UNIVERSITY OF CALIFORNIA, SAN DIEGO

An Interface-Fitted Finite Element Based Level Set Method: Algorithm, Implementation, Analysis and Applications

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

in

Mathematics

by

John P. Shopple

Committee in charge:

Professor Bo Li, Chair
Professor Li-Tien Cheng
Professor Olga K. Dudko
Professor Michael Holst
Professor Daniel M. Tartakovsky

2009

The dissertation of John P. Shopple is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2009

EPIGRAPH

It is by going down into the abyss
that we recover the treasures of life.
Where you stumble, there lies your treasure.
The very cave you were afraid to enter
turns out to be the source of what you were looking for.
— Joseph Campbell

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I give many, many thanks my adviser, Bo Li. I was always amazed by his incredible patience with me. I am grateful for his financial support as a graduate research assistant. Without his confidence in me, I may not have been able to complete my thesis.

I would also like to thank my committee members, Li-Tien Cheng, Olga Dudko, Michael Holst, and Daniel Tartakovsky, for their willingness to serve on my committee.

VITA

2002	B. S. in Mathematics, *summa cum laude*, Temple University
2005	M. A. in Mathematics, University of California, San Diego
2003–2005	Summer Research Assistant, University of California, San Diego
2002–2006	Teaching Assistant, University of California, San Diego
2006–2008	Research Assistant, University of California, San Diego
2009	Ph. D. in Mathematics, University of California, San Diego

ABSTRACT OF THE DISSERTATION

An Interface-Fitted Finite Element Based Level Set Method: Algorithm, Implementation, Analysis and Applications

by

John P. Shopple
Doctor of Philosophy in Mathematics
University of California San Diego, 2009
Professor Bo Li, Chair

Simulation of problems involving different media that are flowing or deforming requires tracking the boundary between the media. These are called moving interface problems. Often the velocity of the moving interface is determined by some underlying physical model. Thus, moving interface problems usually require separate methods to track the interface and to determine the velocity of the interface.

The level set method has become a popular way of numerically tracking a moving interface. This method represents the interface implicitly as the zero level set of a higher dimensional continuous function called ϕ, and this function is evolved in time by the partial differential equation $\partial_t \phi + V_{\mathbf{n}} |\nabla \phi| = 0$, known as the level set equation. The interface normal velocity $V_{\mathbf{n}}$ must be determined concurrently by some separate method depending on the specific problem.

Finite difference methods are well established for implementing the level set method, but finite difference methods are not ideally suited for solving problems involving arbitrarily shaped regions such as those that occur in moving interface problems. This is because they require a regular spaced grid that does not explicitly locate the interface. Thus, they cannot easily solve problems that depend on knowing the exact location of the interface.

To address these difficulties, an interface-fitted finite element level set method

is considered. This method uses a base mesh to solve the level set equation and track the interface. The base mesh is refined at each time step to explicitly locate the interface and separate regions defined by the interface. A finite element method can then be used to solve field equations on the refined mesh.

Using the interface-fitted mesh, a new method of reinitializing of the level set function, a new method of extending the velocity away from the interface, and a new method of calculating curvature are proposed.

The interface-fitted finite element level set method is applied to modeling solidification problems and determining optimal solute-solvent interfaces of solvation systems.

Chapter 1

Introduction

1.1 Moving Interfaces and Their Mathematical Descriptions

Moving interfaces are boundaries that separate different media that are deforming or flowing. They occur commonly in science, engineering and daily life. For instance, an ice-water boundary moves during the change of temperature and an oil bubble moves in water. More examples include growing crystal surfaces, phase boundaries in solid-solid phase transformations such as precipitate and martensite interfaces, domain boundaries that separate different parts of material such as grain boundaries in polycrystals, domain walls in ferromagnetic materials, two phase fluid flows, and surfaces of large biomolecules moving in water. Here interfaces are understood in a broad sense: an interface can be a geometrical surface that has no thickness—a sharp interface; it can also mean a diffuse interface that can have certain thickness, e.g., of a few atomic diameters. Compared with those of bulk phases, the properties of moving interfaces can be more and more important as the length scales become smaller and smaller. As modern technologies demand heavily that devices be much smaller in scale, the study of moving interfaces has become more and more important.

