remaining chapters.

1.1 Volume fractions

A volume fraction grid is a data structure that describes the location of some material within a domain. The domain is partitioned into discrete cells, and each cell is given a value $v, 0 \leq v \leq 1$, representing the proportion of the cell filled by material. For example, an empty cell has value 0, a cell completely filled has value 1, and a cell partially filled has the appropriate fractional value. Figure 1.1 shows an example of a circular material described by a 5-by-5 volume fraction grid.

This structure was first introduced by Noh and Woodward [16] in an algorithm to simulate advection of a body of fluid. Their technique had the advantages of conserving volume, automatically handling interface topology changes, and avoiding problems associated with moving meshes. Many variations and improvements to the core method have been developed, such as methods by Chorin [9], Hirt and Nichols [12], and Puckett [22].

The name “Material Interface Reconstruction” refers to the problem of translating volume fraction grid data into a representation of the material boundary (e.g. a mesh). It is valuable in the visualization of simulation results, as well as for providing one possible way of computing geometric properties of the interface itself, such as normal vectors and principal curvatures.

The Material Interface Reconstruction problem is not well-posed: many possible regions can have the same volumes. In Figure 1.2, we have shown three qualitatively different regions that all yield the same 2×2 volume fraction grid. The first solution is a smooth curve, the second is piecewise linear, and the third is made up of horizontal or vertical lines only. Due to

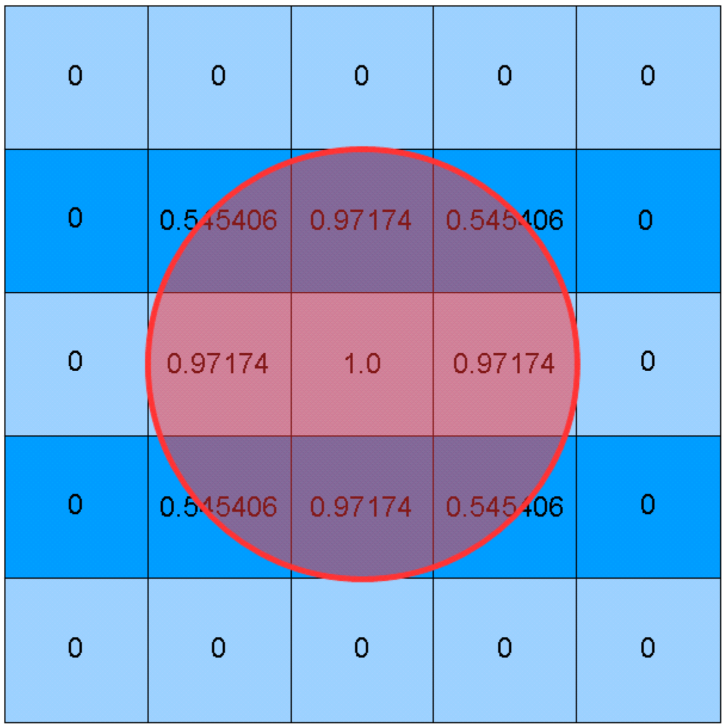


Figure 1.1: Example volume fractions corresponding to a circle.

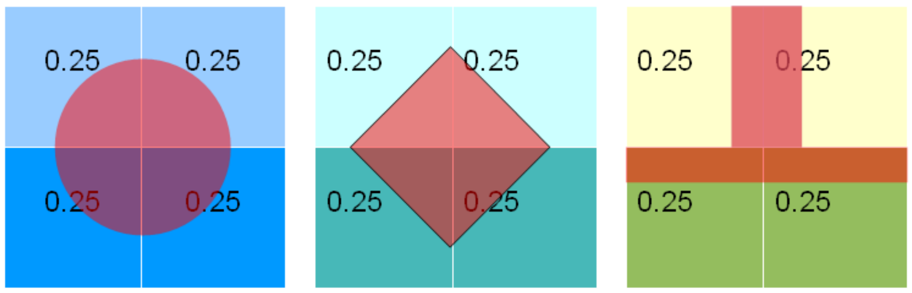


Figure 1.2: Different reconstructions of the 2-by-2 grid filled with the value $\frac{1}{4}$

the many possible interfaces satisfying the volume requirements, there are several numerical methods available to perform the task, each defining the surface in a different way.

1.2 Previous methods

We now briefly examine three techniques: the Simple Line Interface Calculation (SLIC) method, the Piecewise Linear Interface Calculation (PLIC) method, and the Active Interface Reconstruction method. The first two methods are non-iterative. For each cell, they directly compute the shape of the interface using data in a small number of neighboring cells. The third method iteratively refines a mesh of the interface, one vertex at a time. The interface within a single cell is thus affected by the structure of the entire interface, due to coupling. In the rest of this chapter, we limit our discussions to two dimensions, but the ideas are applicable to any number of dimensions.

1.2.1 Simple Line Interface Calculation

The original reconstruction algorithm, Simple Line Interface Calculation, or SLIC (Noh and Woodward [16]), is a basic way to reconstruct an interface from volume fractions. Its original goal was to calculate the advection of a fluid front as represented by the volume fraction data.

SLIC reconstructs an interface based on one-dimensional sweeps. It examines a 3x1 or 1x3 window about each cell, and based on a lookup table of which values of these cells are 0, 1, or between 0 and 1, it determines whether to place a horizontal or vertical line (or neither). The location of the line segment is then determined by the volume fraction. The advection algorithm works by using the x-component of the fluid velocity and a 3x1 window to compute the amount of fluid to move left or right, then repeating the calculation with the y-component and a 1x3 window to determine the amount to move up or down.

SLIC is very fast, but the interface is always either horizontal or vertical, causing the reconstruction to look blocky. Other methods are able to represent the interface in more detail.

1.2.2 Piecewise Linear Interface Calculations

The Piecewise Linear Interface Calculation (PLIC) approach refers to a collection of methods that construct the interface in each cell with a single straight line segment of arbitrary orientation. The unknown in each cell is the normal $\hat{n}$, or equivalently in 2D, the slope m. Once the normal is determined, the location of the line segment is computed based on the volume fraction. We discuss two different approaches to calculating $\hat{n}$ for a given cell: (1) by computing finite differences on the volume fraction grid, and (2) by solving a least squares problem on a 3×3 window about the cell in question.

Finite Difference approximations

The slope m may be obtained by finite differences on the volume fraction grid. Hirt and Nichols [12] used the centered difference formula

$$m = \frac{1}{2}(v_{i+1,j+1} - v_{i-1,j+1} + v_{i+1,j} - v_{i-1,j} + v_{i+1,j-1} - v_{i-1,j-1}),$$

while Parker and Young [18] calculate the normal as the gradient of the volume fraction grid, (v_x, v_y), which they determine by the formulas

$$v_x = \frac{1}{4+2\alpha}(v_{i+1,j+1} + \alpha v_{i+1,j} + v_{i+1,j-1} - v_{i-1,j+1} - \alpha v_{i-1,j} - v_{i-1,j-1})$$

$$v_y = \frac{1}{4+2\alpha}(v_{i+1,j+1} + \alpha v_{i,j+1} + v_{i-1,j+1} - v_{i+1,j-1} - \alpha v_{i,j-1} - v_{i-1,j-1}),$$

where α is a free parameter. The authors recommend $\alpha = 2$.

Least Squares Reconstruction

The method "Least squares Volume-of-fluid Interface Reconstruction Algorithm" (LVIRA) is due to Puckett [22], and determines the normal $\hat{n}$ of cell ij as follows. The material interface within the 3×3 window about ij (from $i-1$ to $i+1$, $j-1$ to $j+1$) is approximated by a straight line. The material is located on one side of this line: define $\hat{n}$ such that it points away from the material region. Let $\tilde{v}$ represent the volumes occupied by the material bounded by this line. By requiring $\tilde{v}_{ij} = v_{ij}$, there exists only one such line corresponding to each $\hat{n}$. Then the optimal value of $\hat{n}$ is chosen to minimize

$$\sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} (v_{i'j'} - \tilde{v}_{i'j'})^2$$

The advantage to this technique is that if the volume fraction data can be fit to a straight line, then LVIRA will reproduce the line exactly. This is guaranteed by construction, and is not true of the other PLIC methods discussed so far. Consequently, the algorithm reconstructs with second order accuracy when applied to smooth shapes.

To be more precise, we define second order accuracy as follows. Consider any region Ω such that $\partial\Omega$ is C^1. Define V_h as the volume fraction grid corresponding to Ω where each cell measures h on a side. A reconstruction algorithm applied to V_h produces Ω'. Finally, define the region $XOR(\Omega, \Omega')$ as

$$\text{XOR}(\Omega, \Omega') = \{x : x \in \Omega, x \notin \Omega' \text{ or } x \notin \Omega, x \in \Omega'\}$$

We will define a method to be **second order** if the volume of the region $XOR(\Omega, \Omega')$ decreases as h^2 as $h \to 0$.

A modification to this method exists, called ELVIRA (Efficient LVIRA), due to Pilliod [20], which chooses among only six possiblilities for m to minimize – specifically, ones given by finite differences on the grid. This, too, is able to reconstruct straight lines (see Pilliod and Puckett [21] for an in-depth comparison of PLIC methods).

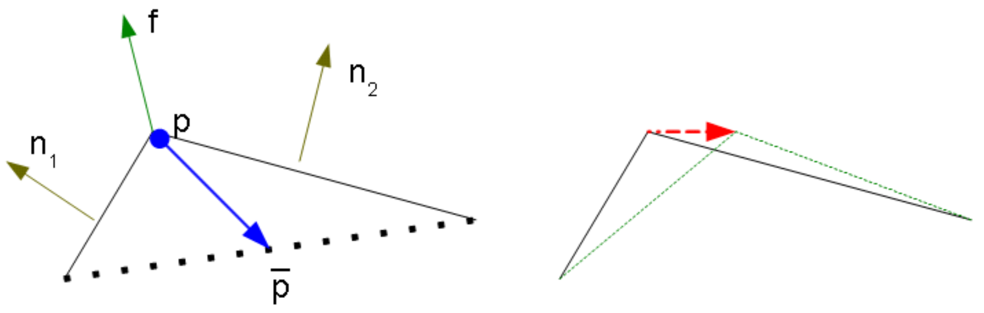


Figure 1.3: An illustration of the force vectors involved in the Active Interface Reconstruction algorithm from 1.4. On the left, the blue arrow is the smoothing vector, while the green is the volume correction vector. On the right, the red is the resultant, and the new interface is drawn in green.

The advantages of SLIC and PLIC methods are speed and simplicity. They all look only to information around a small neighborhood of each grid cell (3×3 or 3×1), and often do not require many lines of code. The disadvantage of these methods is that they do not attempt to match segments from neighboring cells. The normal vectors between cells are not continuous.

1.2.3 Active Interface Reconstruction

We use the name "Active Interface Reconstruction" to refer to the method of J. C. Anderson et al. [3] Their algorithm first chooses an interface topology and initial mesh based on a sophisticated algorithm. It iteratively refines the mesh based on two forces: (1) smoothing, and (2) volume correction. At each step, a mesh point is chosen at random, and the two forces are applied as follows.

For a point p, let $\bar{p}$ be the average of vertices sharing an edge with p. Then the smoothing force is $\bar{p} - p$. In Figure 1.3, this is the blue vector, which links the mesh point p to the midpoint of the line segment connecting p's neighbors. The volume correction force vector, f, is computed as the average of the normals of all faces containing p. In Figure 1.3, this is the green arrow, which is the sum of the yellow normal vectors. The vector f is then scaled by the current volume fraction error for the cell in which p resides. These two forces are scaled by different parameters α and β, chosen by the user, and added together to yield the final displacement for point p. After each iteration, the values of α and β are decreased simultaneously until convergence in a manner similar to an annealing process [3].

1.3 Proposed algorithm

In this thesis, our approach is based on finding the minimum surface area interface that encloses the prescribed volume fraction within each grid cell. There are three advantages to this choice:

- The resulting interface is C^1.
- This choice models the effects of surface tension at scales of one grid cell or smaller. Therefore, we expect qualitative similarity between the reconstructed and exact interfaces if the volume fraction values come from fluid interface simulations.
- Surface area is reduced via curvature flow, which is conveniently represented in a level set framework.

The rest of this thesis is organized to discuss this formulation and an algorithm for computing it. In Chapter 2, we review Level Set methods, since they will be used by our algorithm. In Chapter 3, we define our solution for surfaces in $\mathbb{R}^d$ $(d = 2, 3)$ defined by $(\mathbf{x}, f(\mathbf{x}))$, where $x \in \mathbb{R}^{d-1}$, and $f : \mathbb{R}^{d-1} \to \mathbb{R}$ is a height function, and derive a gradient descent algorithm with an L^2 projection to handle the constraints. In Chapter 4, we extend the ideas of Chapter 3 to surfaces defined implicitly by $\phi = 0$, where $\phi : \mathbb{R}^d \to \mathbb{R}$ is a level set function. We also discuss our algorithm as applied to multiple-phase volume fraction surface reconstruction. In Chapter 5, we show results of our algorithms in 2D and 3D. In Chapter 6, we conclude and discuss further improvements.

Chapter 2

Level Set Methods

In this chapter, we introduce the Level Set method for tracking a moving interface in $\mathbb{R}^n$. The outline is as follows.

1. We formulate the Level Set Method[17] and derive the main equation.
2. We discuss the upwinding scheme used to advance the solution, and the Narrow Band technique [6] which reduces the computational complexity of the method.
3. We discuss how to reinitialize the level set function to a signed distance function via the Fast Marching Method [25].
4. Finally, we discuss how to maintain the signed distance function by using extension velocities [2].

2.1 Formulation

The Level Set Method, first introduced by Osher and Sethian [17], works as follows. Let $\phi(x,t) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ be a function where $\Gamma(t)$, the interface, is defined implicitly by $\Gamma(t) = \{x : \phi(x,t) = 0, x \in \mathbb{R}^n\}$. If we track some point $y = y(t) \in \Gamma(t)$ on the interface, then

$$0 = \frac{d}{dt}\phi(y(t),t) = \phi_t(y,t) + \nabla\phi(y,t) \cdot \mathbf{F}$$

where $\mathbf{F}$ is the velocity of the interface point. The normal direction to the level sets of ϕ is given by $\frac{\nabla\phi}{|\nabla\phi|}$. If we assume, for speed $F \in \mathbb{R}$, that $\mathbf{F} = F\frac{\nabla\phi}{|\nabla\phi|}$, then

$$\phi_t + F|\nabla\phi| = 0 \tag{2.1}$$

which is the level set equation. One choice for F is $F_n - \epsilon\kappa$, where $F_n \in \mathbb{R}$ is a scalar expansion or contraction component, and κ is the curvature of the interface. Expressed in terms of ϕ, we have that $\kappa = \operatorname{div}\hat{n} = \operatorname{div}\frac{\nabla\phi}{|\nabla\phi|}$.

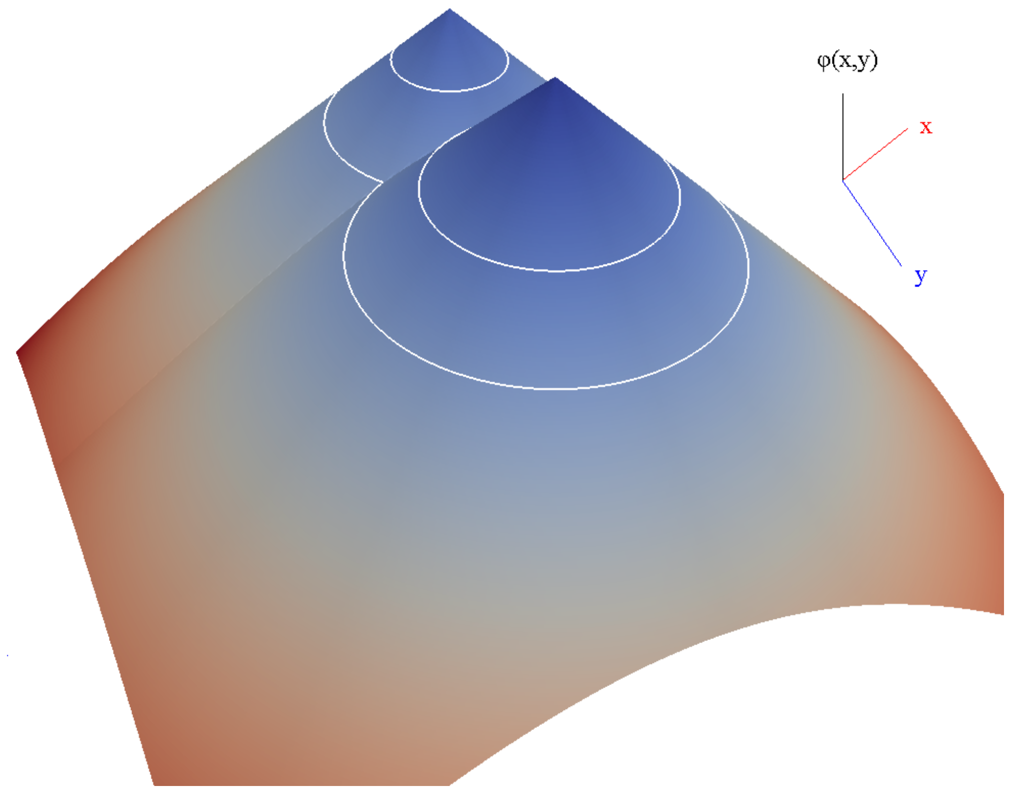


Figure 2.1: Level set representation embeds a curve as slices of this height map. If the curve evolves under $F = 1$, then the level sets correspond to the curve at different times t. Two slices of differing topology and smoothness are indicated.

2.2 Upwinding and stencils

Numerically computing $|\nabla\phi|$ in equation 2.1 must be done carefully. Centered differences introduce smearing, while one-sided differences need to account for the direction of travel of the level set. Since the PDE involves transport along the direction of the level sets of ϕ, the direction of information flow is crucial. Choosing the wrong points for the stencil will violate the CFL condition and result in failure to solve the PDE.

Osher and Sethian produced an upwind one-sided stencil to approximate the PDE and perform the correct behavior near shocks. Following [17] and writing the level set equation in the form

$$\phi_t + F|\nabla\phi| - \epsilon\kappa|\nabla\phi| = 0$$

we can approximate $(F|\nabla\phi|)_{ij}$ by $\max(F_{ij}, 0)\nabla^+_{ij} + \min(F_{ij}, 0)\nabla^-_{ij}$, where

$$\begin{aligned}
\nabla^+_{ij} &= \sqrt{\max(D^{-x}\phi_{ij},0)^2 + \min(D^{+x}\phi_{ij},0)^2 + \max(D^{-y}\phi_{ij},0)^2 + \min(D^{+y}\phi_{ij},0)^2} \\
\nabla^-_{ij} &= \sqrt{\min(D^{-x}\phi_{ij},0)^2 + \max(D^{+x}\phi_{ij},0)^2 + \min(D^{-y}\phi_{ij},0)^2 + \max(D^{+y}\phi_{ij},0)^2}
\end{aligned} \tag{2.2}$$

Here, $D^{-x}\phi_{ij} = \frac{\phi_{i,j}-\phi_{i-1,j}}{h}$ is the first order, left hand side x-derivative finite difference formula. The other first order, one-sided derivatives are denoted by D^{+x}, D^{-y}, and D^{+y}.

Higher order accuracy can be achieved by more accurately approximating each one-sided derivative using essentially non-oscillatory (ENO) interpolation. More precisely[17], replacing D^{-x} in equation 2.2:

$$D^{-x} + \frac{h}{2}m(D^{-x-x}, D^{-x+x})$$
$$\text{where } m(a,b) = \min(a,b) \text{ if } ab \geq 0, 0 \text{ otherwise}$$

Curvature flow, on the other hand, acts to smooth the interface and propagates in all directions, and is not upwinded. We approximate curvature flow via its expanded definition

$$\kappa = \frac{(\phi_{xx}+\phi_{yy})\phi_z^2 + (\phi_{xx}+\phi_{zz})\phi_y^2 + (\phi_{yy}+\phi_{zz})\phi_x^2 - 2\phi_x\phi_y\phi_{xy} - 2\phi_x\phi_z\phi_{xz} - 2\phi_y\phi_z\phi_{yz}}{(\phi_x^2+\phi_y^2+\phi_z^2)^{\frac{3}{2}}}$$

and use centered differences to approximate all derivatives of ϕ.

2.2.1 Narrow band

The Level Set function is one dimension larger than the surface it represents. For a three-dimensional problem on an equally spaced grid with N grid points per side, updating the entire level set function takes $O(N^3)$. However, we can reduce the complexity to $O(N^2)$ by making the crucial observation that computations involving the zero level set do not depend on faraway points. Indeed, we need only to have enough grid points surrounding the interface to smoothly represent the level set function's derivatives at the interface. This is accomplished by dividing the domain into two regions: a narrow band, and the outside. The narrow band contains grid points up to a certain distance away from the interface, and the outside contains everything else. The points on the outside are fixed, while the narrow band points are updated according to the level set equation. Chopp [6] introduced this time-saving technique on his work on computing minimal surfaces, while Adalsteinsson and Sethian [1] analyzed the method in depth to determine its accuracy, and to suggest optimal sizes for the narrow band. The authors suggest six grid points on each side of the zero level set.

As the interface moves, it may eventually become close to the edge of the narrow band. When this happens, a new band will need to be constructed, and the function ϕ will need to be reinitialized within the new domain. We reinitialize ϕ to be a signed distance function using the Fast Marching Method, as discussed in the next section.

2.3 Reinitialization and the Fast Marching Method

If the level sets of ϕ become too close together or too far apart, the accuracy of numerical solutions to our PDE (2.1) diminishes. To fix this, we **reinitialize**, which means that we build a new function ϕ that leaves the zero contour intact, but ensure that ϕ is a signed distance function. A **signed distance function** is a function ϕ such that the value of $|\phi(x)|$ is the distance to the set $\{\phi(x) = 0\}$, and $\phi(x)$ is positive on one side of the zero level set, and negative on the other. We adopt the convention that $\phi(x) < 0$ on the interior of our region. A signed distance function solves the PDE

$$|\nabla\phi| = 1.$$

This, coupled with the boundary condition $\phi(x) = 0$ for $x \in \Gamma$, where Γ is the previous zero contour, becomes a boundary-value problem . This is solved numerically through the approximation

Using the stencil (2.2), our approximation to the PDE $|\nabla\phi| = 1$ is given by

$$1 = (\nabla^+_{ij}\phi)^2 = \max(D^{-x}\phi_{ij}, 0)^2 + \min(D^{+x}\phi_{ij}, 0)^2 + \max(D^{-y}\phi_{ij}, 0)^2 + \min(D^{+y}\phi_{ij}, 0)^2 \quad (2.3)$$

To code the boundary conditions, since the surface will not align with the grid, we fix the values of ϕ_{ij} within one grid point of the interface. Their values are the distances to the interface computed directly (see Section 2.4.3).

We must solve equation (2.3) for all grid points simultaneously. A computationally optimal method is described in the next section.

2.3.1 The Fast Marching Method

The crucial observation with these equations is that information flows **away** from each contour (in increasing absolute value of ϕ). This is evident both from (1) the PDE, that the values of ϕ increase as the distance from the source interface $\phi = 0$; and (2) the numerics, where we can see that derivative information is used only if ϕ_{ij} is larger than the other points in the stencil.

We can use this fact to order our points, to build our solution outward from the interface. Sethian introduced the Fast Marching Method [25] which works as follows:

1. Separate grid points into three groups: **known**, **tentative**, and **unknown**.
2. Place grid points within one unit of the boundary, with their correct distances, into **known**.
3. Remove the smallest valued node ij from **tentative** and add it to **known**.

4. For each point $i'j'$ adjacent to ij, if it is in either **tentative** or **unknown**, compute $\phi_{i'j'}$ using the fact that $\phi_{i'j'} > \phi_{ij}$, and the assumption that any $\phi_{i''j''} > \phi_{i'j'}$ for $i''j''$ not in **known**.

5. Add $i'j'$ to the **tentative** group. If it is already there, update the value only if a smaller value was computed.

6. Repeat from (3) until all nodes have been processed.

If the assumption in step 4 is not true ($\phi_{i''j''} < \phi_{i'j'}$), then $\phi_{i''j''}$ will eventually be accepted first, and the **tentative** value of $\phi_{i'j'}$ will be recomputed. Thus, the algorithm correctly computes the stencil from equation 2.2.

Each grid point is accepted only once, and its values do not change afterwards. Each point will be visited no more times than the number of neighbors it has. The efficiency of this method rests on the ability to pick the smallest-valued node from **tentative**. This is accomplished by means of a priority queue or heap data structure.

2.3.2 Heaps

A **min-heap** is a binary tree data structure satisfying two properties. (1) The value at each node is smaller than either of its children, and (2) the tree is complete. This means, at each depth of the tree, all nodes that can exist do exist, with the exception of the bottom layer, which fills up left-to-right.

A more illuminating description comes from indexing the nodes from 1 to n, and defining node i's children to be nodes $2i$ and $2i + 1$. Criteria (2) means that there are no holes in this list, each index corresponds to an existing node.

Criteria (1) is the property we will exploit to make the priority queue, and criteria (2) prevents degenerate, inefficient tree structures.

To use this heap as a priority queue we need to define only two operations: **insert** and **remove**. In more detail:

Insert. To add a node to the heap of size n, we place it at position $n + 1$. We then compare its value to its parent. If it is smaller, we swap values. If not, we are done. If a swap occurs, this process repeats. This "bubbles" up the new node into its correct position in the tree. For example, were we to insert the value 2 into Figure 2.2, it would start as the right child of 7, then bubble up, moving the 7 into its former place and the 3 into the 7's former place, and finally staying to the left of the root node.

Remove. To remove a node, after we make a note of its contents, we insert the node at position n into position 1 (and decrement n). Then, starting at the first node, we "bubble" the node downwards. Specifically, if any of the child nodes are smaller, we take the smallest of those children and swap it with the node in question, and repeat. For example, when we remove the root node from Figure 2.2, the 10 is placed at the top, and then it bubbles down. The 3 is moved to the root, then the 4 into the 3's former spot, and then the 5 where the 4 used to be, and then the 10 finally resting where the 5 used to be.

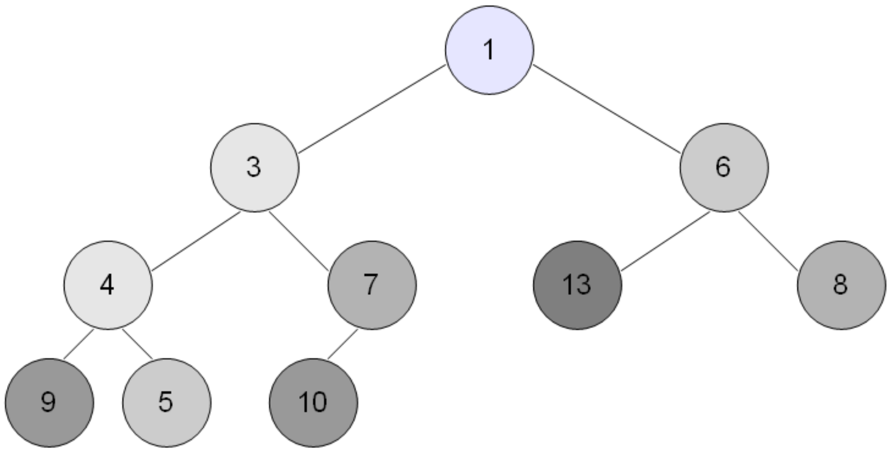


Figure 2.2: Example of a min-heap.

Each operation completes in $\log(n)$ time. Thus, using a heap for the Fast Marching method yields a computational complexity of $O(N\log(N))$, where N is the total number of nodes to reinitialize. In practice, the complexity is lowered to $O(N\log(N'))$, where N' represents the number of nodes necessary to represent a single level set.

2.3.3 Nearby distances for the Fast Marching Method

To obtain nearby values of distances to the zero level set, we use Chopp's method [7]. This gives ϕ the interpretation of a piecewise tricubic polynomial, and we compute distances to its zero level set exactly. For each $1 \times 1 \times 1$ grid cell containing the interface, we obtain the coefficients a_{ijk} for the interpolant

$$\phi \approx f(x,y,z) = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} a_{ijk}x^i y^j z^k$$

by solving a linear system, where values of $\phi, \phi_x, \phi_y, \phi_z, \phi_{xy}, \phi_{xz}, \phi_{yz}, \phi_{xyz}$ on the 8 points of the cube are prescribed. We use centered difference approximations of the derivatives, making the total stencil 4^3 points.

For each of the 8 vertices on the $1 \times 1 \times 1$ cell, the closest point where $f(x,y,z) = 0$ is solved via a modified Newton iteration. Since each vertex v is also a member of 8 other cells, each cell may provide its own closest point to v for its local tricubic interpolant. We choose the point closest overall to v.

2.4 Extension velocities

In a Lagrangian formulation, the speed law F need only be defined at the interface. However, in the level set equation (2.1), F needs to be defined everywhere in the Eulerian narrow band. In some approaches where the level set represents a moving fluid interface, the fluid velocity can be used directly (as done by Rhee, Talbot, and Sethian [23]). In places where the problem does not naturally yield an extension velocity, Malladi, Sethian, and Vemuri proposed using the velocity at the closest point to the interface [14]. For other approaches, see Sethian and Strain [27], or Chen, Merriman, and Osher [5].

Adalsteinsson and Sethian [2] derived a way to compute extension velocities that preserves ϕ as a signed distance function. This is the algorithm we use in this work. In this approach, F_{ext} is defined to be constant along the characteristics of the equation $|\nabla\phi| = 1$. Constancy of F_{ext} along characteristics of $|\nabla\phi| = 1$ means that F_{ext} must solve the PDE

$$\begin{cases} 0 = \nabla F_{\text{ext}} \cdot \nabla\phi \\ F_{\text{ext}}|_{\phi=0} = F \end{cases} \tag{2.4}$$

To solve this equation, one may modify the Fast Marching Method (FMM) to also propagate velocity information. The FMM follows the characteristics of the signed distance PDE out from the zero level set. In computing distance, the value along the characteristic changes with the length traveled along the characteristic. Equation 2.4 requires that the value of the velocity is constant.

The ordering involved in constructing the solution to $\nabla F \cdot \nabla\phi = 0$ is inherited from the Fast Marching Method. As an example, suppose the max and min switches in stencil 2.2 chose grid points $\phi_{ij}, \phi_{i'j}$, and $\phi_{ij'}$. Then the constraint $0 = \nabla F_{\text{ext}} \cdot \nabla\phi$ becomes

$$0 = \left(\frac{(\phi_{ij} - \phi_{i'j})(i-i')}{h}, \frac{(\phi_{ij} - \phi_{ij'})(j-j')}{h} \right) \cdot \left(\frac{(v - F^{\text{ext}}_{i'j})(i-i')}{h}, \frac{(v - F^{\text{ext}}_{ij'})(j-j')}{h} \right)$$

Solving for v gives us F^{ext}_{ij}.

Extension velocities are used throughout our algorithm, since (1) they cut down the demand for reinitialization, which can alter the curvature of the interface, and (2) our primary speed law will only be defined at the interface, making it necessary to extend it throughout the narrow band.

Chapter 3

Variational Formulation and Height Function Interfaces

In this chapter, we discuss a simpler version of our problem. We seek the minimum length interface, but instead of arbitrary curves, we examine graphs of functions, and instead of volume fraction cells, we constrain the area under the curve within intervals. In this framework we discuss

- properties of the solution,
- a gradient descent algorithm for computing the solution,
- examples of the algorithm's output, and
- alternatives to length minimization.

3.1 Height function formulation

Given a partition $0 = x_0 < x_1 < \ldots < x_N = b$ of the interval $[0, b]$, and given $c_1 \ldots c_N$, our goal is to find a function $u \in \mathcal{L}^2$ that minimizes the length functional $L : \mathcal{L}^2[0, b] \to \mathbb{R}$

$$L(u) = \int_0^L \sqrt{1 + u_x^2}\, dx$$

subject to constraints that

$$g_i(u) := \int_{x_i}^{x_{i+1}} u(x)\, dx = c_i$$

for $i = 1 \ldots N$.

We use Lagrange multipliers to examine the exact solution of this constraint minimization problem. On the Hilbert space $\mathcal{L}^2$, for the function $L : \mathcal{L}^2 \to \mathbb{R}$ and constraint functions

$g_i : \mathcal{L}^2 \to \mathbb{R}$, the Lagrange multiplier result takes the following form [30]: For any test function $v \in \mathcal{L}^2$,

$$\begin{aligned}
0 &= \tfrac{d}{d\tau}\left(L(u+\tau v) - \sum_{i=1}^{N} \lambda_i g_i(u+\tau v)\right)\Bigg|_{\tau=0} \\
&= \tfrac{d}{d\tau}\left(L(u+\tau v) - \sum_{i=1}^{N} \lambda_i \int_{x_i}^{x_{i+1}} (u+\tau v)\,dx\right)\Bigg|_{\tau=0} \\
&= \int_0^L \frac{2(u_x+\tau v_x)v_x}{2\sqrt{1+(u_x+\tau v_x)^2}}\Bigg|_{\tau=0} dx - \sum_{i=1}^{N} \lambda_i \int_{x_i}^{x_{i+1}} v\,dx \\
&= -\int_0^L \left(\frac{u_x}{\sqrt{1+u_x^2}}\right)_x v\,dx - \sum_{i=1}^{N} \lambda_i \int_{x_i}^{x_{i+1}} v\,dx \\
&= \int_0^L -\kappa v dx - \sum_{i=1}^{N} \lambda_i \int_{x_i}^{x_{i+1}} v\,dx \\
&= \sum_{i=1}^{N} \int_{x_i}^{x_{i+1}} (-\kappa - \lambda_i) v\,dx
\end{aligned} \tag{3.1}$$

Since the above relation must hold for any $v \in \mathcal{L}^2$, we infer that the integrands must be zero, and u will have to satisfy

$$-\kappa - \lambda_i \equiv 0$$

in each of the cells i. In other words, the surface will have piecewise constant curvature.

3.2 Steepest descent algorithm

We now discuss a steepest descent algorithm to find u, the minimizer of L. At a high level, our approach takes the gradient of $L(u)$, projects it to a curve such that the constraint functions are unchanged, and then updates u by a small timestep times the result.

Define $A : \mathcal{L}^2 \to \mathbb{R}^N$ such that $(Au)_i = \int_{x_i}^{x_{i+1}} u(x)\ dx$ for $i = 1 \ldots N$. Note that A is a linear operator. Let $v \in \mathcal{L}^2$ be the direction of change to our function u. Fixing v, the rate of change of $L(u + \tau v)$ gives us the variational derivative [10]:

$$\frac{d}{d\tau} L(u+\tau v)\Bigg|_{\tau=0} = \int_0^b -\kappa v\,dx$$

where we omit the details, since the calculation is essentially the same as in 3.1.

We choose an update direction v (where $\|v\| = 1$) such that (a) v induces the steepest change in L while (b) maintaining the correct values of the g_i. This becomes the requirement that (a) the value of the functional $\int_0^b -\kappa v\,dx$ is negative and largest in magnitude and (b) that $Av = 0$.

Therefore we choose v to be the L^2 projection of κ to the space $Av = 0$, in other words, the minimum of $\|v - \kappa\|^2$ where $Av = 0$. Supposing without loss of generality that $v = \kappa - w$, we find the minimum of $\|w\|^2$ such that $Aw = A\kappa$.

Fixing our domain in a discrete setting, where $u \in \mathbb{R}^n$, A becomes an $N \times n$ matrix. Finding w becomes an underdetermined least squares problem, solved by $w = A^\dagger A\kappa$, and so we obtain $v = \kappa - w = (I - A^\dagger A)\kappa$, where $A^\dagger$ is the pseudoinverse.

We now examine the pseudoinverse more closely. We can use Lagrange multipliers to minimize $\frac{1}{2}\|w\|^2$ subject to $Aw = A\kappa$:

$$w - \sum_{i=1}^{N} \lambda_i \nabla g_i(w) = w - A^T\lambda = 0$$

We deduce that w is a linear combination of the **area gradients** ∇g_i. With a discrete area matrix A, ∇g_i corresponds to the ith row of A. In the case of the trapezoidal rule on an equally spaced grid, w is a linear combination of the vectors $(\frac{1}{2}, 1, \ldots, 1, \frac{1}{2})$ on each interval.

To solve for λ we need to find the linear combination of the area gradients that gives the correct area. In other words, $AA^T\lambda = A\kappa$. $G = AA^T$ is a sparse matrix (tridiagonal in 1D) that determines the areas under the regions of interest that each gradient vector affects; and the solution is thus given by $\lambda = G^{-1}A\kappa$. This will be used again in further sections.

3.2.1 One-dimensional examples

The previous analysis did not depend on any specific choices of the matrix A and the discretization of u. To study an example, however, we make those values concrete. We discretize each interval into s equally spaced points (let $h = 1/s$), with the endpoints of each interval shared with its neighbors. Let $n = Ns$, the total number of grid points. Define A as the matrix of the trapezoidal rule applied to each interval $[x_i, x_{i+1}]$. For example, if $s = 3$,

$$A = \frac{1}{3}\begin{bmatrix} \frac{1}{2} & 1 & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 0 & \frac{1}{2} & 1 & 1 & \frac{1}{2} & 0 & \ldots \\ \ldots & & & & & & & & \end{bmatrix}$$

We compute $A^\dagger A$ by using LAPACK's SVD functionality. $A = U\Sigma V^T$, so $A^\dagger A = V\Sigma^\dagger\Sigma V^T$. Since it is clear that A has rank N (it computes area for each of the N intervals, which can take on any values), we can take the first N columns of V and call it $\tilde{V}$, so the projection becomes $\tilde{V}\tilde{V}^T$. For each iteration, we compute κ based on the three point centered difference stencil as in [26], then perform the projection to obtain v, and finally perform a simple update

$$u^{n+1} = u^n + vdt$$

and iterate until convergence.

Consider the function $f(x) = 1 - |x|$ on the periodic interval $[0, 2)$. We partition the interval into N equally spaced intervals, and set $c_i = \int_{x_i}^{x_{i+1}} f(x)\, dx$. We also initialize our

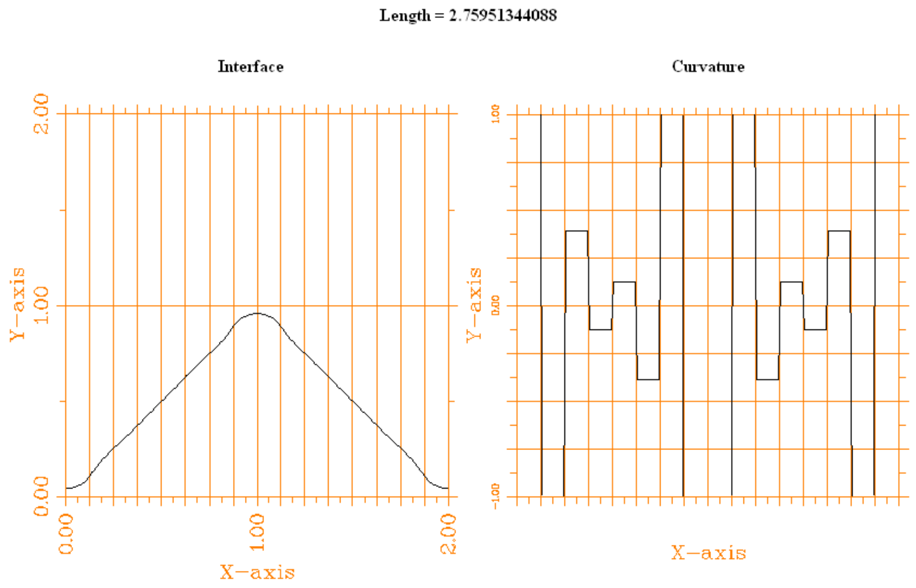


Figure 3.1: Result of computation, $N = 16, s = 10$. The curvature plot is shown at the right, matching with our theoretical expectations of a piecewise-constant-curvature solution.

solution vector to be exactly $u^0(x) = f(x) = 1 - |x|$, evaulated at the n discrete node points. We apply our gradient descent algorithm.

Our results from Figure 3.1 are consistent with a solution having piecewise constant curvature. Additionally, we note that under further refinement of the number of regions, N, the sides do converge to straight lines, as can be seen in Figure 3.2.

3.2.2 Two-dimensional height functions

The same algorithm can easily be applied for a function $u = u(x, y)$. Let n be the number of grid points in each direction, and N be the number of volume regions in each direction. We lay down a grid over the entire domain, and create the $N^2 \times n^2$ matrix which computes the volume underneath each region (for example, by the tensor product of the trapezoidal rule in each dimension). For each iteration,

- Compute the vector $\kappa = \text{div}\frac{(\nabla u, -1)}{|(\nabla u, -1)|}$
- Compute $v = (I - A^\dagger A)\kappa$

With the projection step in this form, $A^\dagger A$ is a dense matrix, which is quite costly to compute and evaluate. Instead, we will replace $A^\dagger A$ by a sparse version utilizing a local

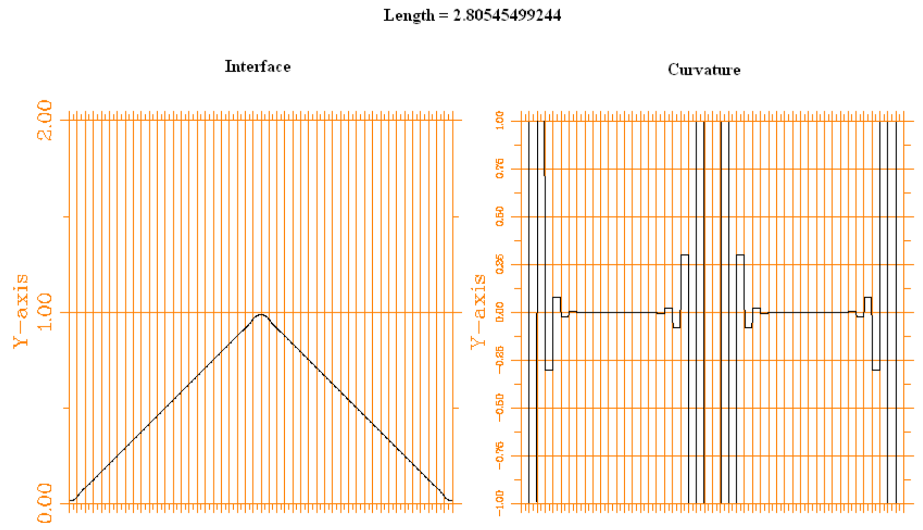


Figure 3.2: With $N = 48, s = 10$ volume cells, the curvature drops to near zero except for the first few cells near the bend.

inversion formula. Recall from earlier that our projected vector was a linear combination of the area gradient vectors. In 1D, we imagine a given volume fraction cell is the middle cell of a cluster of 3. Then our matrix $G = AA^T$ is only 3×3 and is readily invertible. Using the trapezoid rule, and writing the middle cell as the second component of a size 3 vector, our inversion formulas takes the form

$$\lambda_2 = \frac{1}{3h-4}(h, h-4, h) \cdot ((A\kappa)_1, (A\kappa)_2, (A\kappa)_3)$$

where we multiply each element of the vector with the area of κ in the middle cell.

For the entire grid, we then use the approximation for each cell individually:

$$\lambda_i \approx \frac{1}{3h-4}(h, h-4, h) \cdot (A\kappa)_{i-1:i+1}$$

For a 5 cell window, the approximation is

$$\lambda_i \approx \frac{1}{5h^2 - 20h + 16}(h^2, h^2 - 4h, h^2 - 12h + 16, h^2 - 4h, h^2) \cdot (A\kappa)_{i-2:i+2}$$

Computing the solution this way saves the cost of computing a dense $N \times N$ matrix inverse. In multiple dimensions it suffices to take the tensor product of these formulas with themselves.

We justify this approximation based on the structure of the matrix $G = AA^T$. We want to show that G^{-1} is concentrated on the diagonal, with elements falling off as h^r, where $r := \min(|j-i|, |N-j-i|)$ is the distance to the diagonal. We can write $G = I - \frac{h}{4}L$, where L is the negative of the discrete laplacian (tridiagonal of $(-1, 2, -1)$). Then

$$G^{-1} = \left(I - \frac{h}{4}L\right)^{-1} = \sum_{n=0}^{\infty}\left(\frac{h}{4}L\right)^n.$$

Now, as a tridiagonal matrix, we see that $(L^n)_{ij} = 0$ for $r > n$. Further, $\|L\|_1 = 4$, so we can bound the size of each nonzero entry of L^n by 4^n. Putting this together,

$$\begin{aligned}
\left|\left[\sum_{n=0}^{\infty}\left(\frac{h}{4}L\right)^n\right]_{i,i+r}\right| &= \left|\sum_{n=0}^{\infty}\left(\frac{h}{4}L\right)^n_{i,i+r}\right| \\
&= \left|\sum_{n=r}^{\infty}\left(\frac{h}{4}L\right)^n_{i,i+r}\right| \\
&\leq \sum_{n=r}^{\infty} h^r \left\|\frac{L}{4}\right\|_1^r \\
&\leq h^r\left(\tfrac{1}{1-h}\right).
\end{aligned}$$

Off-diagonal term fall off as h^r, so the faraway contributions are small, which justifies using our local inversion formulas.

Over many time steps, these small errors in approximating the projection will accumulate, but this is solved by either (1) applying the approximate projection more than once, or (2) letting the projection also correct for the error in the current solution u, i.e. $\lambda = (AA^T)^{-1}(A\kappa + e)$, $e = c - Au$.

As an example, we applied the local inversion technique in reconstructing a 125^2 height map of the San Francisco Bay bathymetry data set from USGS, with $s = 5$. The data is a two-dimensional grid with size 882×789. We extracted the subset of cells $[100-600, 0-500]$ and broke them into 4×4 clusters by averaging. We formed an initial approximation by projecting the constant zero function to a solution with the correct volumes. The final result completed in 18 seconds on an Intel Core2 Duo 3.0 GHz computer. See Figures 3.6 and 3.7.

3.3 Reconstruction from data derived from nonsmooth interfaces

We now examine what happens when the "true" solution to the minimization problem is not smooth. The function

$$f(x, y) = 2 - |1-x| - |1-y|,$$

represents a pyramid that has a corner (a location where both principal curvatures are singular), and four edges (locations where exactly one principal curvature is singular). The corner is located at $(x, y) = (1, 1)$, while the edges are located along the lines $x = 1$ and $y = 1$. We run the algorithm for the initial condition[1]

$$u_0 = f(x', y'), \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\pi/4) & -\sin(\pi/4) \\ \sin(\pi/4) & \cos(\pi/4) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.2}$$

The result of the algorithm, with $s = 5$, is shown in Figure 3.4. Oscillations appear throughout the edges. We see that they propagate differently in different coordinate directions. Indeed, having piecewise constant mean curvature still allows the principal curvatures to vary independently of each other. One approach to combat this will be discussed below.

3.4 Other possible functionals to minimize

It is clear from our formulation that length is only one of many possible "energy" functionals. The algorithms thus described in this chapter will work with any convex function L (to ensure the existence of some minimizer). To summarize, the algorithm

1. Finds the direction v such that $\frac{d}{d\tau} L(u + \tau v)|_{\tau=0}$ is negative and largest in magnitude, and
2. Computes $\lambda = (AA^T)^{-1} Av$
3. Advances $u^{n+1} = u^n + dt\,(v + A^T \lambda)$

As an alternative, we can choose our energy function L to simulate a surface under elastic equilibrium, $L(u) = \frac{1}{2} \int_D (\Delta u)^2 \, dx$. The variational derivative of L is

$$\begin{aligned} \frac{d}{d\tau} \int_D \left[\frac{1}{2} \Delta(u + \tau v) \right]^2 dx \bigg|_{\tau=0} &= \int_D [\Delta(u + \tau v)] \, \Delta v \, dx \bigg|_{\tau=0} \\ &= -\int_D \nabla(\Delta u) \cdot \nabla v \, dx \\ &= \int_D v \Delta\Delta u \, ds \end{aligned}$$

Therefore we choose $v = -\Delta\Delta u$ for step 1. In Figure 3.5, we run the input function 3.2 with the above descent direction. The oscillations are still present, due to the resulting equations, $\Delta\Delta u = \lambda_i$ for $x \in C_i$, having a piecewise constant right hand side. However, the oscillations are no longer as large or as pervasive.