A moving interface can be mathematically described by a sharp interface, i.e., a surface without any thickness, or by a field function that takes constant

values in different regions with a sharp but continuous transition from one region to another, representing an interface. The latter description is often called diffused interface description. In this thesis, we will consider a sharp interface description of moving interfaces.

Consider a moving interface which is represented by the set of points $\Gamma = \Gamma(t)$ which depends on time t. Let $\mathbf{x}(t)$ be an arbitrary point that remains on Γ. The velocity (vector) of such a point is defined by

$$\mathbf{V}(t) = \frac{d\mathbf{x}(t)}{dt}. \tag{1.1.1}$$

The interface motion is then determined by the normal velocity (the normal component of the velocity vector) at each point of the interface. Equations that determine the normal velocities are often called motion laws for moving interfaces. Thus, moving interfaces are mathematically determined by certain motion laws governing the normal velocity of the interface. In general, motion laws are given from the underlying physics, chemistry, etc. Motion by mean curvature is a common and important example of such motion laws. If we denote by $\Gamma = \Gamma(t)$ the geometrical surface at time t, then the motion law is

$$V_{\mathbf{n}} = -H,$$

where $V_{\mathbf{n}} = V_{\mathbf{n}}(\mathbf{x}, t)$ denotes the normal velocity (the normal component of the velocity vector) of the point $\mathbf{x} \in \Gamma$ at time t, and H is the mean curvature.

In many applications, normal velocity also involves more physical quantities. Therefore partial differential equations (PDEs) typically must be satisfied on either side of the interface for certain quantities such as the temperature field. In such a case, the movement of the interface is then described in terms of the relationship between the solutions of the PDEs on either side of the interface. As the interface continually evolves the PDEs must be solved on continually evolving domains. The interface may be a complex shape and can change topologically over time by merging or breaking apart. The challenge for computational scientists is to accurately solve such problems.

1.2 Different Types of Numerical Methods for Interface Motion

Let us imagine two regions separated by an interface that is moving in time according to a certain motion law. In two dimensions the interface would be a curve separating two areas. In three dimensions the interface would be a surface separating two volumes. We would like to track this moving interface numerically. We can designate an interior region and exterior region and call them Ω_- and Ω_+. The interface separating Ω_- and Ω_+ is designated by Γ, and the boundary of the computational domain is designated by $\partial\Omega$ (see Figure 1.1).

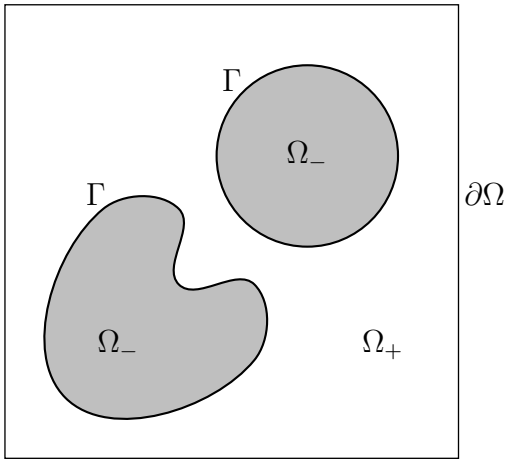
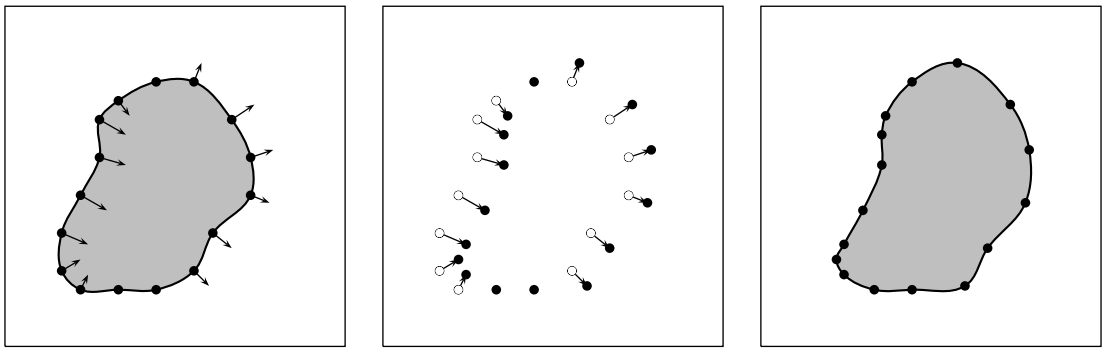


Figure 1.1: A typical domain Ω with boundary $\partial\Omega$. Interface Γ shown separating regions Ω_- and Ω_+.