[1] In this example, we have chosen to rotate the coordinate system to make sure that edges do not align with the coordinate system. When they align, the algorithm performs artificially well.

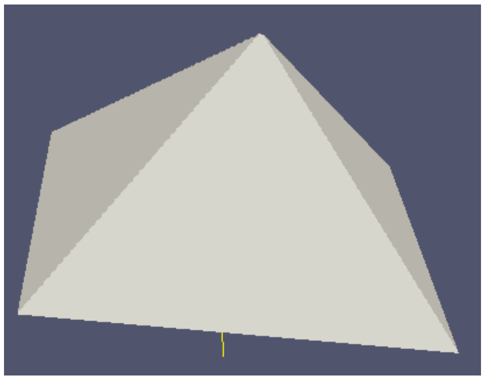


Figure 3.3: Initial shape $f(x) = 2 - |1 - x| - |1 - y|$ input into 2D height function example.

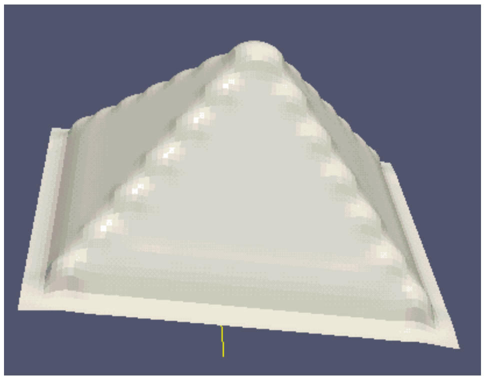


Figure 3.4: Result of surface area minimization. Notice the oscillations along the edges of the pyramid.

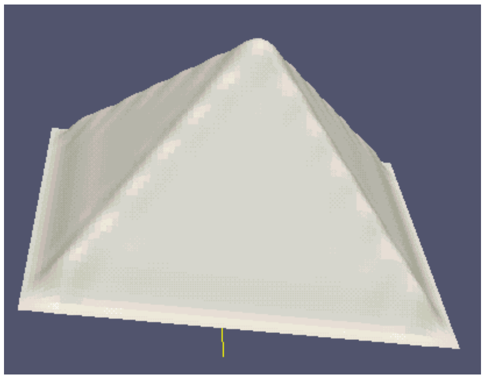


Figure 3.5: Result of elastic equilibrium. Notice it is smoother than surface area minimization.

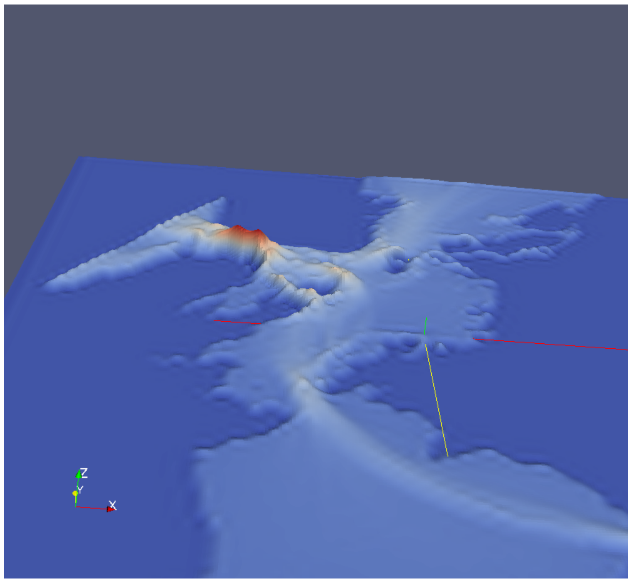


Figure 3.6: Result of elastic equilibrium on bathymetry data of the San Francisco Bay depth map. It took 18 seconds on a Core2 Duo 3.0 GHz, with grid size 125^2, and $s = 5$. See also Figure 3.7 for a close-up. The surface is colored by height, with dark blue being the lowest regions, and dark red the highest.

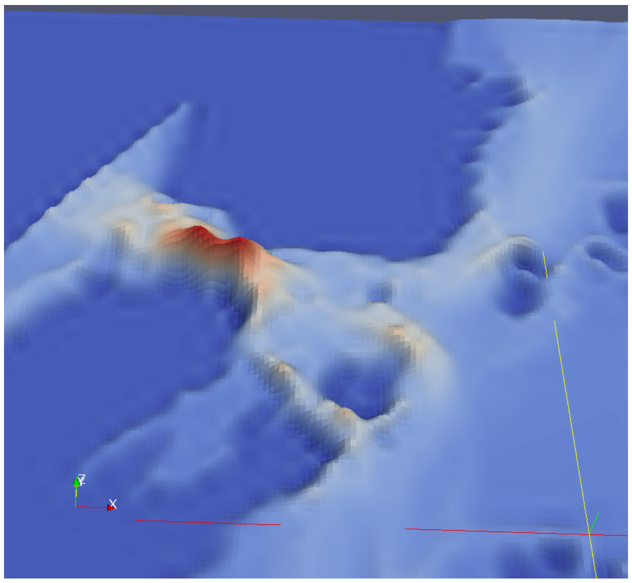


Figure 3.7: Zoom in of the same data as Figure 3.6.

Chapter 4

Full Algorithm

4.1 Outline

In this chapter we develop an algorithm for computing the solution to the surface area minimization problem, subject to volume fraction constraints, on an arbitrary rectangular volume fraction grid.

1. In Section 4.2, we discuss the level set representation of the interface, and discuss how it is discretized. Let Ω be the rectangular domain of the volume fraction grid. We embed the material region as $\{x : \phi(x) \leq 0\}$ where $\phi : \mathbb{R}^d \to \mathbb{R}$. The surface area and volume functionals are defined in terms of the level set function ϕ. In Section 4.2.1, we then discretize the level set function on a $s \times s$ or $s \times s \times s$ subgrid of the volume fraction grid. Our algorithm requires the evaluation of curvature, which is discretized based on a 3^2 or 3^3 stencil. In Section 4.2.2, we discretize the volume fraction functionals using the Marching Tetrahedrons algorithm.

2. The proposed algorithm works in two stages, (1) an approximate iteration based on a speed function defined by error and curvature terms, and (2) iterations by the projected steepest descent algorithm. In Section 4.3, we discuss the first stage of the algorithm. We provide an initial approximation to the interface by computing an isosurface of the volume fraction grid. The approximation is refined by advancing the level set equation by the speed law $F(x) = E(x) - \epsilon\kappa(x)$, where $E(x)$ is the reconstruction error of the cell containing x, κ is the curvature, and $\epsilon \in \mathbb{R}^+$.

3. In section 4.4, we discuss iterations of the algorithm's second stage, the projected gradient descent algorithm, analogous to that described in Chapter 3. These iterations are performed after Step 2 completes. One step of curvature flow is performed, and then we find the smallest additional perturbation to restore the volume fractions of the computed interface to the given values. To do this, we compute the linear combination of the gradients of each volume functional that cancels out the volume changes induced

by curvature flow. In Section 4.4.1, we discuss the overall shape of the volume gradients as $s \to \infty$, and why we cannot rely on that approximation. In Section 4.4.2, we discuss how to perform a projection on a nonuniform grid in one dimension. In Section 4.4.3, we discuss how to compute the gradients of the volume functionals, adapting the formulas from 4.4.2. In Section 4.4.4, we discuss how to compute the combination of volume gradients, and how to complete one iteration of the projection method.

In summary, the proposed algorithm represents the interface with a level set function ϕ, which is discretized onto a fine grid with subgrid parameter s. ϕ is initialized by interpolating the volume fraction grid onto the fine grid, and then subtracting a scalar so that the initial condition corresponds to an isosurface of the volume fraction data. The algorithm then performs several iterations of a first approximation based on the speed function $F(x) = E(x) - \epsilon\kappa$, where $E(x)$ is the error within the grid cell containing x. Once these approximations no longer improve the accuracy of the interface[1], the algorithm switches to the steepest descent with L^2 projection iteration, analogous to that described in Chapter 3. This iteration proceeds until some fixed time T^*, discussed in Section 5.2.2. This process is summarized by Figure 4.1.

4. In 4.5, we discuss an alternative, constraintless formulation based on optimizing a single objective function that adds epsilon times surface area, plus the sum of square errors in reconstruction. In Section 4.5.1, we formulate the objective function and compute its gradient. In Section 4.5.2, we discuss the relationship between its parameter, ϵ, and the L^1 reconstruction error.

5. Finally, in Section 4.6, we extend our algorithm to multiple phase reconstruction. In Section 4.6.1, we derive our method using multiple level set functions, running the projection algorithm for each material separately, and combining the level set functions via the Voronoi Interface (Saye and Sethian [24]). In 4.6.2, we define the Voronoi interface and how it is computed. In 4.6.3, we discuss how we compute the volume functionals exactly based on the resulting mesh.

4.2 Formulation

We embed our surface as the zero level-set of a higher dimensional function $\phi(\mathbf{x})$ with $\mathbf{x} \in \Omega \subset \mathbb{R}^d$, Ω our rectangular domain.

The surface area functional, L, is

$$L(\phi) = \int_D \delta(\phi(\mathbf{x}))|\nabla\phi(\mathbf{x})|d\mathbf{x}$$

[1]Currently, the number of iterations is set in advance by empirical observation of the stopping criteria. It is observed to depend on the parameter s. See Section 5.2.2.

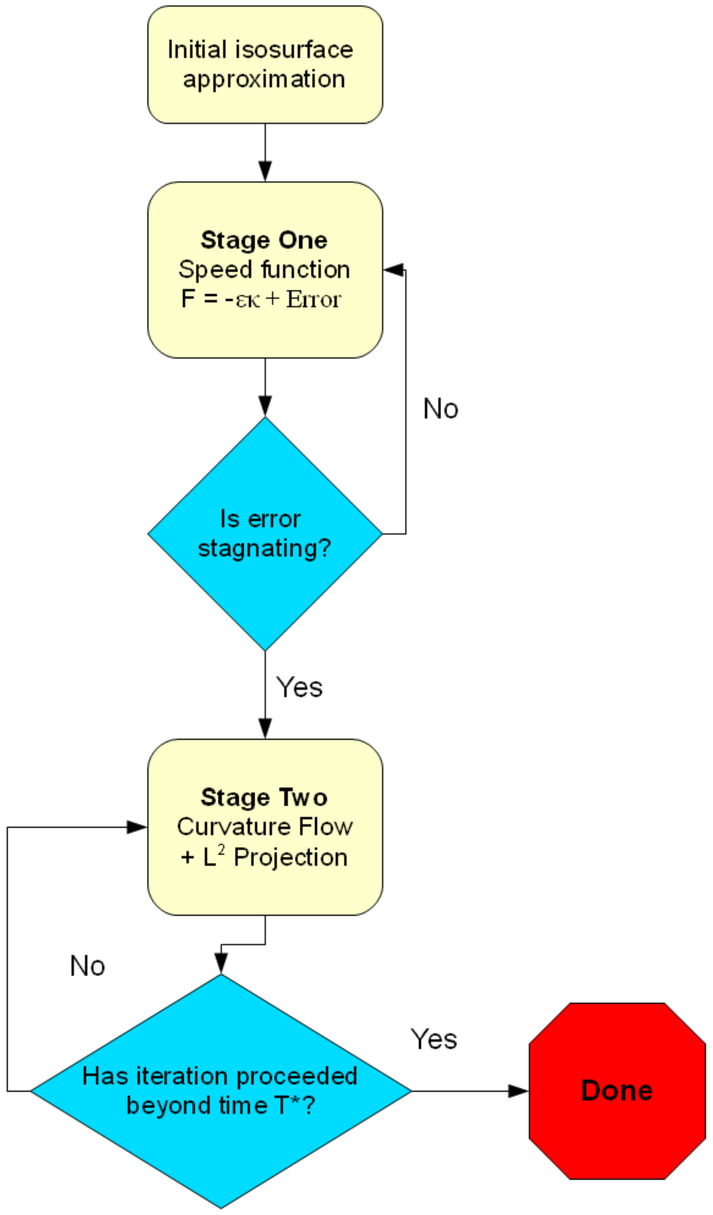


Figure 4.1: Flowchart of the algorithm as applied to single phase reconstruction.

whose steepest descent is calculated to be

$$\frac{d}{d\tau}L(\phi+\tau v)|_{\tau=0}=\int\limits_{\{\phi(\mathbf{x})=0\}}\left(\nabla\cdot\frac{\nabla\phi}{|\nabla\phi|}\right)\frac{v}{|\nabla\phi|}dS$$

which is curvature flow on the surface. See Jung et al for a discussion of a similar variational level set problem (except with only one volume constraint) [13].

The volume functionals, $V_i(\phi)$, are given by

$$(V\phi)_i:=\int\limits_{C_i}H(-\phi)dx$$

for each cell $C_i\in\Omega$, where H is the Heaviside function.

4.2.1 Discretization of ϕ

For an arbitrary $M\times N\times P$ grid of volume fractions $V_r(I,J,K)$, $I\in 1\ldots M$, $J\in 1\ldots N$, $K\in 1\ldots P$, we define a level set function on a subgrid (refer to Figure 4.2) of size $m\times n\times p$, where $m=Ms, n=Ns, p=Ps$, and s our subdivision parameter (usually chosen to be $s=3$ in 3D; see Chapter 5). Grid points $(Is..Is+s, Js..Js+s, Ks..Ks+s)$ will represent volume fraction block (I,J,K).

Both stages of our algorithm (which we will discuss in Sections 4.3 and 4.4) advance the level set function numerically via Euler's Method:

$$\phi^{n+1}=\phi^n+\Delta t(\epsilon\kappa^n-F^n)\nabla\phi^n,$$

κ is discretized according to the 3^2 or 3^3 stencil discussed in Chapter 2. Time step restrictions for advancing ϕ are $\Delta t\leq\frac{h^2}{2d\epsilon}$ for curvature flow (d is the dimension), and $\Delta t\leq h/\max(|F_{ijk}|)$ for the expansion/contraction term F.

4.2.2 Discretization of the volume functionals

To compute the volume functionals on the fine grid, we use piecewise linear interpolatation of our level set function ϕ with the "Marching Tetrahedrons" splitting of the domain [4, 19, 11], which has two important advantages:

- Each tetrahedron, made up of four points, determines a linear function exactly;
- Neighboring tetrahedra match up faces with each other, so that continuity is guaranteed.

It works as follows:

The domain is broken up into $1\times 1\times 1$ grid cubes on the fine mesh. Each cube is subdivided into six tetrahedra. Those tetrahedra, shown in Figure 4.3, are (offset from the point (i,j,k))

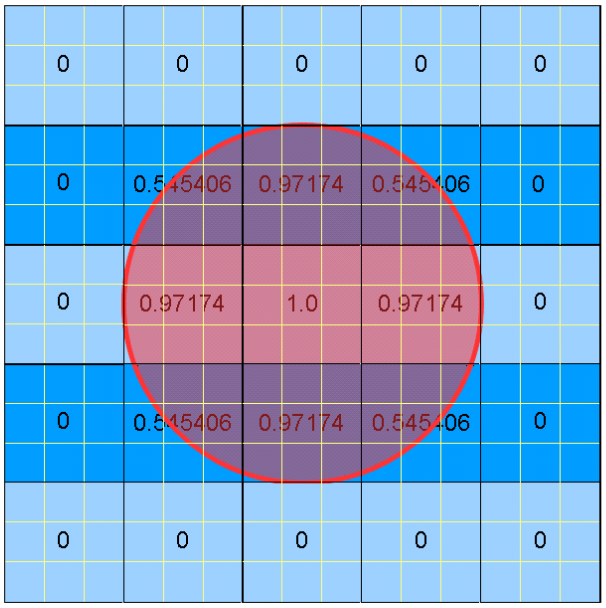


Figure 4.2: A 3×3 subgrid, in yellow, of the example volume fraction grid from Figure 1.1.

1. (0,0,0), (0,1,0), (1,1,0), (0,0,1)
2. (0,1,1), (0,1,0), (1,1,0), (0,0,1)
3. (1,1,1), (0,1,1), (0,0,1), (1,1,0)
4. (1,1,1), (1,0,1), (0,0,1), (1,1,0)
5. (1,0,0), (1,0,1), (0,0,1), (1,1,0)
6. (0,0,0), (1,0,0), (1,1,0), (0,0,1)

The tetrahedra are all isometric to each other and have volume $\frac{1}{6}$. For each tetrahedron, we compute the volume of the region where $\phi(\mathbf{x}) < 0$:

1. If the number of negative endpoints is 0 or 4, then the volume is 0 or $\frac{1}{6}$, respectively.
2. Otherwise, fit a linear interpolant $f(x, y, z) = c_0 + c_1 x + c_2 y + c_3 z$ and find the points along the edges of the tetrahedron where $f = 0$.
3. Split the tetrahedron into two regions $f > 0$ and $f \leq 0$.
4. If the number of negative endpoints is 1 or 3, then one region is a smaller tetrahedron, and its volume is computed via $\frac{1}{3}BH$; if its complement is the region with $\phi < 0$ then simply subtract from $\frac{1}{6}$.

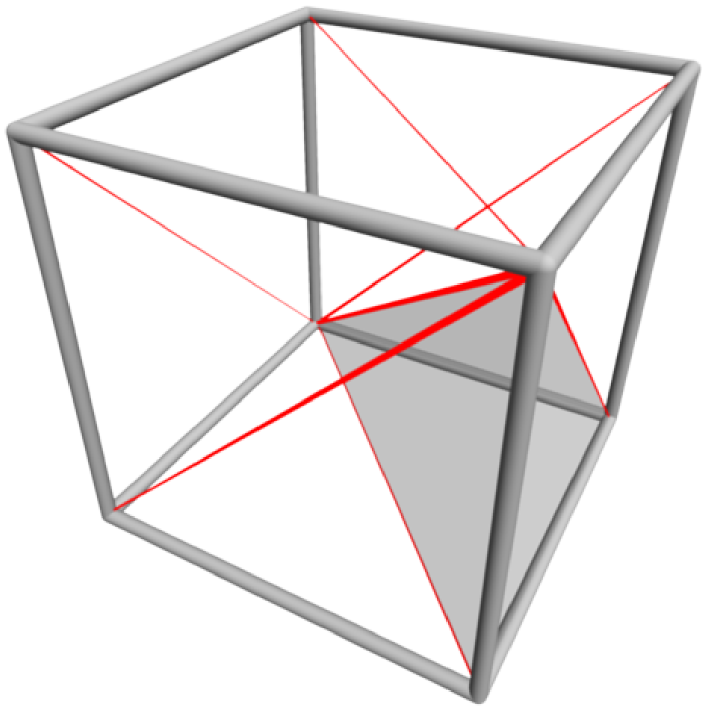


Figure 4.3: The Marching Tetrahedra splitting of the unit cube.

5. If the number of negative endpoints is 2, then the tetrahedron is split into two frustum-like regions. Either region can itself be split into one pyramid and one tetrahedron, and those volumes in turn may be computed. See Figure 4.4.

In 2D, this calculation is simpler, as we need only partition each square (i, j) to $(i+1, j+1)$ into two triangles, $(i, j), (i+1, j), (i, j+1)$ and $(i+1, j+1, (i, j+1), (i+1, j)$. We omit the details for brevity.

This calculation is not linear, unlike the formulas presented in Chapter 3, due to the use of conditionals involved in computing volume. However, in Section 4.3, we linearly approximate the volume operator by locally treating the curve as a height function.

4.3 First approximation

To initialize the level set function ϕ, we compute an isosurface of the volume fraction grid. More specifically, we set

$$\phi = \mathcal{I}(V_r - c)$$

where $c \in \mathbb{R}, 0 < c < 1$, and $\mathcal{I}$ is an interpolation operator, which we now define.

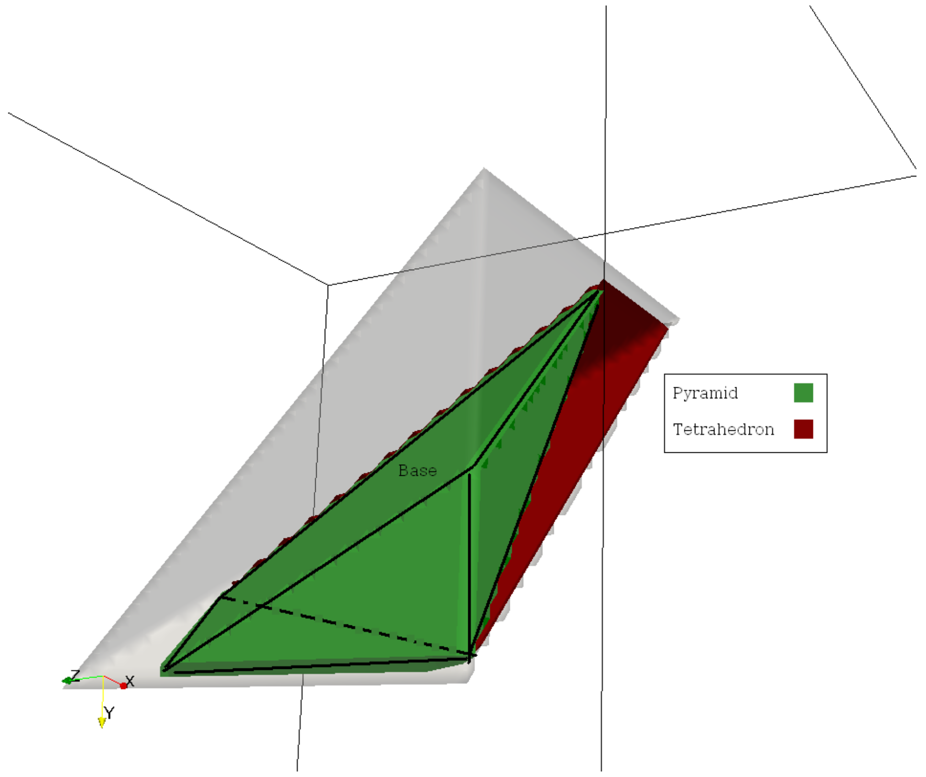


Figure 4.4: Illustration of splitting of the tetrahedron, in the case of two positive and two negative vertices.

The operator $\mathcal{I} : \mathbb{R}^{M\times N\times P} \to \mathbb{R}^{m\times n\times p}$ has the form $\mathcal{I} : G \mapsto g$, where $G(I, J, K), I \in 1 \dots M, J \in 1 \dots N, K \in 1 \dots P$ is defined on the coarse grid, and $g(i, j, k)$, $i \in 0 \dots m, j \in 0 \dots n, k \in 0 \dots p$ is defined on the fine grid. We assume the midpoints of each cell, $(sI + \frac{s}{2}, sJ + \frac{s}{2}, sK + \frac{s}{2})$, have the values $g(sI + \frac{s}{2}, sJ + \frac{s}{2}, sK + \frac{s}{2}) = G(I, J, K)$, and the remaining points are interpolated trilinearly from the nearest eight of these values. (If s is odd, the process is the same. The value of g in the middle is simply a convenience for defining interpolation, since no grid point exists there.)

The choice of c determines which isosurface of the interpolated volume fraction grid will be the initial guess. We choose c small, e.g. 0.01, so that the algorithm will "shrink" and smooth the surface towards the solution, instead of expanding and sharpening.

Alternatively, the level set function can be initialized to have spheres of the correct volume occupying each cell (and overlapping when such sphere does not fit within). This has the advantage of adapting to a wider range of topologies, but the disadvantage of starting the interface with oscillations at every grid cell, and hence curvature flow takes longer to damp them out.

The level set function is advanced via a speed law F containing (a) motion in the normal direction, and (b) curvature:

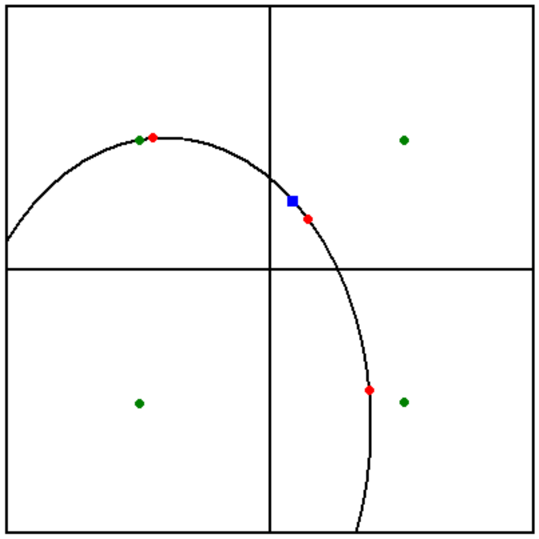


Figure 4.5: Our interpolation computes the value of the speed function at a point (blue square) by interpolating between data given at the cell centers (green dots). However, ideally the speed law should depend only on arc length position on the interface (e.g, interpolation based on the points in red.) The projection step will have this property.

$$F = \mathcal{I}[V_r - V_a] - \epsilon\kappa$$

with $V_a(I, J, K)$, the actual volume made up by our shape within a cell (I, J, K), computed via the Marching Tetrahedra algorithm discussed previously, and $\mathcal{I}$ is the interpolation operator defined previously. The goal is to advance the front under smoothing (κ), and correct any existing volume errors in the flow ($V_r - V_a$).

We interpolate the error term $V_r - V_a$, since by leaving the term piecewise constant, the speed function at boundaries is not well defined. It may not be possible to achieve matched volumes in all cells if, by correcting one cell, the value of its neighbors is altered unpredictably. We use trilinear interpolation using values of the error function at cell centers because it is simple and fast to evaluate on an Eulerian grid.

Interpolation of the error term is not equivalent to the projection step. It makes the speed function depend on the Eulerian position, but it should only depend on the position relative to the interface. See Figure 4.5 for further illustration of this point.

4.4 Projection algorithm

The main algorithm requires the flow

$$\phi_t + \mathcal{P}(\kappa)|\nabla\phi| = 0$$

where $\mathcal{P}$ is an approximate projection operator, which requires choosing the closest vector v to κ such that v does not change the volume fractions. We write $\mathcal{P}(\kappa) = \kappa + w$ to break the algorithm into two steps.

- $\phi^{n+1/2} = \phi^n + dt\,\kappa$. Compute $V_a(\phi^{n+1/2})$.
- Compute w which is the smallest change restoring $V_a(\phi^{n+1/2})$ to V_r (or to $V_a(\phi^n)$).

In Chapter 3, we saw that this was equal to subtracting the smallest such perturbation (in the L^2 sense) that cancels out the volume changes.

4.4.1 Limit as $s \to \infty$

In the limit as the number of subdivisions $s \to \infty$, in the height function case, we obtain the projection to be a piecewise constant flow. Since the goal is to minimize $\frac{1}{2}\int w^2 dx$ subject to $\int_{x_i}^{x_{i+1}} w dx = c_i$, the variational form of Lagrange multipliers gives

$$\begin{aligned}
0 &= \frac{d}{d\tau}\left(\frac{1}{2}\int_0^b (w+\tau v)^2 - \sum_{i=1}^N \lambda_i \int_{x_i}^{x_{i+1}} (w+\tau v)dx\right)\Bigg|_{\tau=0} \\
&= \int_0^b wvdx - \sum_{i=1}^N \int_{x_i}^{x_{i+1}} \lambda_i v dx \\
&= \sum_{i=1}^N \int_{x_i}^{x_{i+1}} v(w-\lambda_i)\,dx
\end{aligned}$$

for any $v \in \mathcal{L}^2$, which indicates that w is piecewise constant. Unfortunately, as discussed earlier, piecewise constant flows have convergence problems. Our parameter s is quite small, invalidating the applicability of the limit process.

The projection step must be done according to the gradients of the volume functionals, as discussed in Chapter 3.

4.4.2 Projection on a nonuniform grid in one dimension

Recall we sought the minimum of $\|w\|^2$ with $Av = A\kappa$ and found that

$$w - \sum_{i=1}^N \lambda_i A_{i,:} = w - A^T\lambda = 0$$

We deduced that w was a linear combination of the area gradients. In the case of the trapezoidal rule, w was a linear combination of the vectors $(\frac{1}{2}, 1, \dots, 1, \frac{1}{2})$ on each interval, which is crucially different from the constant flow, $(1, \dots, 1)$, near the edges of volume fraction cells.

A difference between the height function case and the general case, as we will see in the next section, is that the surface mesh will in general not be equally spaced nor aligned to boundaries. More precisely, the discrete points on the level set grid which we can modify to change the interface will not all change the interface by the same amount[1]. By examining how to perform the projection on a nonuniform grid in one dimension, we can determine the necessary steps in the general case.

To minimize $\int w(x)^2 dx$ on a nonuniform grid (grid points $0 \dots n$), the trapezoid rule gives us an expression of the form

$$\sum_{i=0}^{n} \rho_i w_i^2$$

where $\rho_i = \frac{1}{2}(\rho_{i,l} + \rho_{i,r})$, where $\rho_{i,l}$ is the distance to the left neighboring point, and $\rho_{i,r}$ is the distance to the right.

The volume constraints are

$$\left(\frac{1}{2}, 1, \dots, \frac{1}{2}\right) \cdot (\rho_{Is,r} w_{Is}, \rho_{Is+1} w_{Is+1}, \dots, \rho_{Is+s,l} w_{Is+s}) = c_I$$

for $I = 1 \dots N$ yield the minimizer

$$\begin{array}{rcl} \rho_i w_i & = & \sum_{I=1}^{N} \lambda_I (0, \dots, \rho_{Is,r}, \rho_{Is+1}, \dots, \rho_{Is+s,l}, 0, \dots) \\ w_i & = & \sum_{I=1}^{N} \lambda_I (0, \dots, \frac{\rho_{Is,r}}{\rho_{Is}}, 1, \dots, 1, \frac{\rho_{Is+s,l}}{\rho_{Is+s}}, 0, \dots) \end{array} \qquad (4.1)$$

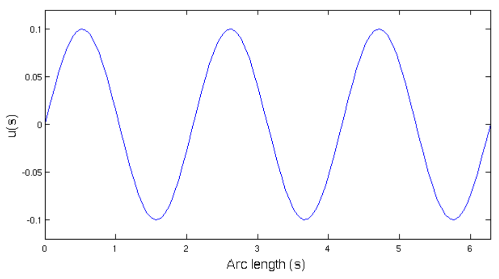
The coefficients of the interior nodes are still 1, while the edge node coefficients are equal to their contribution to the cell in question, divided by their contribution to all cells.

4.4.3 Calculation of ∇V_{IJ} on the level set grid

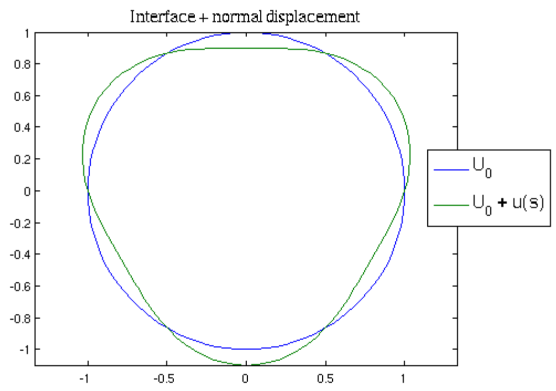
In the case of an interface not represented as a graph of a function, we rely on the following formulation. Let $u = u(s)$ represent the normal perturbation of the interface, in other words, u is the height function in a coordinate system with axes the tangent and normal directions at each point on the interface. In Figure 4.6, we show an example where the interface is a circle, and the displacement is a sine wave. To first order in $\|u(s)\|_\infty$, the change in volume is $\Delta V = \int u(s)\, ds$.

In an implicitly-defined interface, we interpret our volume gradients as being normal displacement functions. We can create these functions via adding values to individual grid

[1]A grid point may correspond to a point exactly on the boundary, a point $1/s$ distance away, or anywhere in-between. The unevenness is determined by the change in curvature along the interface.



(a) Normal displacement function $u(s) = \sin(3s)$.



(b) Interpretation of $u(s)$ as a modification (green) to the original circular interface (blue)

Figure 4.6

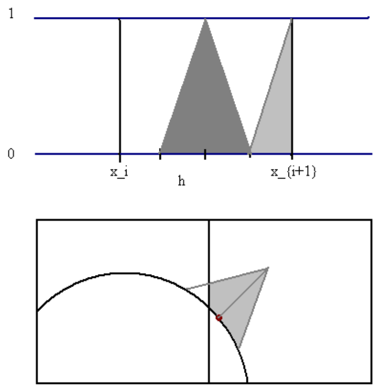


Figure 4.7: Illustration of the the computation of gradient functionals. On top, in the 1D function graph case; on bottom, the extension to the level set case.

points. These changes will correspond to moving the interface at the closest point to the grid point by some perturbation.

For a point ij close to the interface, we compute its corresponding volume gradient normal displacement as follows (see also Figure 4.7):

1. Extend the interface outward a length τ via altering ϕ_{ij}.
2. Compute the change in volume in each applicable volume fraction cell caused by this extension. For a given cell IJ, call this $v_{IJ}(\tau)$.

3. Take the quantity $\frac{v_{IJ}(\tau)}{\tau \sum\limits_{I'J'} v_{I'J'}(\tau)}$, for $I'J'$ neighboring cell IJ, as $\tau \to 0$. In practice this means choosing a sufficiently small τ.

The result is the coefficient of the IJth volume gradient for the component corresponding to the point on the interface nearest ij.

4.4.4 Computing λ and updating the interface

Let $\mathcal{S}$ be the space of surfaces in $\mathbb{R}^d$, and $U \in \mathcal{S}$. Let $V : \mathcal{S} \to \mathbb{R}^c$ be the operator that inputs a surface and returns the volume fractions corresponding to it, where c is the total number of volume fraction cells, $c = MN$ (or MNP in 3D).

1. We linearize V in terms of the normal displacement functions defined on U_0.

   $$V(U) \approx V(U_0) + [\nabla V_i(U_0) \cdot (U - U_0)]_{i=1\ldots c}$$

   The ∇V_i are the volume gradients whose components are described above. We denote $dV := (\nabla V_i)_{i=1\ldots c}$. Both the ∇V_i and $(U - U_0)$ are represented as normal displacement functions.

2. We define the $c \times c$ matrix G such that

   $$G_{k,l} = V_l(\nabla V_k),$$

   which represents the volume changes the gradients sweep out within the volume fraction grid. (Recall from Chapter 3 the matrix $G = AA^T$.) We calculate this as follows. For a volume gradient corresponding to cell IJ, we obtain coefficients for each point close to the interface. Using this as the flow velocity, we move the interface for a smell time step τ and finally, compute volume changes in cell IJ and its neighbors.

3. We compute $dV\kappa$ by flowing $U^\kappa = U_0 + \Delta t\kappa$, then computing $V(U^\kappa) - V(U_0)$.

4. We invert to find $\lambda = G^{-1}(dV\kappa)$

5. We flow $U \leftarrow U_0 + \Delta t\kappa + dV^T\lambda$ (we abuse notation to write $dV^T\lambda := \sum\limits_{i=1}^{c} \lambda_i \nabla V_i$).

The inversion can be done exactly via LAPACK **dgesv**, or via a few iterations of Gauss-Seidel, which emulate the local inversion formulae in Chapter 3.

Finally, it is often not feasible to start with (nor to maintain) a perfectly zero error interface, and as such, the volume changes of curvature is not the only term being projected. In practice we use

$$\lambda = G^{-1}(dV\kappa + \alpha(V_r - V_a))$$

and use $\alpha = 1$ for smooth shapes, but $\alpha < 1$ is recommended for shapes with high curvature or loss of smoothness.

We note that there are alternative ways to perform such a projection step. For example, Fatemi and Sussman [28], in an algorithm to reinitialize $\phi(x)$ to a signed distance function, introduced a constraint to conserve volume within each fine grid cell. They accomplished this by a projection defined using the gradient of a numerical Heaviside function.

4.5 Relaxing the constraints

4.5.1 Formulation

Since reconstructing volume fractions corresponding to regions with corners or edges leads to interfaces with oscillations, we also include a formulation that allows some reconstruction error in exchange for a smoother shape. We recast our problem as the minimum solution of

$$F(\phi) = \epsilon \int_{\phi=0} 1\, ds + \frac{1}{2}\|V_r - V_a(\phi)\|^2, \tag{4.2}$$

where V_r is the vector containing the required volume fractions in each grid cell, and V_a the corresponding vector for volumes occupied by the shape $\phi \leq 0$. Gradient descent yields, for surface area minimization,

$$\nabla F = \epsilon\kappa + dV^T(V_r - V_a)$$

where we denote the volume gradients just discussed by dV^T.

4.5.2 Relationship between ϵ and error

As ϵ moves increases from 0, the error, at first, increases linearly. First, we examine a basic example: a circle, with a single volume fraction. Surface area minimization acts to shrink the circle, while the volume error term acts to stretch it to its original radius. Writing equation 4.2 for this problem in terms of $\rho = r - 1$, we have

$$\begin{aligned} F_\epsilon(\rho) &= 2\pi\epsilon(\rho+1) + \tfrac{1}{2}[\pi(1-(\rho+1)^2)]^2 \\ 0 = \tfrac{d}{d\rho}F_\epsilon(\rho) &= \epsilon + \pi(\rho^3 + 3\rho^2 + 2\rho) \end{aligned}$$

Taking the derivative with respect to ϵ we get

$$\rho_\epsilon = \frac{-1}{2\pi + 6\pi\rho + 3\pi\rho^2}$$

and since $\lim_{\epsilon\to 0}\rho = 0$, $\rho_\epsilon|_{\epsilon=0} = \frac{-1}{2\pi}$. Since $\rho_\epsilon \neq 0$, the difference in radius is locally linear in ϵ. The L^1 error is also locally linear in the radial difference.

For a one-dimensional height function, we demonstrate the linearity of the error versus ϵ as follows. Let the interval $[0, L]$ be partitioned into intervals x_I, with constraints $\int_{x_I} u(x)dx = c_I$, c_I as input. The objective function is

$$F_\epsilon(u) = \epsilon \int_0^L \sqrt{1+u_x^2}\,dx + \frac{1}{2}\sum_I \left(\int_{x_I} udx - c_I\right)^2 .$$

For a given search direction $v \in L^2$, we locally approximate $F_\epsilon(u+\tau v), \tau \in \mathbb{R}$ as a quadratic about $\tau = 0$. We have

$$\frac{d}{d\tau}F_\epsilon(u+\tau v)|_{\tau=0} = \epsilon \int v_x \frac{u_x}{\sqrt{1+u_x^2}}dx + \sum_I \int_{x_I} vdx \left(\int_{x_I} udx - c_I\right)$$

and

$$\frac{d^2}{d\tau^2}F_\epsilon(u+\tau v)|_{\tau=0} = \epsilon \int \frac{v_x^2}{(1+u_x^2)^{3/2}}dx + \sum_I \left(\int_{x_I} vdx\right)^2$$

Solving for the minimum τ^* of $F_\epsilon(u) + \tau F'_\epsilon(u) + \frac{1}{2}\tau^2 F''_\epsilon(u)$, fixing notations for the quantities $Au := \left(\int_{x_I} udx\right)_{I=1\ldots N}$, $r := Au - c$, we get:

$$\tau^* = \frac{\epsilon \int v^2 dx + r \cdot Av}{|Av|^2 + \epsilon \int \frac{v_x^2}{(1+u_x^2)^{3/2}}dx}.$$

Using $u = u_0$, the solution to $\epsilon = 0$, and $v = \kappa_0$,

$$\tau^* = \frac{\epsilon \int \kappa_0^2 dx}{|A\kappa_0|^2 + \epsilon \int \frac{\kappa_{0,x}^2}{(1+u_x^2)^{3/2}}dx}.$$

This approximation is valid for small τ, and we see that $\tau^* = O(\epsilon)$, so we can use this approximation for small ϵ. Finally, our new minimum is

$$u_\epsilon \approx u_0 + \tau^* \kappa_0,$$

and we infer the L^1 difference u and u_0 will be on the order of $|\tau^*|\ \|\kappa_0\|$.

4.6 Multiple phase reconstruction

In this section, we apply the first approximation and projection stages of our algorithm towards reconstructing multi-phase volume fraction data. In 4.6.1, we define what multi-phase volume fraction data means. In Section 4.6.2, we describe a straightforward extension of our algorithm to reconstructions from this data. In Section 4.6.3, we define the Voronoi Interface, which is used to partition, without gaps or overlap, the domain into the regions associated with each phase. Finally, in 4.6.4, we discuss how to compute volumes on the arbitrary polyhedra that emerge with the Voronoi Interface.

4.6.1 Definition

Multiple phase volume fraction data is defined thus: Given m materials, and N cells, for each cell I we have m volume fractions $v_j^I \geq 0$ such that

$$\sum_{j=1}^{m} v_j^I = 1$$

Each v_j^I represents the fraction of cell I occupied by material j.

A grid with a single volume fraction per cell is equivalent to two-phase data, where the second phase is defined simply as $v_2 = 1 - v_1$ for each cell in the grid.

4.6.2 Algorithm

The algorithm can be extended toward multiple phase reconstruction as follows. Let $\phi_i : \mathbb{R}^d \to \mathbb{R}$, $i \in 1 \ldots m$, be level set functions such that $\phi_i < 0$ corresponds to the region inside phase i, and outside otherwise.

1. Perform one iteration for each material i with level set function ϕ_i independently, reducing to an iteration of the single phase reconstruction algorithm.

2. Partition the domain into regions $R_1 \ldots R_m$ corresponding to each material using the Voronoi Interface (defined by Saye and Sethian [24]). Compute the volume of each region.

3. Repeat from step 2.

4.6.3 The Voronoi Interface

The solution requires that a partition of the grid such that each region is occupied by exactly one phase. If we use the zero level set of each ϕ_i separately, then points x such that $\phi_i(x) < 0$ and $\phi_j(x) > 0\ \forall j \neq i$ are unambiguously located within R_i; however, where level set functions overlap ($\phi_i(x) < 0$ and $\phi_j(x) < 0$), or no level set function is inside ($\phi_j(x) > 0\ \forall j$) are ambiguous. To resolve this, we use the Voronoi Interface: we say a point x is within R_i iff

$$\phi_i(x) < \phi_j(x)\ \forall j \neq i$$

For each tetrahedron in the Marching Tetrahedrons partitioning of the fine grid, we compute boundaries of the above regions:

1. For any two level set functions occupying the cell, i, and j, if ϕ_i and ϕ_j are sufficiently close to 0, compute the polygon $\phi_i(x) - \phi_j(x) = 0$, using the linear representation of each function. Call this polygon B.

2. For any third level set function ϕ_k sufficiently close to 0, within the polygon B, compute the line where $\phi_k(x) = \phi_i(x)$.

3. Replace B with the polygon B with $\phi_k(x) < \phi_i(x)$ cut off.

4. Repeat (2) for third, fourth, etc level set functions, if applicable.

4.6.4 Volume calculation

We compute the exact volumes swept out by the piecewise linear Voronoi interfaces as follows:

1. We begin with the unit tetrahedra from the Marching Tetrahedra splitting. To find the volume occupied by Material A, we repeatedly chop the polyhedron by the plane $(\phi_A = \phi_B)$ for all materials B.

2. Compute the volume of the resulting polyhedron by

$$\begin{aligned} V = \int_P 1\,dx\,dy\,dx &= \int_P \tfrac{1}{3}\mathrm{div}(x,y,z)\,dx\,dy\,dz = \tfrac{1}{3}\int_{\partial P}(x,y,z)\cdot\hat{n}\,dS \\ &= \tfrac{1}{3}\sum_F \mathrm{Area}(F)\bar{x}(F)\cdot\hat{n}(F) \end{aligned} \tag{4.3}$$

   where F are the faces of the polyhedron P.

Chapter 5

Results

In this section we document the results of the algorithms described in Chapter 4 applied to several test cases.

1. In Section 1, we focus on these basic tests.
    - We reconstruct lines in 2D of various angles.
    - We reconstruct the sphere, torus, and ellipsoid (in 3D) and discuss convergence as a function of the number of iterations.
    - We reconstruct a sphere in 3D, measuring the XOR-volume between the reconstruction and model, as well as error in computed normal vectors. We discuss convergence as a function of subgrid size (s), and volume fraction grid size (N).
2. In Section 2, we discuss how to generate the remaining test cases, and what quantities we measure about each.
3. In Section 3, we give tables and figures for several examples of two-dimensional reconstructions, both single and multi-phase.
4. In Section 4, we repeat the procedure for three-dimensional data.
5. In Section 5, we examine some test cases from Section 4 under the relaxed optimization reconstruction algorithm. Specifically, we examine those single material model interfaces that have sharp corners or edges.
6. Finally, in Section 6, we show side-by-side comparisons of figures from the Active Interface Reconstruction paper [3] with our own results.

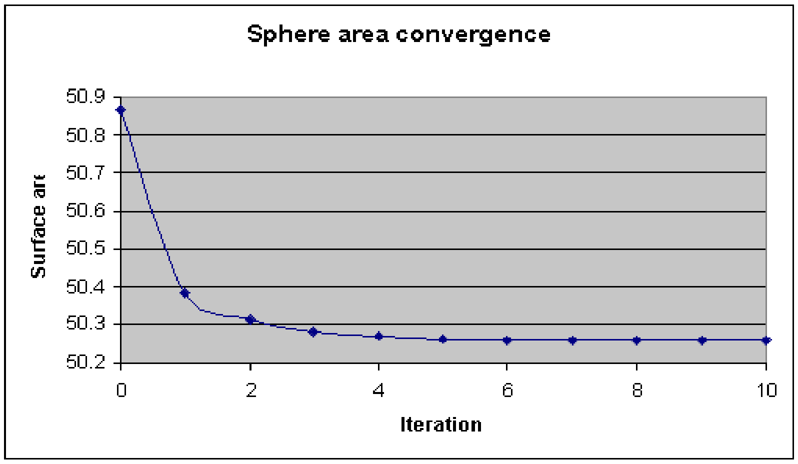


Figure 5.1: The algorithm acts to reduce surface area. Grid size is 40^3.

5.1 General convergence results

5.1.1 Convergence of smooth shapes by number of iterations

We first fix our domain as the cube $[-3, 3]^3$ and examine three inputs: volume fractions corresponding to

- the sphere of radius 2,
- a torus with radii $1.5, 0.9$, and
- an ellipsoid with semiaxes $2.4, 1.6, 2.0$.

The ellipse and torus are rotated by $\pi/5$ about the x-axis, and $\pi/5$ about the y-axis, so that there is no bias towards grid alignment. For this first test, we use a volume fraction grid size of 40^3 and $s = 2$ subdivisions.

We compare the difference between shapes as the volume of the region

$$\mathrm{XOR}(A, B) = \{x : x \in A, x \notin B \text{ or } x \notin A, x \in B\}$$

computed by method similar to that used to find the Voronoi interface. Since both the sphere and the torus are surfaces of constant mean curvature, the reconstructed solutions should be exactly the same sphere and torus. In practice, they are close, but not exact – this is due to two factors: (1) the reconstruction triangulation can only approximate a curved surface to first order, and (2) the projection is approximate to first order.