1.2.1 Marker Method

The marker method is a Lagrangian method that discretizes the interface with a set of points located on the interface. In two dimensions the points are chained together with line segments to form a curve. To move the interface, at each of these marker points the velocity is determined at each time step. The points are then moved forward in time a small amount (see Figure 1.2).

One drawback of this method is that topological changes are difficult to handle. A sophisticated method must therefore be employed to check if two regions of Ω_- join together or if one region splits into two. Another drawback is that the accuracy of the location of the interface depends on how closely the marker particles are place along the interface. As time progresses, some markers may move farther apart creating more error in the location of the interface near those points (see Figure 1.3). So any algorithm needs to periodically check the placement of the particles to insure that they are sufficiently close together to have the desired accuracy. Also, some marker particles may need to be removed or redistributed if there are too many close together along the interface. Yet another drawback of the marker method is that it can become unstable with curvature based motion of the interface. Small errors in the calculated curvature of the interface will grow in time.

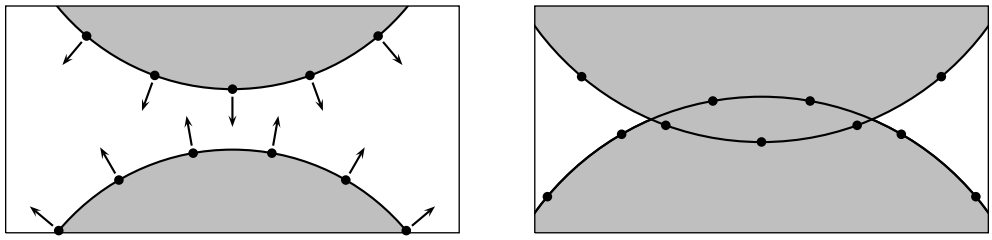


(a) (b) (c)

Figure 1.2: Illustration of the marker method. (a) Markers (dots) represent the interface location. (b) At each time step each marker is moved at its specified velocity to its new location. (c) Interface in its new location.

1.2.2 Volume-of-Fluid Method

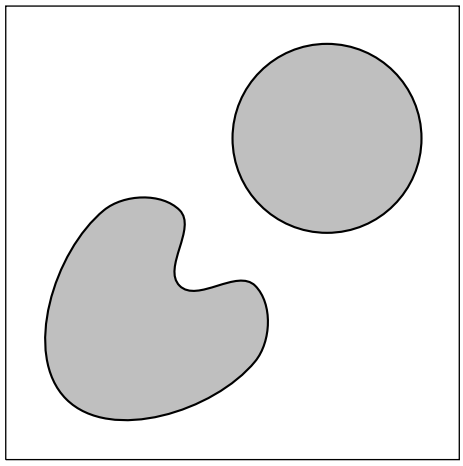
In contrast to the marker method, the volume-of-fluid method tracks the interface implicitly. It uses a grid of cells on the computational domain, and each cell contains a value between 0 and 1. Cells completely contained in Ω_- have a value



(a) (b)

Figure 1.3: One drawback of the marker method. (a) Two interfaces approaching each other. (b) Markers need to be removed after interfaces overlap.

of 1 and those completely contained in Ω_+ have value 0. Cells that the interface passes through contain a value between 0 and 1 based on where the interface passes through that cell. The interface location can then be approximated based on the values in the cells, but many smaller cells may be required to get a desired accuracy (see Figure 1.4). Quantities such as the normal direction and curvature of the interface are difficult to calculate accurately with this method. In spite of the drawbacks, this method has been used to model moving interfaces successfully, in particular, flame front propagation [Cho80, Set84].