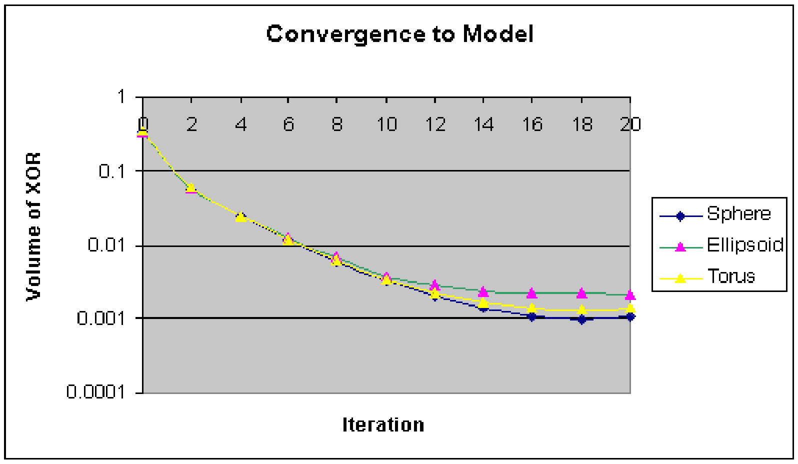


Figure 5.2: Convergence to model shape for three input shapes. Grid size 40^3.

Quantifying these difference, the XOR errors for the sphere and torus are both approximately 10^{-3} on a 40^3 grid. By dividing by the number of volume fraction cells neither completely filled nor empty, this amounts to an average XOR error per cell of less than 10^{-6}. The ellipsoid, however, is not a surface with constant mean curvature. Consequently, the minimum surface area reconstruction will have lower area than the ellipsoid itself (we computed 0.0016 less surface area, from an original area of 49.991), and we will expect that the XOR volume should not be zero, even in theory. Nevertheless, the average XOR error per cell compared to the ellipse is still small (approximately 10^{-6}).

5.1.2 Convergence of sphere by grid size and number of subdivisions

We then measured convergence based on increasing the grid size (N^3) for a sphere. In Table 5.1, we see that the XOR-difference between meshing the exact sphere level set to the reconstruction result is second order in N, while the difference in normal vectors is first order. For higher order accuracy, we would need curved elements (or a higher order meshless volume calculation). The normal difference between $\hat{\mathbf{n}}$ and $\mathbf{n}$ is calculated by $\arccos(\hat{\mathbf{n}} \cdot \mathbf{n})$, where $\hat{\mathbf{n}}$ is the vector from a point to the origin (normalized), and $\mathbf{n}$ is a vertex normal – the average of normals of all faces sharing the vertex. Other normals yielded the same order of accuracy, but higher absolute error.

Finally, we fix a grid size of 32^3 and vary our subdivision parameter, s. See Table 5.2. The order of convergence for the XOR volumes and the normal errors are the same as increasing

Grid size, N	XOR-Diff	Normal error
15	0.171	0.033
20	0.090	0.023
25	0.0578	0.0169
30	0.0413	0.0147
40	0.0229	0.0120
50	0.0144	0.0092

Table 5.1: Convergence for a sphere, $s = 2$ subdivisions. XOR is second order, while normals are first order.

Subdivisions, s	XOR-Diff	Normal error	Time (s)
2	0.0353	0.0142	7.8
3	0.0148	0.00964	37
4	0.0077	0.00675	122
5	0.0044	0.0545	350
6	0.0026	0.0446	734

Table 5.2: Convergence for a sphere, $N = 32$. Again, XOR is second order, while normals are first order. However, time is approximately 5th order.

grid size. However, we note that the run-time is approximately quintic in s. Firstly, the number of grid points is cubic in s, and secondly, the time step restriction is quadratic in $s = 1/h$, therefore, to flow curvature up to a fixed time T, we require quadratically more time steps. The accuracy is sufficient for our purposes with 2-3 subdivisions as well, and so we therefore restrict our algorithm to 2-3 subdivisions in 3D unless otherwise noted.

Grid convergence, however, need not be quintic in time. We assume that the width of a coarse cell is constant (or 1) to preserve feature size at one volume fraction cell. Consequently, the time step restriction does not increase. Furthermore, the region in which the interface lies is of co-dimension one (the remaining cells are either empty or filled completely), and so total compute time should be nearly quadratic. See Tables 5.3, 5.4, 5.5, and 5.6.

5.1.3 Convergence of straight lines

Taking a step back, we verify that the proposed algorithm works for straight lines. In 2D, we ran a sequence of random lines through the origin on a 7^2 grid. On average over 10 slopes, the error was 2.73×10^{-5} (measured as the sum of errors in volume in all cells, divided by the number of cells neither empty nor completely occupied). On further inspection we noted that the center cell's error was always much better than the average error, and the fact that the error was not equal to machine-precision was due to boundary effects. Indeed, as no information can flow inside from outside the boundary, the fast marching method for a line

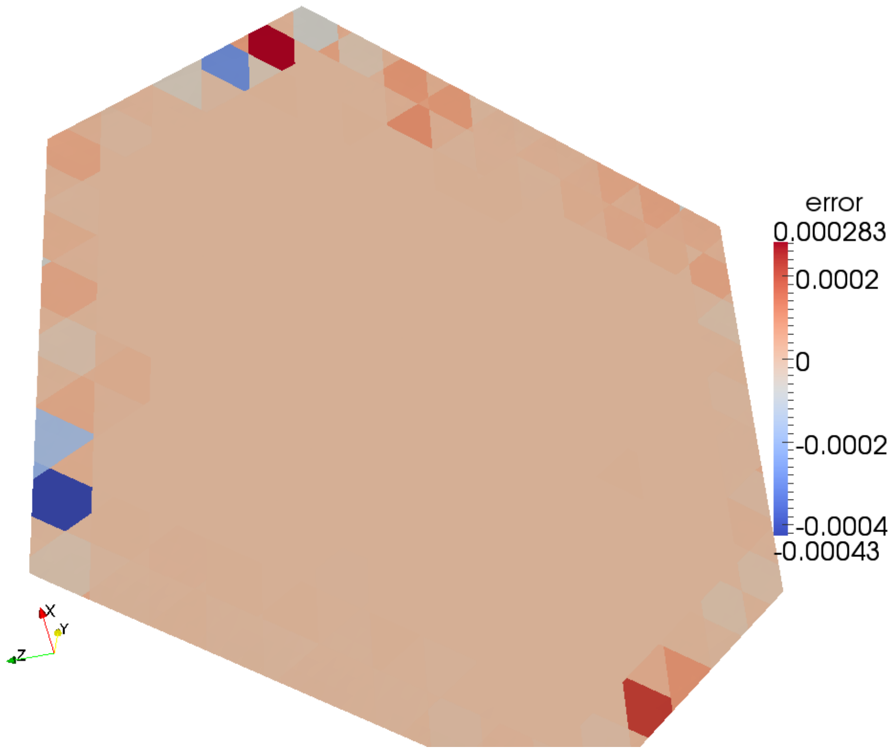


Figure 5.3: Reconstruction of a flat plane in 3D. Boundary effects lead to error on some cells on the order of 10^{-4}, but otherwise the plane is faithfully reproduced.

will not reinitialize ϕ to be a linear function, which, under many iterations of curvature flow, causes a slight bend at the boundary. Nevertheless, the algorithm was able to reproduce straight lines consistently to a reconstruction error of approximately 10^{-5} to 10^{-6}, which empirically is the most accurate the algorithm has been able to deliver in any cases.

The situation is analogous in 3D. For the plane on a 15^3 grid, $s = 3$, not aligned to any axis, our result had an error of 8.2×10^{-6}; see Figure 5.3.

5.2 Models, process, and measurements

5.2.1 Test case generation

For each test case, a model surface (or collection of surfaces) is chosen. For a single material, the model surface is defined implicitly, $\phi(x) = 0$, and the surface is triangulated using linear interpolation in a fine grid. We compute the volume fractions partitioned by this triangulation within our coarse grid, and these serve as input to the reconstruction algorithm.

For m multiple materials, the model surfaces are defined as the the Voronoi interface for level set functions $\phi_1, \ldots, \phi_m$. The volumes of the Voronoi regions are computed using the volume formula given in Section 4.6.2.

5.2.2 Reconstruction process

In the following examples, the reconstruction algorithm was run with several iterations of "stage one", followed by several more iterations of "stage two." "Stage one" refers to the approximate algorithm described in 4.3 (speed law $F = -\epsilon\kappa + \mathcal{I}(V_r - V_a)$, a.k.a. "Loose Reconstruction" as referred to in some of the figures to follow), and "stage two" refers the main algorithm as described in Section 4.4 (curvature flow followed by an approximate L^2 projection to preserve volume).

The parameters of our algorithm were determined empirically. In more detail:

- **Number of subdivisions.** We found in Section 5.1 that compute time is quintic in s, and therefore s should be as small as possible. $s = 2$ worked well on very smooth interfaces, but it was not sufficient for more complicated shapes. $s = 3$, however, was able to perform well on all of our test cases in 3D.
- **Number of iterations per stage.** We determined the number of iterations by measuring convergence of a simple model shape. In 2D, it was the circle, in 3D, the sphere. For stage one, we chose when the reconstruction error stopped improving significantly; for stage two, we chose when the XOR-error between the model and result stopped improving.
- **Epsilon.** In the projection algorithm, ϵ only affects how much the interface can move in one iteration. We chose ϵ small enough that the projection step is still able to counteract the volume changes due to $\epsilon\kappa$.
- **Time step.** Finally, Δt was generally chosen to be as large as possible without violating the stability requirement.

The values of the parameters, once chosen, were fixed and run on an entire class of problems. The specific values chosen are as follows:

- In 2D, unless otherwise noted, there were 30 iterations of stage one, and 15 iterations of stage two, with $\epsilon = 0.02$, $s = 4$, and time step $\Delta_t = \frac{h^2}{6\epsilon}$.
- In 3D, $\epsilon = 0.04$ and $s = 3$. For a single material, the default was 20 iterations of stage one, and 8 iterations of stage two, with time step $\Delta t = \frac{h^2}{8\epsilon}$. For multiple materials, we used 10 iterations of both stage one and stage two, with time step $\Delta t = \frac{h^2}{6\epsilon}$.

5.2.3 Tables

The tables that follow contain measurements of reconstruction result data. *Grid* means coarse grid size, *Mixed* denotes the proportion of coarse grid cells that are occupied by at least two phases, *Time* is the run time in minutes:seconds, and *Average error* is, as defined in [3], $E = \frac{\sum_c \max_i |\text{Error}(c,i)|}{\#\text{Mixed}}$ (c is some cell, i represents a phase) – the average worst error per cell. In practice, our algorithm always commits some reconstruction error. Small ones are due to our approximate methods, with larger ones a matter of truncating high curvature regions.

"Time" is the run time when run on an Intel Core2Duo 3.0 GHz machine, running Visual Studio 2010 Express with full optimization and SSE2 enabled.

5.3 Two-dimensional results

5.3.1 Two phase

Name	Grid	Mixed	Time	Average error
Batman	50^2	7.7%	2.87s	6.6×10^{-5}
Circle	50^2	5.3%	2.38s	3.9×10^{-6}
Circle	100^2	2.7%	9.19s	5.4×10^{-5}
Cassini Oval	20^2	9.0%	0.64s	1.6×10^{-5}
Rounded Deltoid	25^2	9.9%	1.67s	2.6×10^{-5}
Letter 'A'	20^2	18%	0.76s	2.9×10^{-3}
Letter 'A' (allowing pinch off)	20^2	18%	0.76s	4.0×10^{-4}
Mandelbrot	200^2	6.8%	58.7s	1.1×10^{-3}

Table 5.3: Data from the algorithm applied to various single-phase two-dimensional test cases.

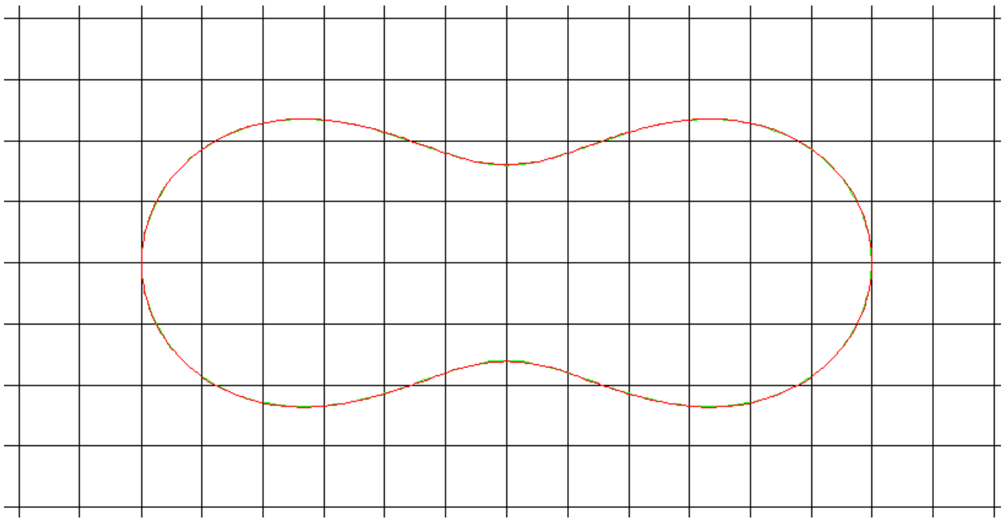
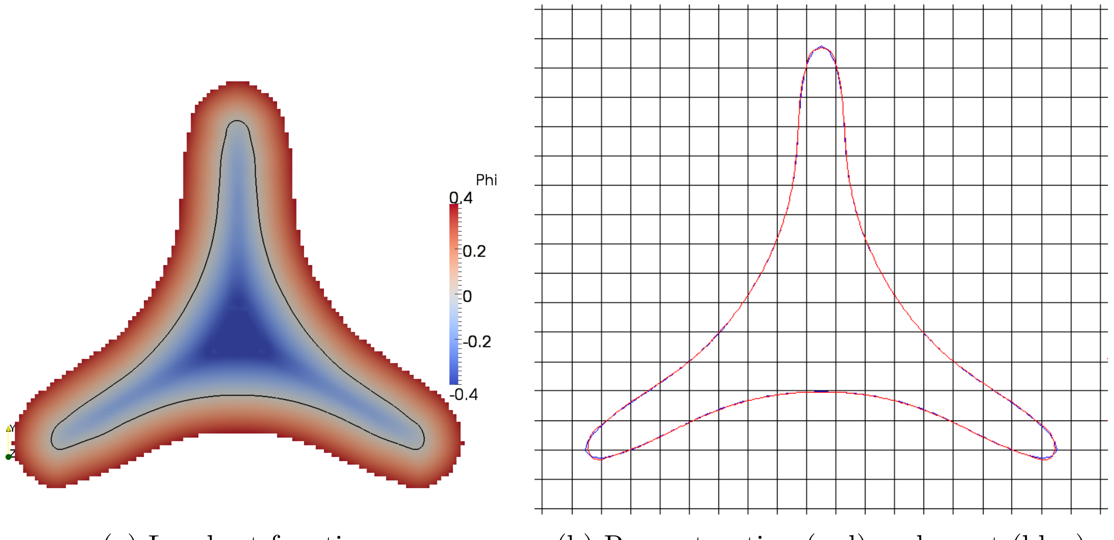


Figure 5.4: **Cassini Oval.** The model shape is given by $0 = (x^2+y^2)^2 - 3(x^2+y^2) - \frac{3}{4}$. Both the reconstruction (red) and exact solutions (green) are shown, with the volume fraction grid overlayed.



(a) Level set function (b) Reconstruction (red) and exact (blue)

Figure 5.5: **Rounded Deltoid.** The model equation is $0 = (x^2+y^2)^2 + \frac{9}{2}(x^2+y^2) - 4(x^3 - 3xy^2) - \frac{43}{16}$. On the left, the reconstructed level set function is shown, with zero contour in black; on the right, the reconstruction (red) and the exact (blue) are both drawn, with the volume fraction grid overlayed.

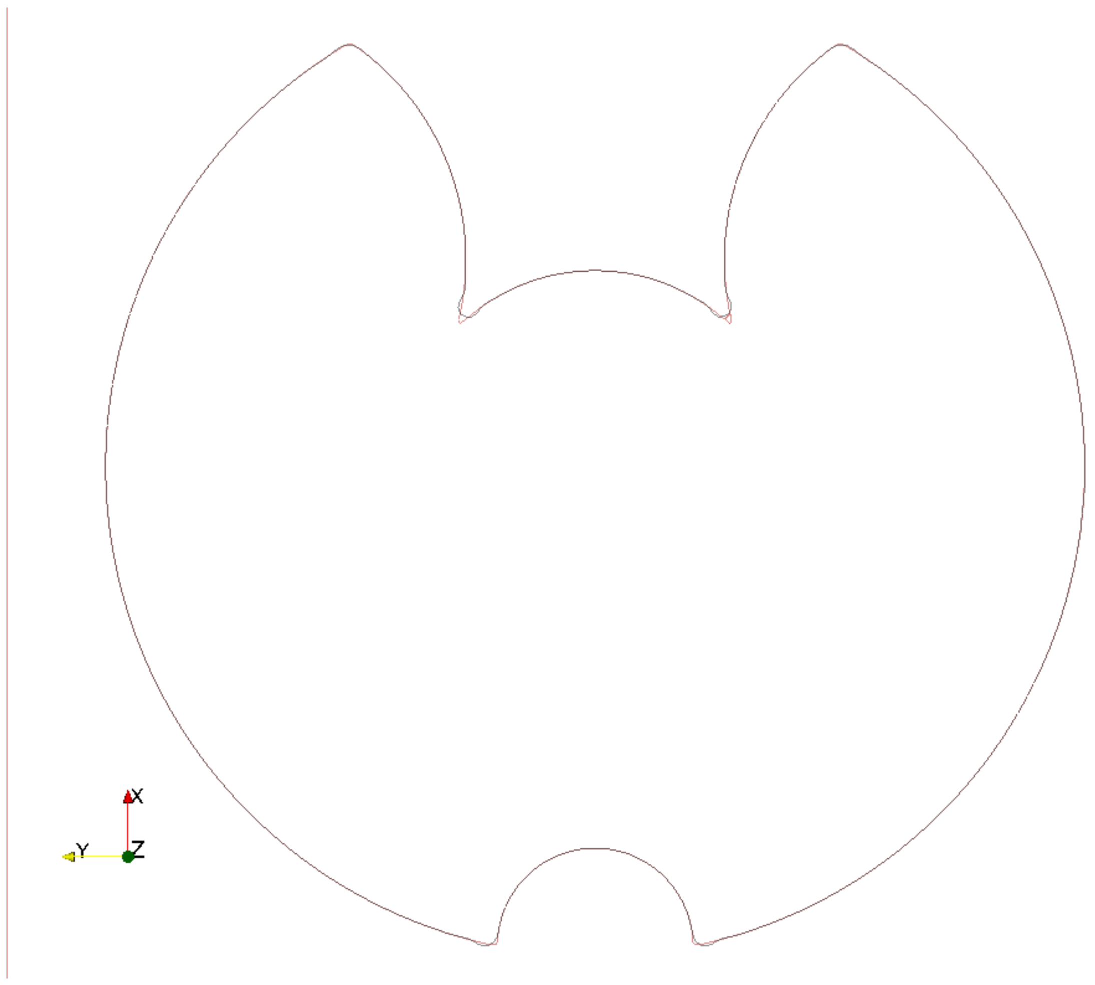


Figure 5.6: **Batman.** The model shape is made of circular arcs joined together, and is not C^2. The model is shown in red, and the reconstruction in green. The overlap is orange.

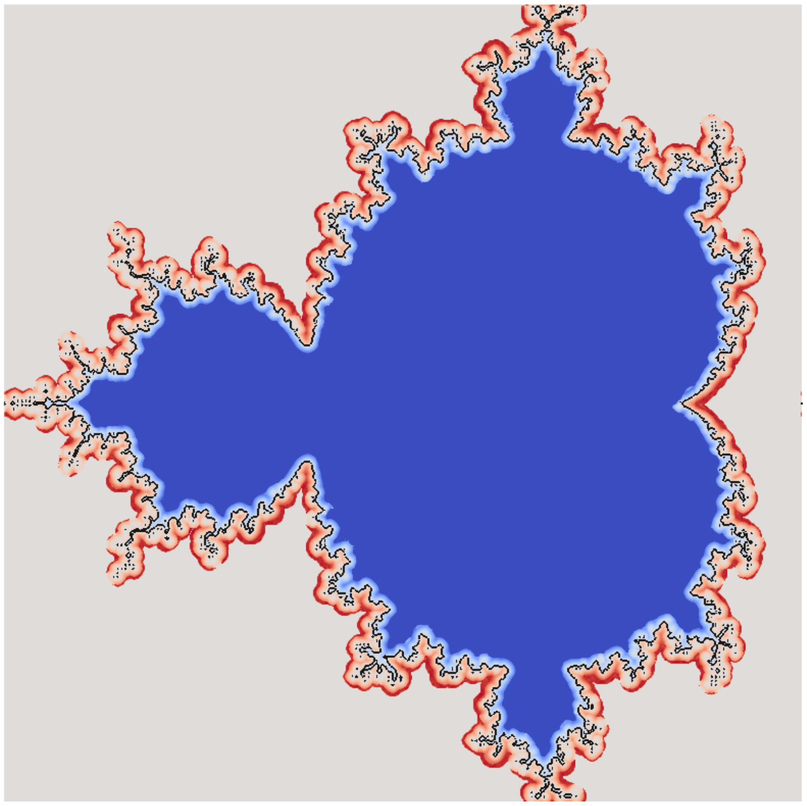


Figure 5.7: **Mandelbrot.** Reconstruction of the Mandelbrot set truncated at 20 iterations, i.e. the set of points $z_0 \in \mathbb{C}$ such that $|f \circ f \circ \cdots f(z_0)| < 2$, the number of compositions being twenty, and $f(z) = z^2 + z_0$. The reconstruction was allowed to form islands due to pinch-off.

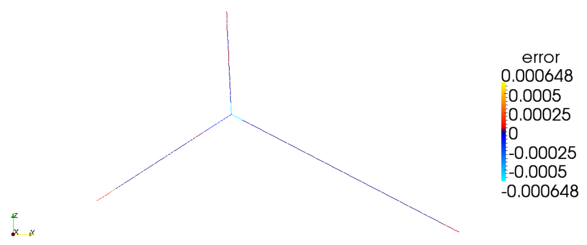


Figure 5.8: **Triple point.** First defined on the interval $[-1, 1]$, with rays at angles $\pi/2, 7\pi/6, 11\pi/6$, then rotated 1 radian, then translated 0.5 units left.

5.3.2 Multiple phase

Name	Grid	Num phases	Mixed	Time	Average error
Triple Point	20^2	3	8.0%	0.8 s	4.1×10^{-5}
Quadruple Point	20^2	4	13.0%	1.1 s	0.00019
Quadruple Point	100^2	4	2.8%	18.7 s	3.51×10^{-5}
Voronoi	20^2	10	27.0%	3.0 s	0.00017
Large Voronoi	200^2	40	6.8%	428.8 s	0.00017
Voronoi swirl	20^2	10	30.0%	3.2 s	0.0012
Voronoi swirl	100^2	10	7.2%	49.1 s	0.00033
Circle/Square (fine)	20^2	3	8.0%	3.3 s	0.00071
Circle/Square, rotated	20^2	3	7.3%	1.2 s	0.00075

Table 5.4: Data from the algorithm applied to various multi-phase two-dimensional test cases.

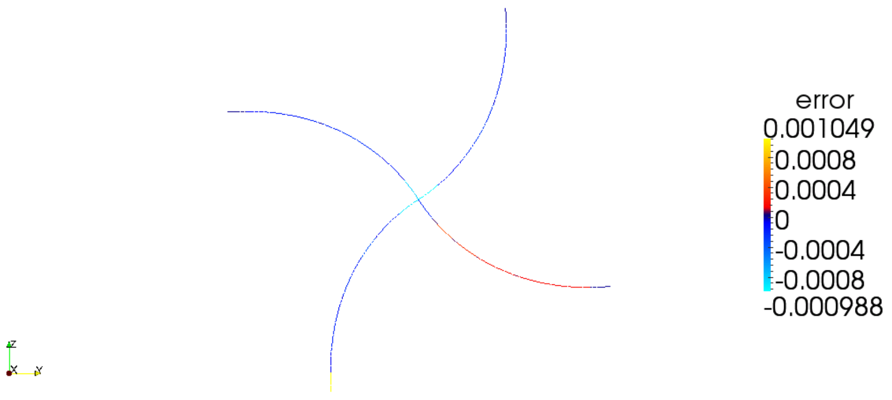


Figure 5.9: **Quadruple point.** First defined on the interval $[-1, 1]$, as rays at angles $0, \pi/2, \pi, 3\pi/2$ from the origin. The swirl is done by the tranform $(r, \theta) \mapsto (r, \theta - r\frac{\pi}{6})$.