0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	.1	.8	1	.8	.1	0
0	0	0	0	0	0	.8	1	1	1	.8	0
0	0	0	0	0	0	1	1	1	1	1	0
0	0	0	0	0	0	.8	1	1	1	.8	0
0	0	.5	.9	.6	0	.1	.8	1	.8	.1	0
0	.3	1	1	.6	0	0	0	0	0	0	0
0	.7	1	1	.8	.6	.6	0	0	0	0	0
0	1	1	1	1	1	.9	0	0	0	0	0
0	.9	1	1	1	1	.5	0	0	0	0	0
0	.2	.9	1	.7	.3	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

(a) (b)

Figure 1.4: Illustration of the volume-of-fluid method. (a) Interfaces enclosing shaded areas. (b) Grid cell values based on proportion of cell filled by shaded areas.

1.2.3 Level Set Method

The level set method, first introduced in [OS88] is an Eulerian method that describes the interface implicitly. The starting point of this method is to describe a moving surface $\Gamma = \Gamma(t)$ at time t implicitly as the zero level set of an auxiliary function $\phi = \phi(\mathbf{x}, t)$, i.e., $\Gamma(t) = \{\mathbf{x} : \phi(\mathbf{x}, t) = 0\}$. For instance, if Γ is a three-dimensional surface (such as a sphere) then the function $\phi(\mathbf{x}, t)$ at any time t is a function defined on the three-dimensional space, i.e., $\mathbf{x}$ has three coordinates. For convenience, we can assume that the function takes on positive values on one side of the interface and negative values on the other side. We shall call ϕ a level set function of Γ. Clearly such functions are vastly nonunique. The level set function evolves over time in such a way that its zero level set always represents the location of the interface. One of the known advantages of the level set method is that it captures naturally, and easily, topological changes of moving interfaces (see Figure 1.5).

The evolution of the level set function is determined by the underlying motion law governing the normal velocity $V_{\mathbf{n}}$ of the moving interface. If the level set function is denoted by $\phi = \phi(\mathbf{x}, t)$ for a point $\mathbf{x}$ on the interface and time t, then the equation determining the level set function is given by

$$\partial_t \phi + V_{\mathbf{n}} |\nabla \phi| = 0.$$

where ∂_t denotes the time derivative and ∇ denotes the spatial gradient. This equation is called the level set equation. One needs to extend the normal velocity $V_{\mathbf{n}}$ (usually determined only on the interface) away from the interface so that the level set equation can be solved in an appropriate space domain. The derivation of the level set equation and more details of the level set method are given in Chapter 2.