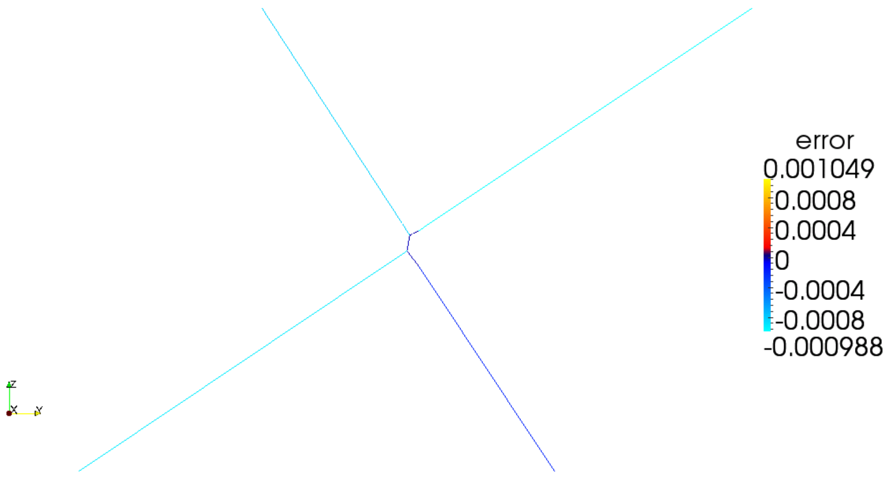


Figure 5.10: **Quadruple point.** Close up of quadruple point. The reconstructed topology is that of two nearby triple points.

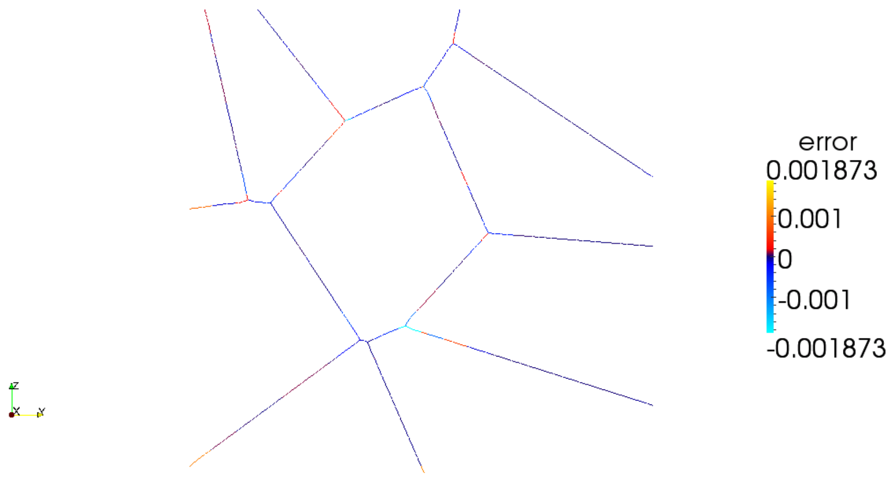


Figure 5.11: **Voronoi.** Reconstruction of a random Voronoi diagram with 10 components.

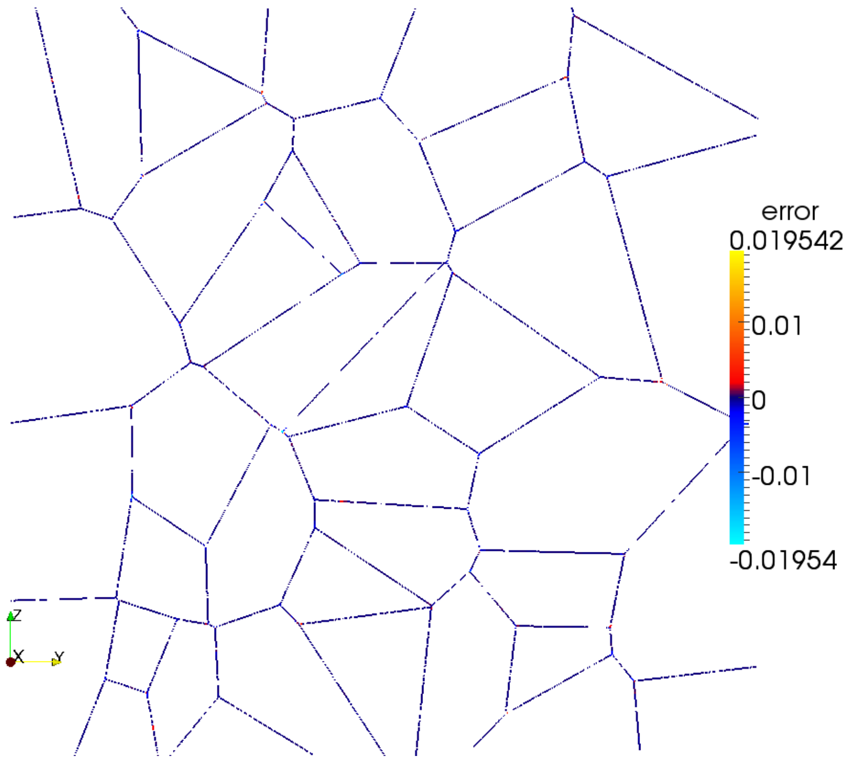


Figure 5.12: **Large voronoi.** Reconstruction of a random Voronoi diagram with 40 components.

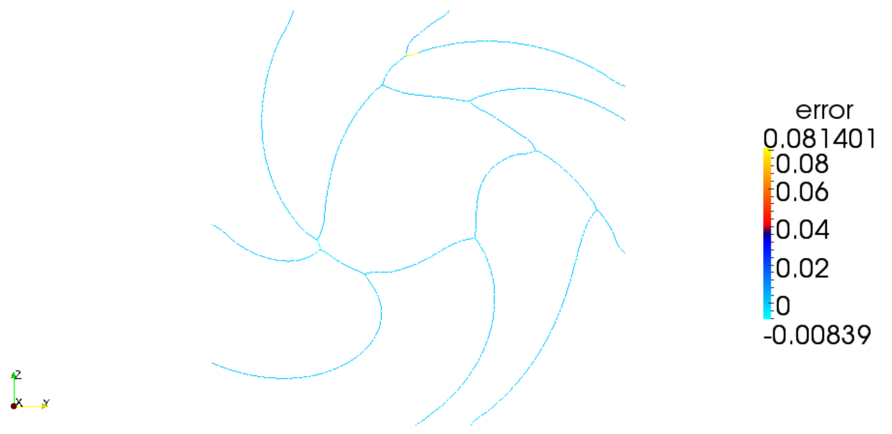


Figure 5.13: **Voronoi swirl.** Starting with the same voronoi diagram as before, on the interval $[-1,1]$, and applying the transform $(r,\theta) \mapsto (r, \theta + \pi r)$.

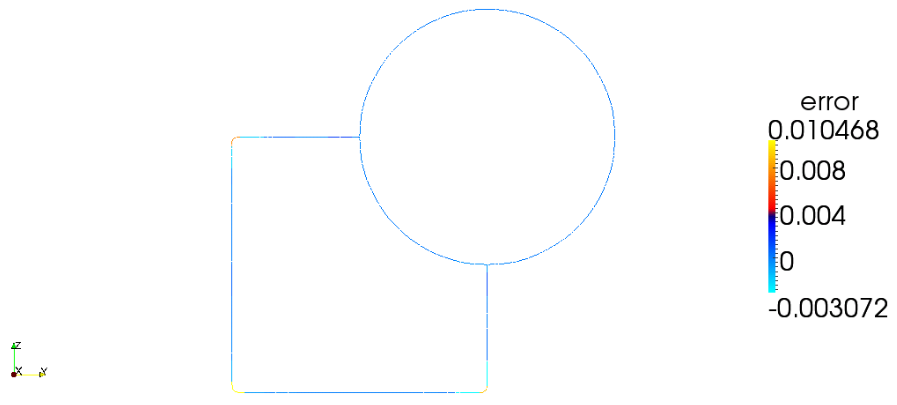


Figure 5.14: **Circle/Square.** On the unit interval, a circle of radius 0.2 centered at (0.6, 0.6), and a square from (0.2, 0.2) to (0.6, 0.6). Where they overlap, the circle takes priority. This shape aligns on the grid. The fine to coarse ratio used was $s = 8$ to be able to resolve the corners.

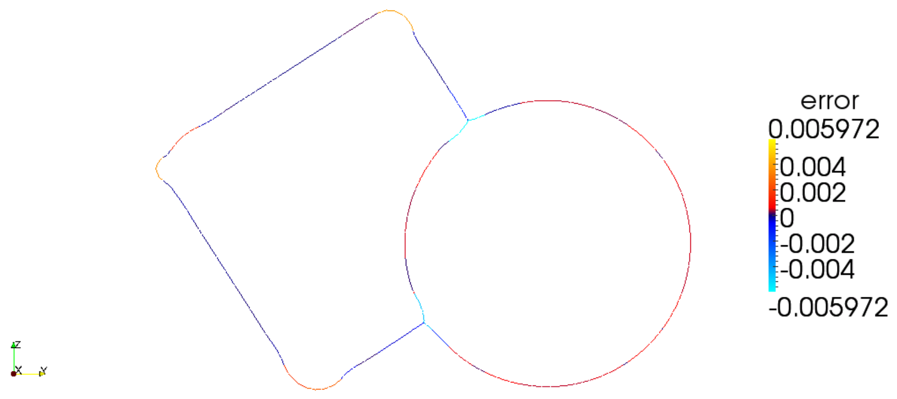
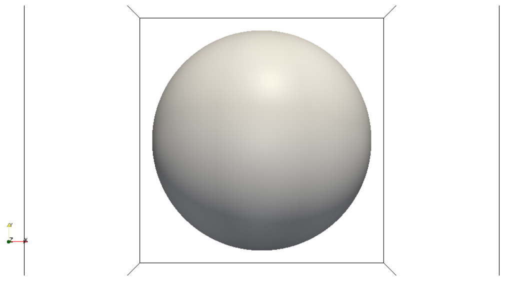
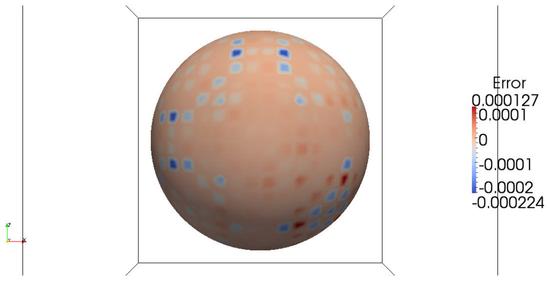


Figure 5.15: **Circle/Square rotated.** Based off of the model in Figure 5.14, except rotated one radian counterclockwise, so that the volume fraction cells do not align with the square. The fine to coarse ratio used is $s = 4$. The answer differs from the model, as the model itself does not minimize surface area.



(a) Reconstructed, 24^3



(b) Reconstruction error highlighted

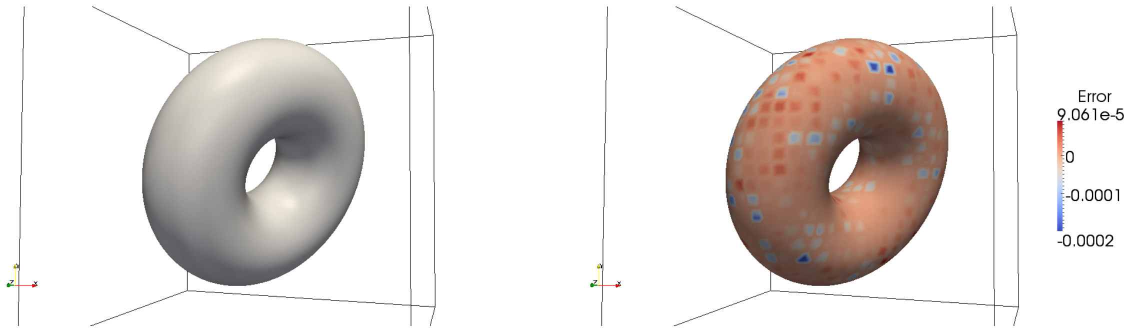
Figure 5.16: **Sphere.** Reconstruction of a sphere of radius 2 on the interval $[-3, 3]$. Reconstruction is indistinguishable from original by naked eye.

5.4 Three-dimensional results

5.4.1 Two phase

Name	Grid	Mixed	Time	Average error
Sphere	24^3	8.4%	0:18	2.84×10^{-5}
Sphere	48^3	4.4%	1:20	1.46×10^{-5}
Torus	24^3	9.3%	0:14	4.11×10^{-5}
Torus	48^3	4.6%	1:14	1.43×10^{-5}
Ellipsoid	20^3	8.7%	0:19	3.78×10^{-5}
Deltoid-like	48^3	1.4%	0:34	0.0019
Bell pepper	24^3	10.7%	0:15	0.0047
Bell pepper	48^3	5.5%	1:27	0.0028
Cube	48^3	4.9%	2:36	0.0076
Sphere minus cylinder	48^3	5.5%	1:44	0.0028
Double cone	48^3	3.3%	1:14	0.011

Table 5.5: Data from the algorithm applied to various single-phase three-dimensional test cases.



(a) Reconstructed, 24^3 (b) Reconstruction error highlighted

Figure 5.17: **Torus.** Reconstruction of a torus, defined as $T = \{x : d(x, A) \leq 0.9\}$, where $A = \{x' : d(x', 0) = 1.5\}$. As with the sphere and other shapes with low curvature, reconstruction error and model error is quite small. The entire region then rotated by $(\pi/5, \pi/5, 0)$ along the coordinate axes – that is, first rotated $\pi/5$ around the x-axis, then $\pi/5$ around the y-axis, and no rotation along the z-axis.

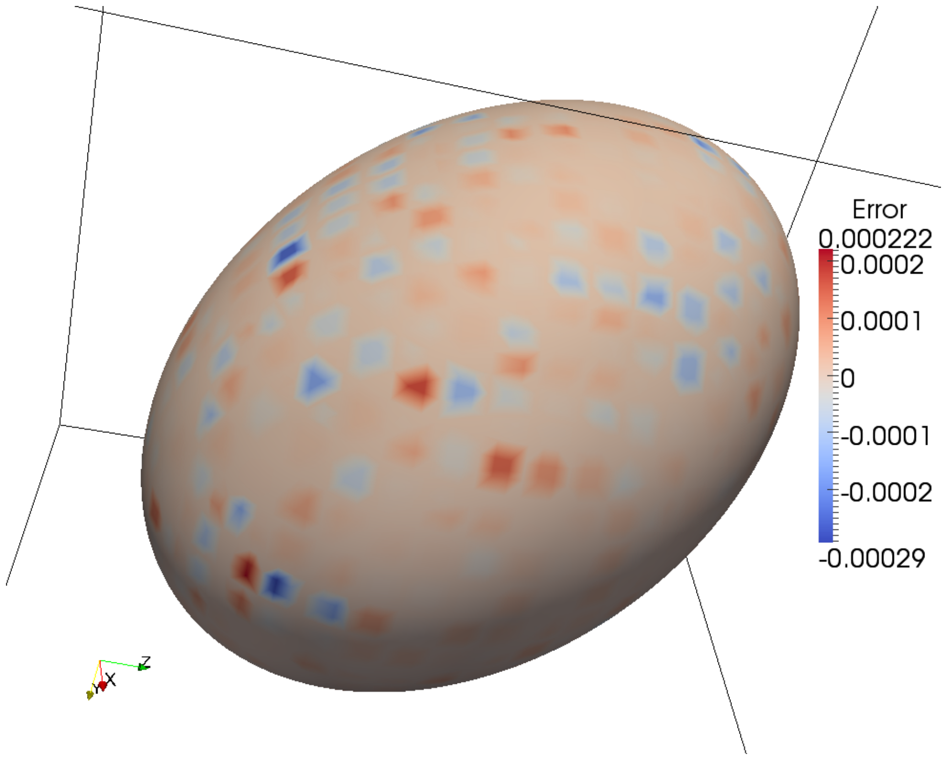
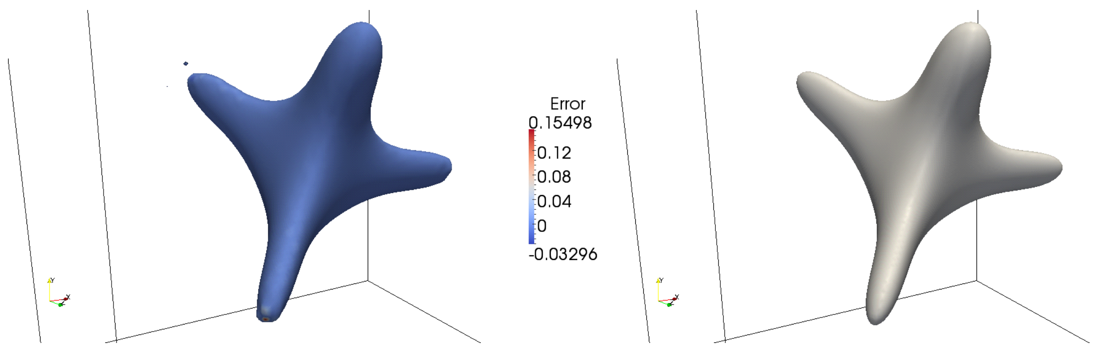


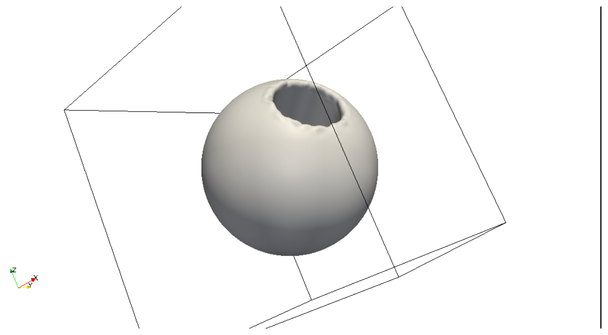
Figure 5.18: Ellipsoid with semiaxes 2.4, 1.6, 2.0, and rotated by the same angles as the torus.

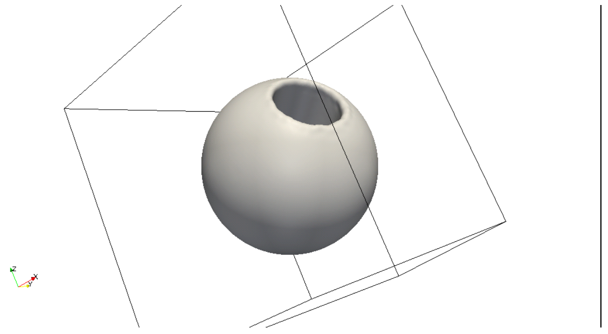


(a) Reconstructed with error colored, 48^3

(b) Model

Figure 5.19: **Deltoid-like.** Reconstruction of a deltoid-like object, defined as $0 = r^4 + 18a^2r^2 - 27a^4 - 8a(x^3 + y^3 - 3xz^2) - 1, a = \frac{1}{2}$. Reconstruction is very accurate away from the tips. The tips, though smooth, have curvature quite high and end up truncated.


(a) Reconstructed, 48^3


(b) Loose reconstruction, 48^3

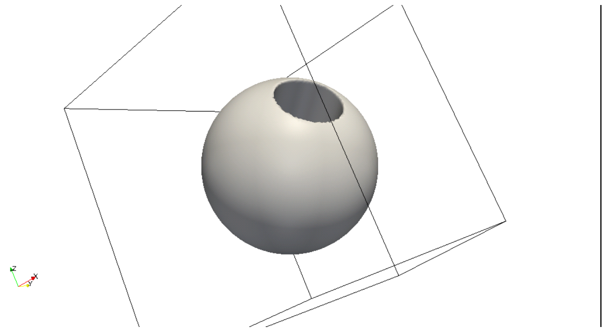
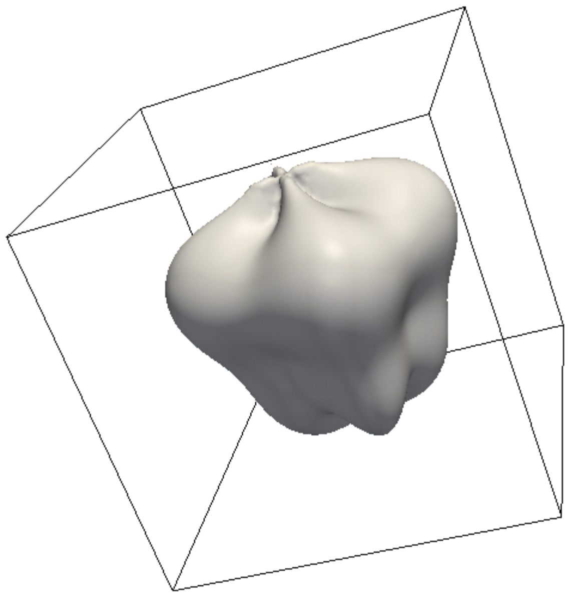
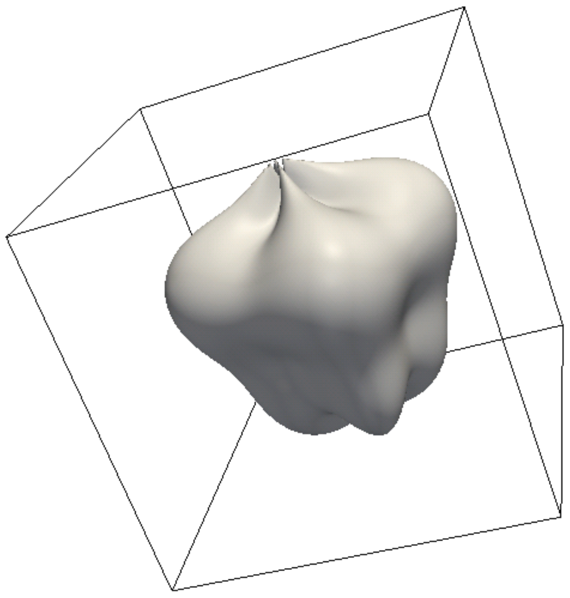

(c) Model

Figure 5.20: **Sphere minus cylinder.** A sphere, radius 2.0 (within the space $[-3,3]^3$), with the cylinder $x^2 + y^2 \leq 0.75$ cut out. The entire region then rotated by $(\pi/5, \pi/5, 0)$ along the coordinate axes. Length minimization leads to oscillations along the sharp edges. Our first stage reconstruction yields a shape without these effects, but with more error (the trade off not well-defined; see section 5.5 instead.)