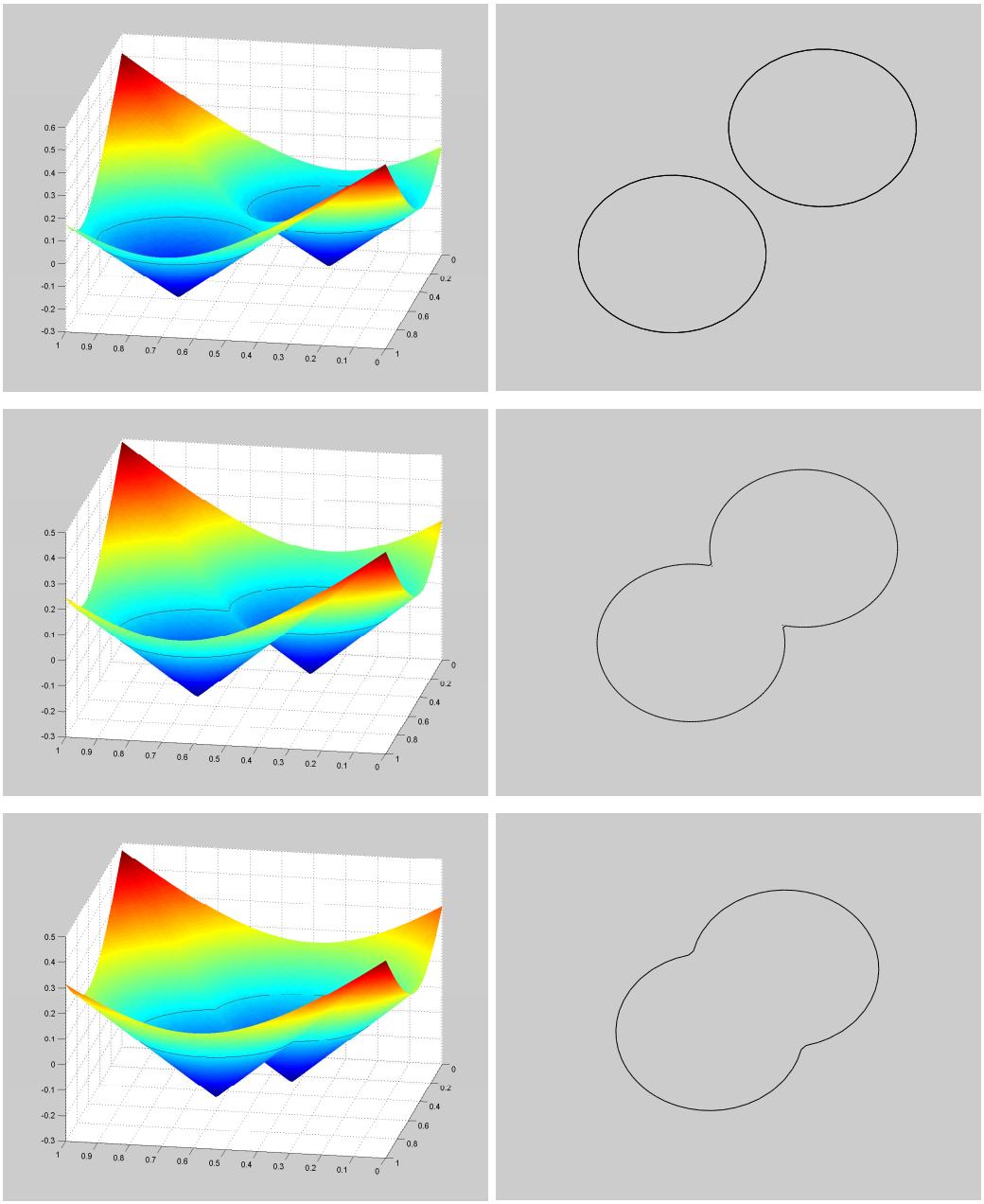


Figure 1.5: An example of a level set function evolving over time (left column) with corresponding plots of the zero level set which represents the interface (right column).

1.3 Contributions of This Thesis

In this thesis project, we develop a new level set method. This is an interface-fitted finite element based level set method. The key features of this method are:

The mesh: We use a uniform base mesh with vertices at lattice points. At each time step the mesh is refined to produce an interface-fitted mesh which explicitly locates the moving interface. Although it takes extra time to refine the mesh at each time step, certain calculations are made easier.

New reinitialization method: We reinitialize the level set function by solving a Poisson's equation type problem. We keep the level set function smooth across the interface by imposing a gradient jump condition, then solve an overdetermined system by the least squares method. This method produces a new level set function that is smooth over the domain and across the interface. Being smooth is important because it increases the accuracy of the curvature calculation. Some examples are given to demonstrate how this reinitialization works.

New methods for velocity extension: The level set method relies on having a velocity function defined near the interface, not just at the interface, so if the velocity of an interface is known only on the interface, we need to extend this velocity away from the interface in a smooth way. We solve Laplace's equation on the interior and exterior of the interface using the velocity on the interface as a boundary condition. This is a simple way to extend the velocity. If smoothness is required we can impose a gradient jump condition across the interface and solve by least squares. (This is similar to our reinitialization method.) Some examples are given to show how this method works.

Numerical analysis of curvature approximations: We also present a new way to calculate the curvature of the level set function. This is comparable to using standard finite difference approximations to first and second derivatives, but this method may be less expensive in certain applications since some of

the quantities used in the calculation may be reused during other steps of the algorithm. A proof and examples of the accuracy of this calculation are given.

Applications to solidification problems: Here we demonstrate the algorithm by using it to solve Stefan problems. The examples model dendritic solidification of a frozen seed placed into a supercooled liquid. The effects of including isotropic, anisotropic and no surface tension are modeled. Other examples show the effects of varying the phase angle of the anisotropic surface tension. These examples demonstrate that using our method produces similar results to those previously produced by a strictly finite difference scheme without interface-fitting [CMOS97].

Applications to molecular solvation: Here we demonstrate the algorithm by using it to find the optimal solute-solvent interface of a solvation system. We use a variational implicit solvent model for equilibrium solvation systems of nonpolar molecules. Each molecule consists of a set of atoms specified in the computational domain. An initial interface enclosing the atoms is moved in the direction that reduces the free energy. The free energy includes the interface energy of the solute and the solute-solvent van der Waals type interactions. Our examples demonstrate how our method converges to the optimal solute-solvent interface of various solvation systems.

Chapter 2

Level Set Method

In this chapter we detail the level set method and explain how it is used in our algorithm. The level set method was first developed in 1987 by Sethian and Osher [OS88] to overcome difficulties of other interface tracking methods.

The level set method tracks a moving interface implicitly. A continuous function is defined on a domain in such a way that the zero level set of this function represents the interface. This function evolves in time according to a partial differential equation which is appropriated called the level set equation. The basic derivation of the level set method is discussed in section 2.1.

Finite difference based algorithms are well developed for the level set method. This is in large part because the two primary originators of the level set method, Stanley Osher and James A. Sethian, developed finite difference schemes and applied them to various problems. Also, the evolution of the level set function is similar to fluid dynamics problems where computations favor large numbers of grid points with lower order approximations. In section 2.2 we explain some of the basic finite difference methods used to numerically solve the level set equation.

Finite element methods for solving the level set equation are less established than finite difference methods, but recently there has been more research in this area. There are several potential benefits of using finite element methods instead of finite difference methods. These include handling complex shaped domains more easily and adaptive refinement of the mesh to increase accuracy. Another key advantage is in certain problems a finite element mesh can also be used to solve

auxiliary equations (e.g. field equations) that may be more difficult to solve using a finite difference method. In section 2.3 we discuss some methods of solving the level set equation using finite elements rather than finite differences.

2.1 Derivation of the Level Set Equation

Let us consider a moving interface $\Gamma = \Gamma(\mathbf{x}, t)$ where $\mathbf{x}$ denotes a generic point of the interface and t denotes time. This moving interface is represented as the zero level set of a function $\phi(\mathbf{x}, t)$. That is, $\Gamma(\mathbf{x}, t) = \{\mathbf{x} \mid \phi(\mathbf{x}, t) = 0\}$. If a velocity vector field $\mathbf{V}(\mathbf{x}, t)$ is given on a domain, then the equation

$$\phi_t + \mathbf{V} \cdot \nabla\phi = 0 \tag{2.1.1}$$

will propagate the level set of ϕ that passes through the point $\mathbf{x}$ with velocity $\mathbf{V}(\mathbf{x}, t)$.

We now derive equation (2.1.1) in two dimensions. We use (x, y) to denote a generic point of the moving curve, the two-dimensional "surface." Since the level set $\phi(x, y, t) = 0$ represents the moving interface, we can imagine an arbitrary point on the interface moving in time and remaining on this level set. Say the location of this point is $\big(x(t), y(t)\big)$. If this point remains on the interface as time changes, then

$$\frac{d}{dt}\,\phi\big(x(t), y(t), t\big) = 0. \tag{2.1.2}$$

By the chain rule,

$$\phi_t + \phi_x x_t + \phi_y y_t = 0 \tag{2.1.3}$$

$$\phi_t + (x_t, y_t) \cdot (\phi_x, \phi_y) = 0 \tag{2.1.4}$$

$$\phi_t + \mathbf{V} \cdot \nabla\phi = 0 \tag{2.1.5}$$

where $\mathbf{V} = (x_t, y_t)$. cf. equation (1.1.1). Equation (2.1.5) is the level set equation for motion by an externally generated velocity field. Note that this equation also determines how each level set of ϕ propagates in time. That is, the level set of ϕ that passes through the point (x, y) at time t will move with velocity $\mathbf{V}(x, y, t)$ at time t. Also note that the component of $\mathbf{V}(x, y, t)$ that is tangential to the level

set of ϕ at the point (x, y) does not affect the motion of the level set; only the normal component of $\mathbf{V}$ moves the level set.

Since a unit normal to the level set $\phi(x, y, t) = c$ is

$$\mathbf{n}(x, y, t) = \frac{\nabla\phi(x, y, t)}{|\nabla\phi(x, y, t)|}, \tag{2.1.6}$$

we can write the normal velocity field as

$$V_{\mathbf{n}} = \mathbf{V}(x, y, t) \cdot \frac{\nabla\phi(x, y, t)}{|\nabla\phi(x, y, t)|}. \tag{2.1.7}$$

Rearranging, we get

$$V_{\mathbf{n}}|\nabla\phi(x, y, t)| = \mathbf{V}(x, y, t) \cdot \nabla\phi(x, y, t). \tag{2.1.8}$$

And substituting this into equation (2.1.5) gives us

$$\phi_t + V_{\mathbf{n}}|\nabla\phi| = 0. \tag{2.1.9}$$

This is the level set equation for internally generated velocity fields. The normal velocity $V_{\mathbf{n}}$ is a scalar function and its direction at a point in the domain depends on the orientation of the level set of ϕ at that point. This form of the equation is good for representing an outward or inward interface velocity. Alternately, the velocity in equation (2.1.1) transports the interface through the domain without regard to the interface's orientation.