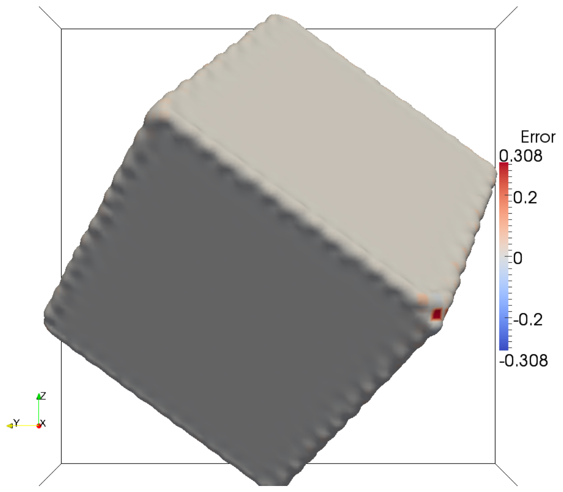


(a) Reconstructed, 48^3

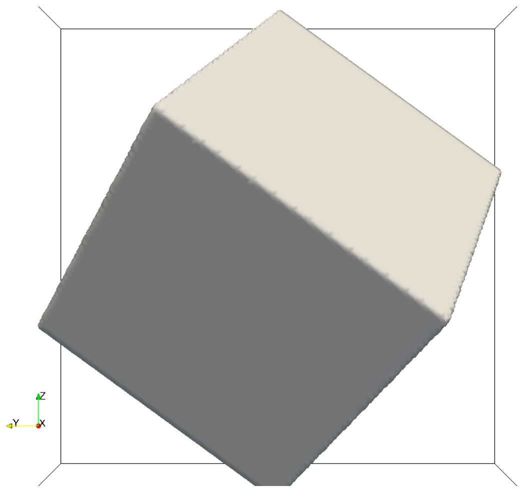


(b) Exact

Figure 5.21: **Bell pepper.** Reconstruction is effective except at the cusps, which get truncated. The model solution comes from: $0 = \rho + \frac{1}{4}(\sin(4\theta) + \sin(4\phi)) - 2.0$, in spherical coordinates.

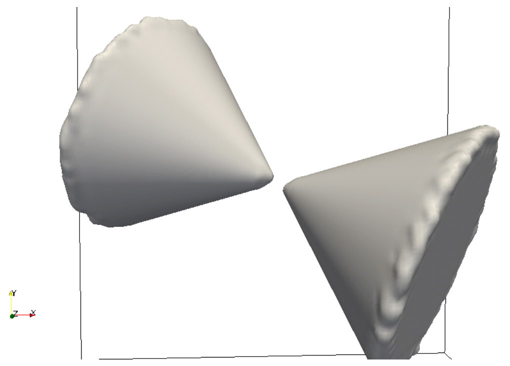


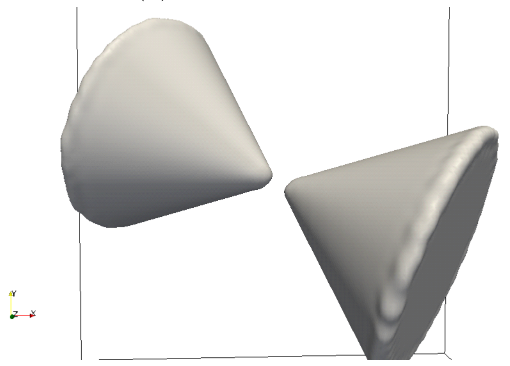
(a) Reconstructed, 48^3



(b) Exact

Figure 5.22: **Cube.** A cube, with side length 3, on the interval $[-3, 3]$. The cube is rotated about the origin by the angles $(\pi/5, \pi/5, 0)$, as done previously. The edges have the same aliasing effect as seen in Chapter 3.


(a) Reconstructed, 48^3


(b) Loose reconstruction, 48^3

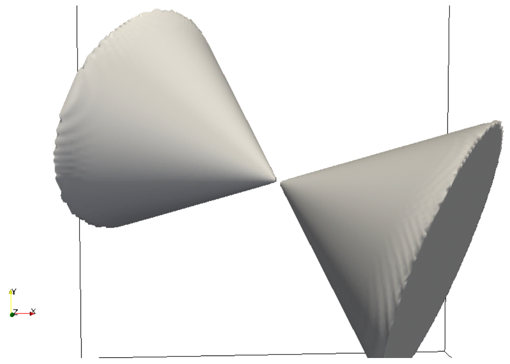
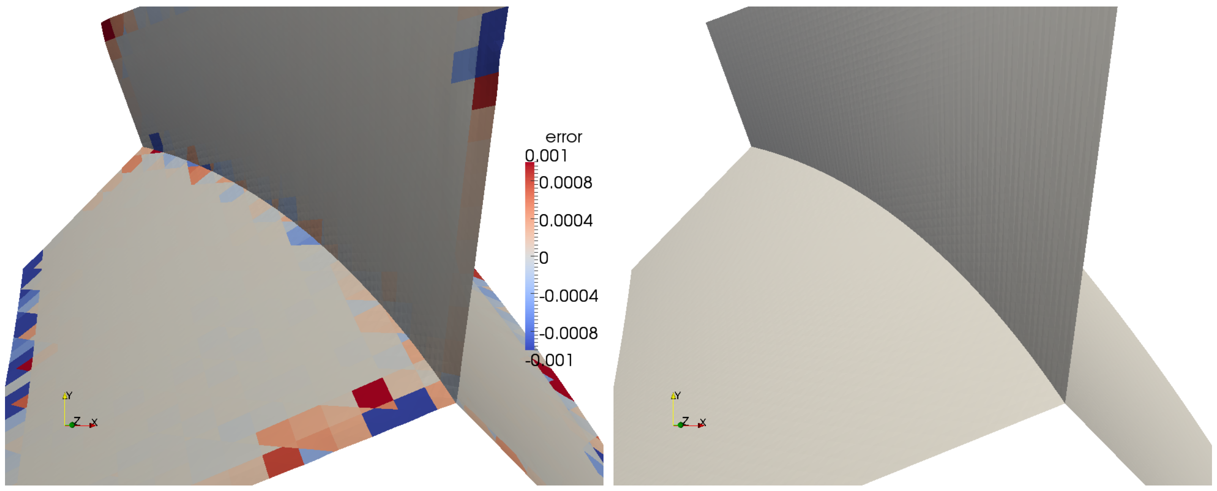

(c) Model

Figure 5.23: **Double cone.** A double cone, defined by $1.96(x^2 + y^2) - z^2 = 0, z > -2.1, z < 2.1$. The entire region then rotated by $(\pi/5, \pi/5, 0)$ along the coordinate axes.

5.4.2 Multiple phase

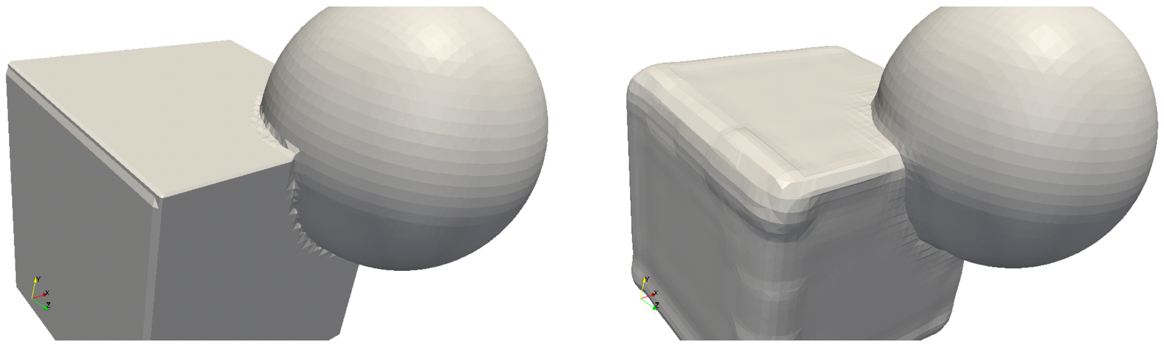
Name	Grid	Num phases	Mixed	Time	Average error
Triple Line	24^3	3	11%	1:19	0.00021
Triple Line	40^3	3	6.7%	4:27	0.00013
Quadruple Point	24^3	4	12.9%	1:35	0.00045
Quadruple Point	40^3	4	7.9%	6:22	0.00030
Concentric Spheres	13^3	6	58.2%	1:30	0.00040
Voronoi	24^3	8	21.2%	2:48	0.0012
Voronoi	40^3	8	13.0%	9:48	0.00078
Voronoi swirl	40^3	8	12.9%	9:49	0.00083
Hemispheres	24^3	3	10.6%	1:18	0.0012
Hemispheres	40^3	3	6.5%	4:32	0.00062
Flat Triple Line	24^3	3	10.4%	1:19	0.00052
Intersecting Spheres	24^3	4	15.1%	1:52	0.00068

Table 5.6: Data from the algorithm applied to various multi-phase three-dimensional test cases.



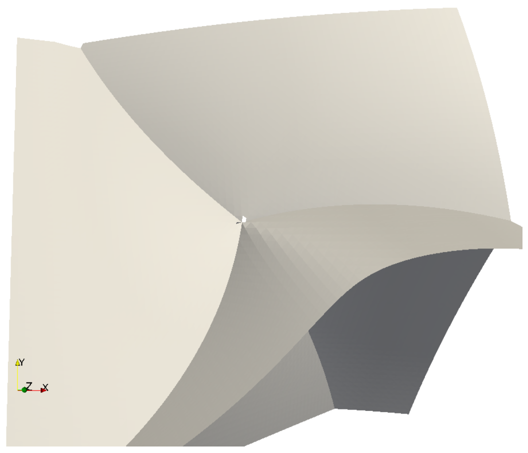
(a) Reconstructed, with error colored in, 24^3 (b) Exact

Figure 5.24: **Triple Line.** Reconstruction of a 120-degree three material junction. The model, specifically: on the region $[-1, 1]^3$, first, voronoi regions created with the points $(0, 0.5, z), (\pm\sqrt{3}/4, -0.25, z)$, translated (0.11, 0, 0), then deformed under $(r, \theta, z) \mapsto (r, \theta + z/4, z)$

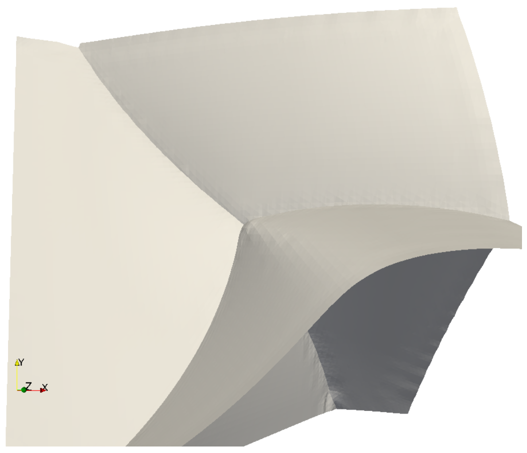


(a) Model (b) Reconstructed, 24^3

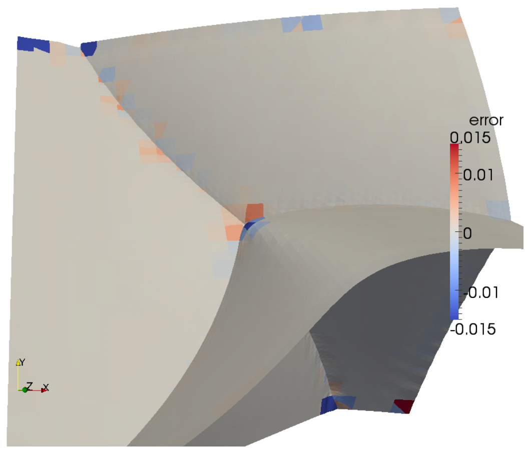
Figure 5.25: **Sphere/Box.** See [3]. Within $[0, 1]^3$, the sphere is centered at (0.6, 0.6, 0.6, with radius 0.2; the cube is from (0.2, 0.2, 0.2) to (0.6, 0.6, 0.6), and is only defined where the sphere is not. The third material is empty space. The average error reported is over 1%, however, as we can see, most of the error is in the reconstruction of the cube itself. The sphere is reconstructed as a sphere, and the junction between the cube and sphere is a smooth curve.



(a) Model

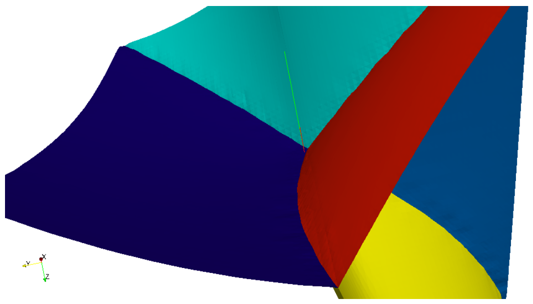


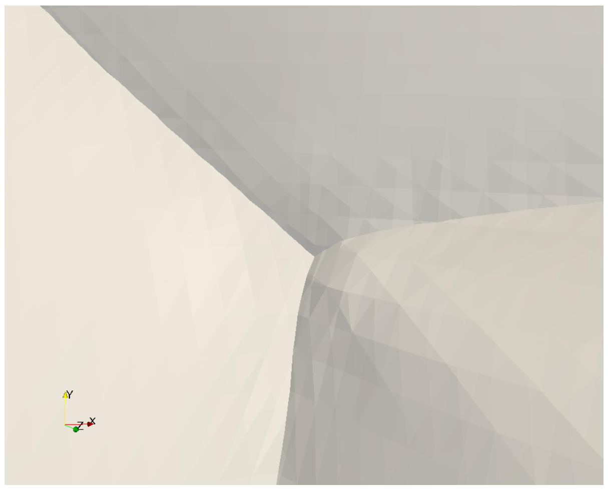
(b) Reconstruction

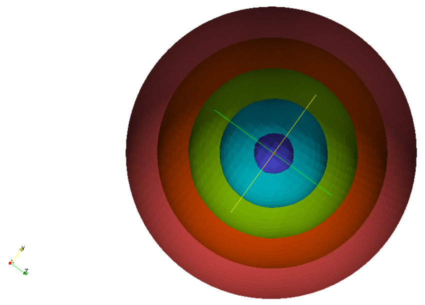


(c) Error

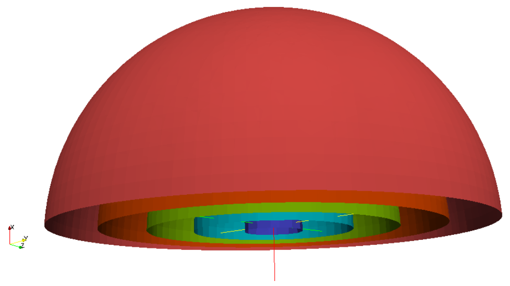
Figure 5.26: **Quadruple Point.** Four point junction, defined as the Voronoi region between the points (0,0,0), (1,1,0), (0,1,1), (1,0,1), on the interval $[0,1]^3$. It is then deformed by $(r,\theta,z) \mapsto (r,\theta+(z-0.5)/4+0.5r,z)$ where the origin in cylindrical coordinates is the point (0.5, 0.5, 0).


(a) Colors showing four regions.

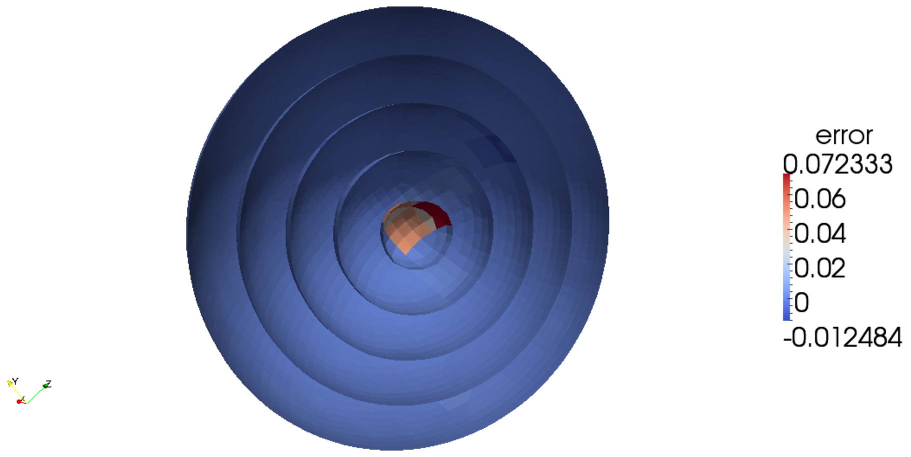

(b) Close up of junction.



(a) View of interior

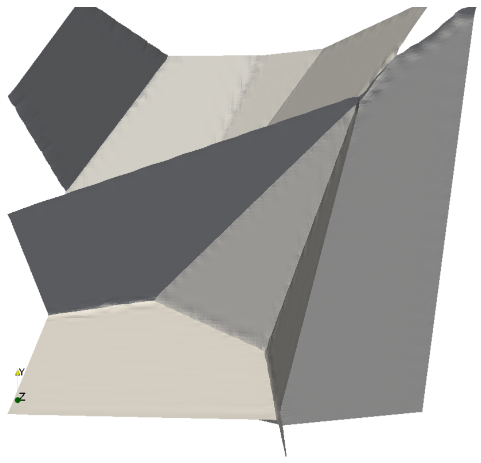


(b) View of exterior (largest sphere)

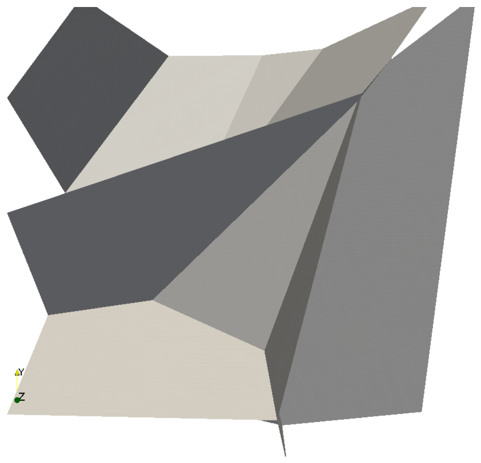


(c) Error plot

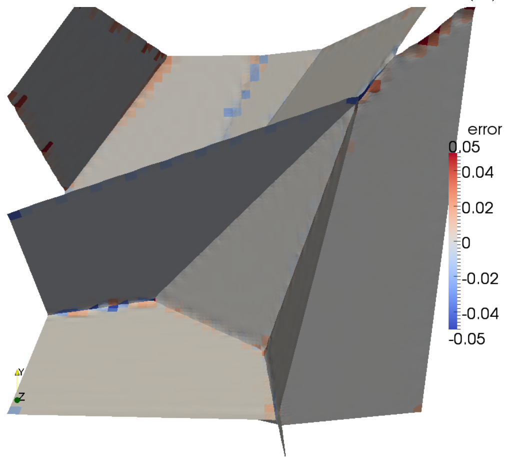
Figure 5.27: **Concentric Spheres.** Five concentric spheres are reconstructed. Within the domain $[-1/2, 1/2]^3$, the sphere radii are $1/13, 2.25/13, 3.5/13, 4.75/13, 6/13$. This is a test case from [3]. The error is primarily in the smallest sphere. This is not surprising; the algorithm is known to be effective at reconstructing spheres individually. The smallest sphere, however, is underresolved by the volume fraction grid. In order to aid convergence, the time step was artificially adjusted after several steps.



(a) Reconstruction

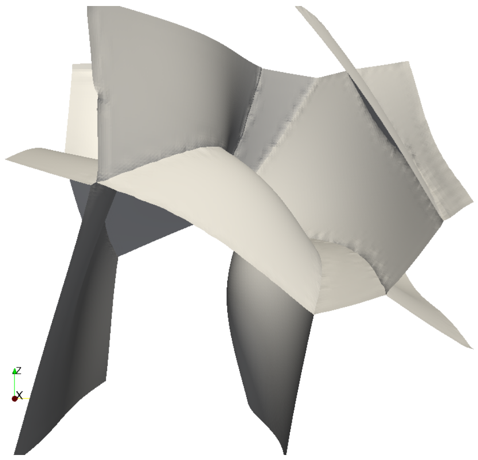


(b) Model

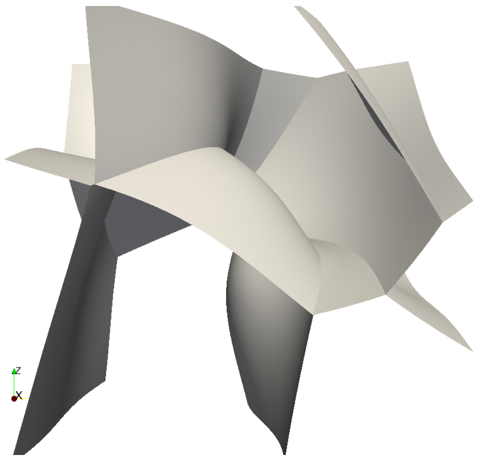


(c) Error plot

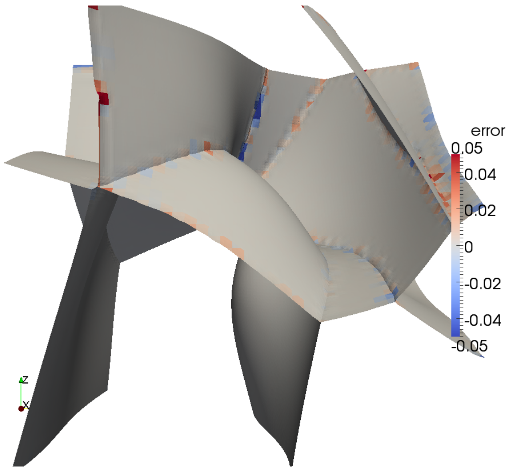
Figure 5.28: **Voronoi.** A random Voronoi diagram with 8 components in 3D.



(a) Reconstruction

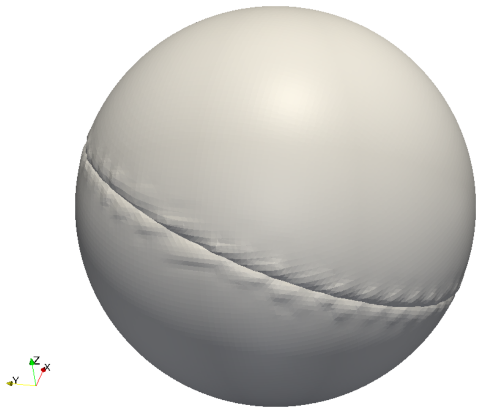


(b) Model

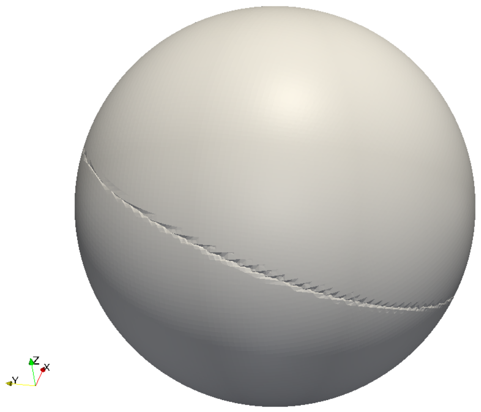


(c) Error plot

Figure 5.29: **Voronoi.** The same Voronoi diagram as previous, except rotated by the angles $(\alpha, 0, \gamma)$ about the axes, with $\alpha(r) = \pi r \exp(-r), \gamma(r) = \pi r \exp(-1.2r)$.



(a) Reconstruction



(b) Model

Figure 5.30: **Hemispheres.** There are three materials: the northern hemisphere, southern, and outer space. The structure rotated along the coordinate axes by $(\pi/5, \pi/5, 0)$.

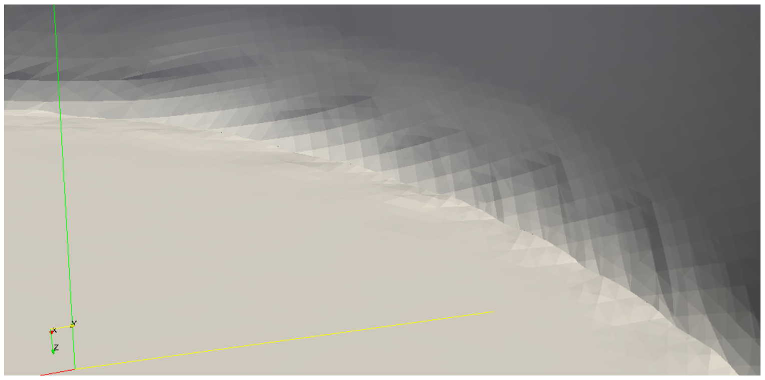
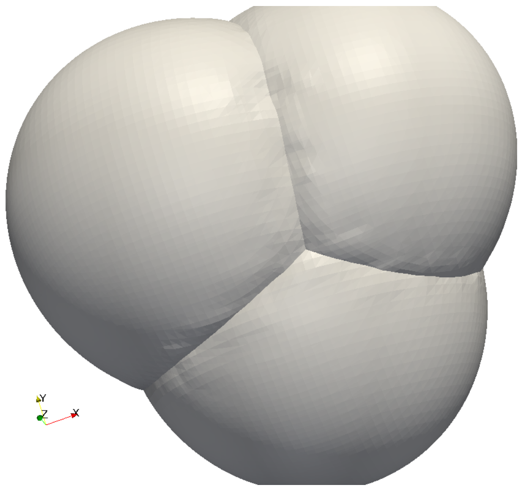


Figure 5.31: Interior view, to see the boundary between hemispheres.


(a) Reconstruction.

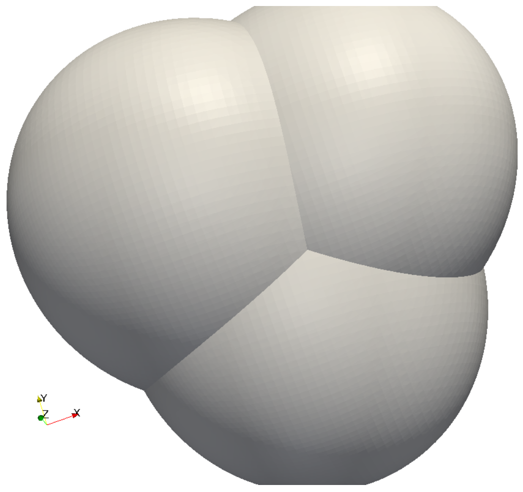
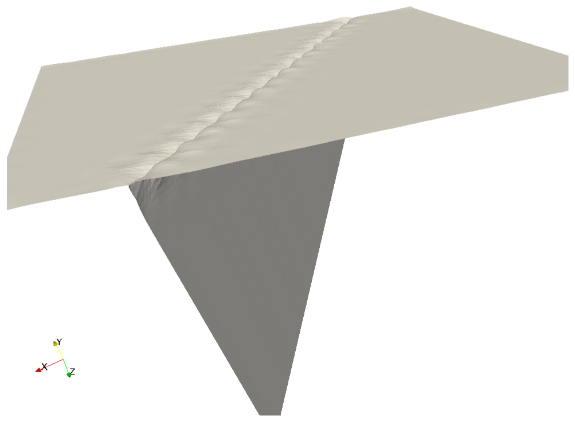
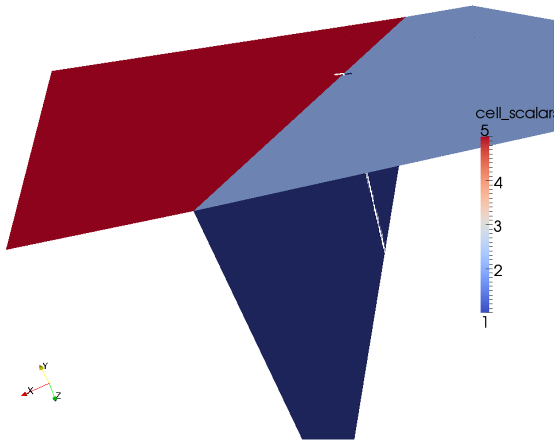

(b) Model.

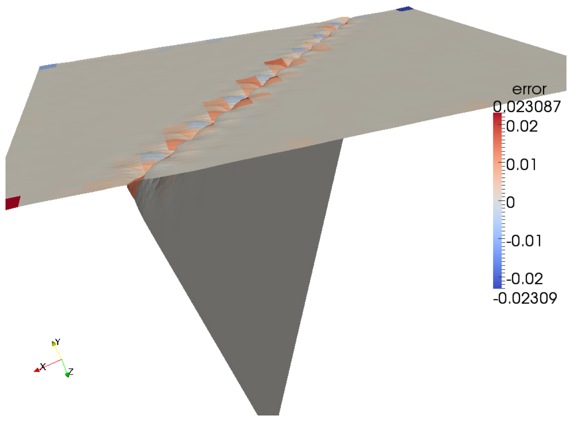
Figure 5.32: **Intersecting Spheres.** Define $d(\mathbf{x}, \mathbf{y})$ to be the distance between points $\mathbf{x}$ and $\mathbf{y}$. Using functions $\phi_0(\mathbf{x}) = d(\mathbf{x}, (0, \frac{1}{2}, 0))$, and $\phi_{1,2}(\mathbf{x}) = d(\mathbf{x}, (\pm\frac{\sqrt{3}}{4}, \frac{1}{4}, 0))$, and the function $f(\mathbf{x}) = \frac{3}{4} - |\mathbf{x}|$, the Model is the boundary between regions where one of the functions is minimal.



(a) Reconstruction.



(b) Model.



(c) Error plot.

Figure 5.33: **Flat Triple Line.** The goal is to create a triple line as a T junction. Using functions $\phi_0(x) = y$, $\phi_1(x) = -y$, $\phi_2(x) = 100z$, we create a T junction where the top of the T is curved very slightly. We then rotated by the angles $(\pi/5, \pi/5, 0)$ about the coordinate axes. The algorithm causes some aliasing to occur at the junction.

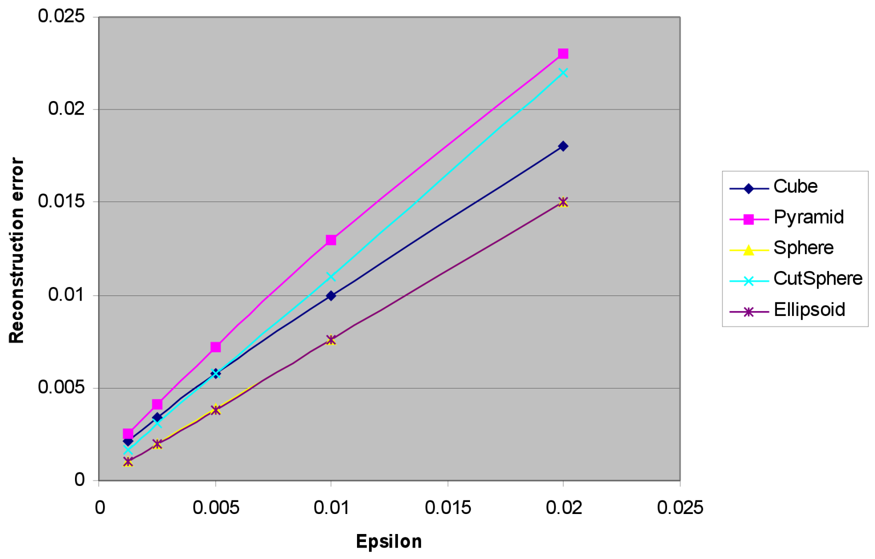


Figure 5.34: All examples run on a 32^3 grid with 3 subdivisions.

5.5 Relaxed flow

We ran the relaxed constraint reconstruction algorithm on the majority of 3D single phase examples. Reconstruction error, empirically, decreases linearly with ϵ, as we can see in Figure 5.34. This agrees with the discussion in Chapter 4.

We also found that, for some sharp cornered shapes, that this optimization algorithm is easier to run to obtain low error solutions than the projection method. For example, the cube with this alternate method yielded an error of 0.0017 compared to 0.0076, with a run time of 2:08 instead of 2:36.

The reconstructions that follow, of a cube (Figure 5.36), pyramid (Figure 5.37), cut sphere (Figure 5.38), and double cone (Figure 5.39), are drawn from the same data as Figure 5.34.

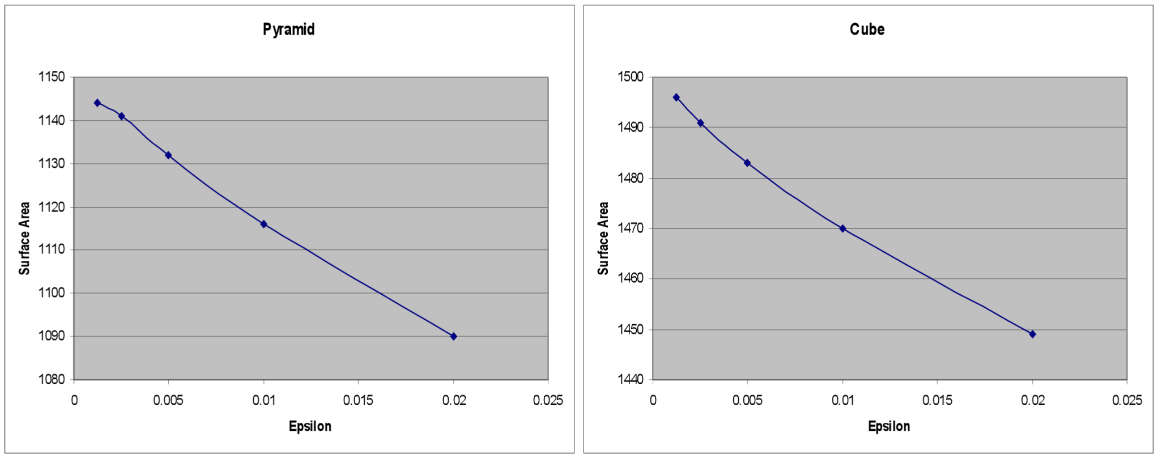
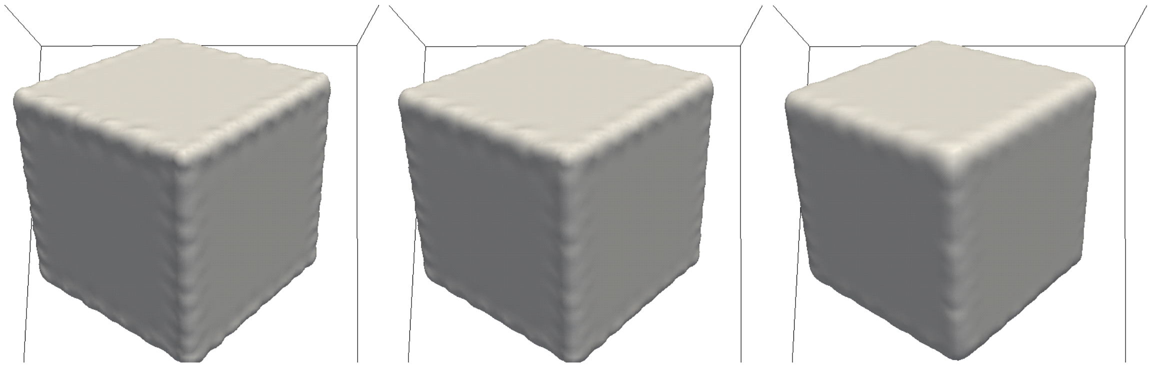
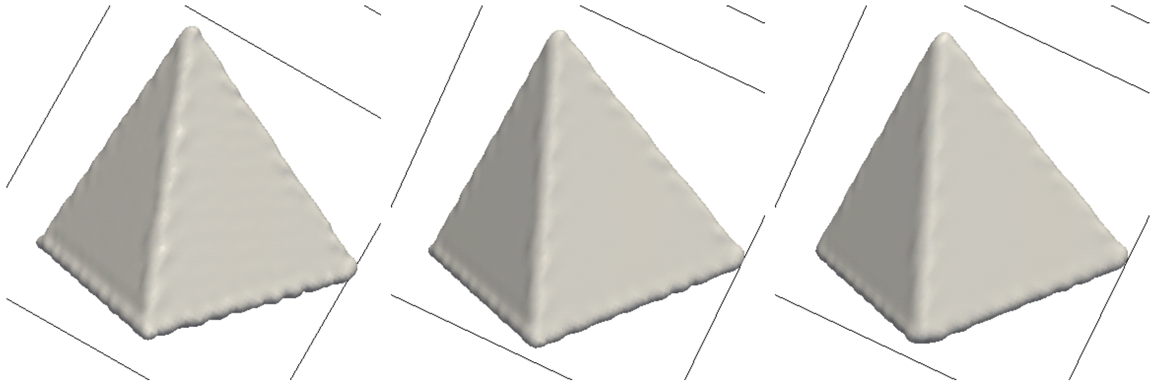


Figure 5.35: Both shapes from the same 32^3 runs. Original areas, in grid units, 1174.3 and 1524.



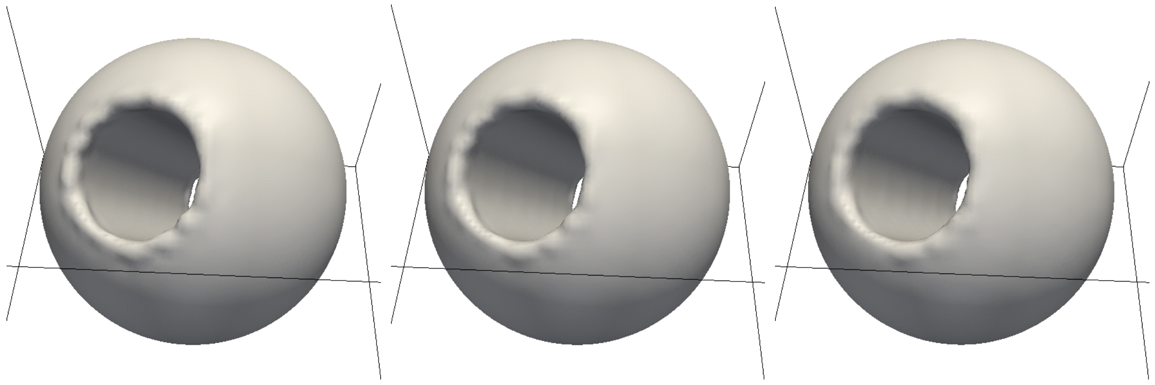
(a) $\epsilon = 0.00125$ (b) $\epsilon = 0.005$ (c) $\epsilon = 0.02$

Figure 5.36: A range of cube reconstructions.



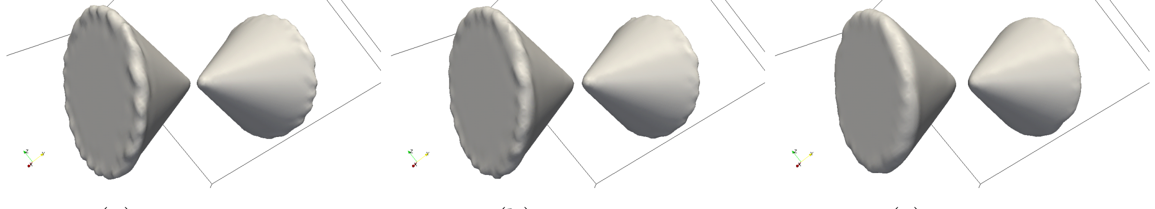
(a) $\epsilon = 0.0008$ (b) $\epsilon = 0.004$ (c) $\epsilon = 0.01$

Figure 5.37: A range of pyramid reconstructions.



(a) $\epsilon = 0.00125$ (b) $\epsilon = 0.005$ (c) $\epsilon = 0.01$

Figure 5.38: A range of cut sphere reconstructions.



(a) $\epsilon = 0.00125$ (b) $\epsilon = 0.005$ (c) $\epsilon = 0.02$

Figure 5.39: A range of double cone reconstructions. The error with $\epsilon = 0.00125$ is lower than the example run of the projection method in 5.4.1.

Chapter 6

Conclusions

6.1 Future work

The volume fraction reconstruction algorithm proposed in this thesis can be improved in several ways. In Section 6.1.1, we discuss ways in which to speed up the computation time in running this algorithm, and in Section 6.1.2, we discuss ways in which to improve the quality of reconstructed objects. Finally, in Section 6.2, we summarize the results of this thesis and conclude.

6.1.1 Speed optimization

The following ideas can be studied to reduce the computation time.

Initial approximation

A fast PLIC-based method should be use to generate a close initial shape. For example, LVIRA could be used because it has second order convergence properties.

Step size control

Often taking the largest stable time step for curvature flow leads to too much smoothing of the interface, beyond where the linearized projection can recover in a step. This happens when the curvature within a cell is large.

This problem is minimized by intelligent step size control. For the constraint-free formulation of the problem, we may use a line search method to find a time step satisfying the Wolfe Conditions [15]. This requires evaluating the gradient and the inner product (a surface integral) of the gradient with a search direction.

Fast heaps

Profiling the code shows that computing extension velocities is the bottleneck, which in turn points to heap operations within the Fast Marching Method. Heaps with more than two children have been studied previously to speed up delete operations. [29].

6.1.2 Shape quality

Other energy functionals

We discussed the possibility of minimizing $\int |\Delta u|^2 dx$ in our two-dimensional height function example. Within the framework of a general interface, the analogous flow is the intrinsic laplacian of curvature, $\Delta_{\{\phi(x)=0\}}\kappa$. Study is required, however, for two main reasons:

- Due to the sensitivity of fourth derivatives, the correct numerical method to advance the flow needs to be carefully chosen. Sethian and Chopp [8] try several different methods before they find one that effectively advances the solution.
- A static fluid is subject to surface tension, and so at subgrid resolutions, the boundary would tend to shrink. This motivated us to use a minimum surface area / curvature flow formulation. The laplacian of curvature does not describe surface tension, but may more accurately describe a solid under the influence of Hooke's laws at subgrid resolution. Thus, reconstructed interfaces using such a formulation may be more appropriate for describing deformable solids.

Corner and edge detection

One may assume that if the interface curves significantly (say, 30° in two dimensions) within a single cell, either resolution is insufficient, or there is a loss of smoothness within. It would be then possible to handle corners or edges separately from the main algorithm using heuristic methods.

6.2 Summary

We formulated a solution to the volume fraction reconstruction problem based on minimizing total surface area of the interface while maintaining the prescribed volume fractions.

- We explored an analogous 1D height function formulation where the volume fractions became constraints on the area under the curve. We used this to derive an algorithm based on gradient descent, which projected the gradient of the length functional onto an area-preserving direction.

- We then set up a framework to solve the general case using an interface defined by a level set function. We extended the steepest descent projection algorithm. We then recast our problem as an optimization without a constraint. In this formulation, we seek the minimum of an objective function defined as a weighted sum between surface area and curvature. This allowed us to examine smoother possible solutions to our problem, and in some sharp cornered test cases, to obtain lower error. We finally extended our algorithm toward multiple phase reconstruction via the Voronoi Interface.

- We presented results of the algorithm in two and three dimensions. We found that the algorithm was able to reproduce straight lines, and achieved second order reconstruction accuracy, and first order normal vector approximation. The algorithm was able to reproduce smooth shapes faithfully, as well as the smooth regions within shapes containing edges and corners. We presented results for multiple material reconstruction, and we found our algorithm was able to handle a variety of different cases and topologies robustly.

In conclusion, the minimal surface reconstruction algorithm described in this thesis provides its user with a robust method for recovering arbitrary surfaces from both single and multi-phase volume fraction data. It performs best when the curvature of the expected interface is bounded. When it is not, however, a soft-constraint formulation is available, using the same ideas developed within the text, that provides a free parameter to control the smoothness of the interface.

Bibliography

[1] D. Adalsteinsson and J. A. Sethian. “A Fast Level Set Method for Propagating Interfaces”. In: *Journal of Computational Physics* 118.2 (1995), pp. 269–277.

[2] D. Adalsteinsson and J. A. Sethian. “The Fast Construction of Extension Velocities in Level Set Methods”. In: *Journal of Computational Physics* 148 (1999), pp. 2–22.

[3] J. C. Anderson et al. “Smooth, Volume-Accurate Material Interface Reconstruction”. In: *IEEE Transactions on visualization and computer graphics* 16.5 (2010), pp. 802–814.

[4] S. L. Chan and E. O. Purishima. “A new tetrahedral tesselation scheme for isosurface generation”. In: *Computers and graphics* 22 (1998), pp. 83–90.

[5] S. Chen et al. “A Simple Level Set Method for Solving Stefan Problems”. In: *Journal of Computational Physics* 138 (1997), pp. 8–29.

[6] D. L. Chopp. “Computing Minimal Surfaces via Level Set Curvature Flow”. In: *Journal of Computational Physics* 106 (1993), pp. 77–91.

[7] D. L. Chopp. “Some Improvements of the Fast Marching Method”. In: *SIAM Journal of Scientific Computing* 23.1 (2001), pp. 230–244.

[8] D. L. Chopp and J. A. Sethian. “Motion by Intrinsic Laplacian of Curvature”. In: *Interfaces and Free Boundaries* 1 (1999), pp. 1–18.

[9] A. J. Chorin. “Flame Advection and Propagation Algorithms”. In: *Journal of Computational Physics* 35 (1980), pp. 1–11.

[10] L. C. Evans. *Partial Differential Equations.* Providence, RI: American Mathematical Society, 2010.

[11] A. Gueziec and R. Hummel. “Exploiting Triangulated Surface Extraction using Tetrahedral Decomposition”. In: *IEEE Transactions on Visualisation and Computer Graphics* 1.4 (1995), pp. 328–342.

[12] C. W. Hirt and B. D. Nichols. “Volume of Fluid (VOF) method for the dynamics of free boundaries”. In: *Journal of Computational Physics* 39 (1981), pp. 201–225.

[13] Y. Jung, K. T. Chu, and S. Torquato. “A variational level set approach for surface area minimization of triply-periodic surfaces”. In: *Journal of Computational Physics* 223 (2007), pp. 711–730.

[14] R. Malladi, J. A. Sethian, and B. C. Vemuri. “Shape Modeling with Front Propagation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.2 (1995), pp. 158–175.

[15] J. Nocedal and S. J. Wright. *Numerical Optimization*. New York: Springer Verlag, 2006.

[16] W. Noh and P. Woodward. “A Simple Line Interface Calculation”. In: *Proceedings, Fifth International Conference on Fluid Dynamics*. Ed. by A. I. van de Vooran and P. J. Zandberger. Springer-Verlag, 1976.

[17] S. Osher and J. A. Sethian. “Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations”. In: *Journal of Computational Physics* 79 (1988), pp. 12–49.

[18] B. J. Parker and D. L. Youngs. *Two and three dimensional Eulerian simulation of fluid flow with material interfaces*. Tech. rep. 0192. Aldermaston, Berkshire: UK Atomic Weapons Establishment, 1992.

[19] B. A. Payne and A. W. Toga. “Surface Mapping Brain Functions on 3D Models”. In: *IEEE Computer Graphics and Applications* 10.2 (1990), pp. 41–53.

[20] J. E. Pilliod Jr. “An Analysis of Piecewise Linear Interface Reconstruction Algorithms for Volume-of-Fluid Methods”. MA thesis. University of California, Davis, 1992.

[21] J. E. Pilliod Jr. and E. G. Puckett. “Second-order accurate volume-of-fluid algorithms for tracking material interfaces”. In: *Journal of Computational Physics* 199 (2004), pp. 465–502.

[22] E. G. Puckett. “A volume-of-fluid interface tracking algorithm with applications to computing shock wave refraction”. In: *Proceedings of the Fourth International Symposium on Computational Fluid Dynamics*. Ed. by H. Dwyer. Davis, CA, 1991, pp. 933–938.

[23] C. Rhee, L. Talbot, and J. A. Sethian. “Dynamical Study of a Premixed V Flame”. In: *Journal of Fluid Mechanics* 300 (1995), pp. 87–115.

[24] R. I. Saye and J. A. Sethian. “Analysis and Applications of the Voronoi Implicit Interface Method”. In: *Journal of Computational Physics* 231.18 (2012).

[25] J. A. Sethian. “A Fast Marching Level Set Method for Monotonically Advancing Fronts”. In: *Proceedings of the National Academy of Sciences* 93.4 (1996), pp. 1591–1595.

[26] J. A. Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1999.

[27] J. A. Sethian and J. D. Strain. “Crystal Growth and Dendritic Solidification”. In: *Journal of Computational Physics* 98 (1992), pp. 231–253.

[28] M. Sussman and E. Fatemi. “An Efficient, Interface-Preserving Level Set Redistancing Algorithm and its Application to Interfacial Incompressible Fluid Flow”. In: *SIAM Journal of Scientific Computing* 20.4 (1999), pp. 1165–1191.

[29] M. A. Weiss. *Data Structures and Algorithm Analysis (2nd ed.)* Addison-Wesley, 2007, p. 216.

[30] E. Zeidler. *Applied functional analysis: main principles and their applications.* New York, NY: Springer-Verlag, 1995.