One should note that to solve either equation (2.1.1) or (2.1.9) the velocity must be given on the entire domain. In many applications the velocity is only determined on the interface itself, so one must extend this velocity from the interface over the domain to make a velocity field. Ideally, this extension is smooth so that when solving either level set equation, the level set function remains smooth and the interface moves at the correct velocity. See Section 3.6 for our implementation of velocity extension.

Another interesting form of the level set equation is when the motion of the interface is proportional to the mean curvature of the interface in the normal direction. This can be used to model a surface tension force at the interface. Motion by mean curvature in the outward normal (convexity) direction is inherently unstable; small perturbations will grow larger in the outward direction. But motion

by mean curvature in the inward normal (concave) direction is stable and tends to smooth the interface. For this reason, it is convenient to let

$$V_{\mathbf{n}} = -\epsilon\kappa \tag{2.1.10}$$

where $\epsilon > 0$ and κ is the mean curvature. The mean curvature is easily computed from the level set function by the formula

$$\kappa = \nabla \cdot \mathbf{n} = \nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|} \right). \tag{2.1.11}$$

So the level set equation for motion by mean curvature is

$$\phi_t = \epsilon\kappa|\nabla\phi|. \tag{2.1.12}$$

2.2 Finite Difference Based Level Set Methods

Depending on the form of the level set equation, various discretizations can be used. The level set equation with vector velocity field,

$$\phi_t + \mathbf{V} \cdot \nabla\phi = 0, \tag{2.2.13}$$

requires upwind differencing to capture information traveling in the direction of the velocity. A first order space discretization with forward Euler time discretization in two dimensions would be

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + u^n\phi_x^n + v^n\phi_y^n = 0, \tag{2.2.14}$$

where ϕ^n, u^n and v^n denote the approximation of ϕ, u and v, respectively at time t_n, and $\mathbf{V} = (u, v)$. At each grid point, the first directive in each spatial direction is chosen as a forward difference or backward difference depending on the velocity at the grid point. The forward difference at grid point x_i would be

$$D^+\phi_i^n = (\phi_x)_i^n = \frac{\phi_{i+1}^n - \phi_i^n}{\Delta t}, \tag{2.2.15}$$

and a backward difference would be

$$D^-\phi_i^n = (\phi_x)_i^n = \frac{\phi_i^n - \phi_{i-1}^n}{\Delta t}. \tag{2.2.16}$$

So if the velocity component in the x-direction at grid point x_i is positive, then information is carried in the positive x-direction and the backward difference, $D^-\phi_i^n$, is chosen.

For this scheme to be stable, the time step must satisfy the Courant-Friedreichs-Lewy condition (CFL condition), which means that $\Delta x/\Delta t$ must be greater than $|\mathbf{V}|$. That is,

$$\Delta t < \frac{\Delta x}{\max|\mathbf{V}|} \tag{2.2.17}$$

where the maximum is taken over the entire domain.

If one wants to increasing the spacial accuracy by using central differencing ($\phi_x = (\phi_{i+1} - \phi_{i-1})/2\Delta x$) while still using forward Euler discretization, then stability is harder to achieve. One way to have stability is to use the costly time step restriction of Δt proportional to $(\Delta x)^2$. Another way is by adding some artificial dissipation to equation (2.1.1) to get

$$\phi_t + \mathbf{V} \cdot \nabla\phi = \epsilon\Delta\phi. \tag{2.2.18}$$

For these reasons, higher order upwind differencing methods known as essentially non-oscillatory (ENO) are usually preferred over central differencing for spacial discretizations. The idea of ENO methods was first introduced by Harten et al. in [HEOC87], and these ideas were improved by Shu and Osher in [SO88] and [SO89]. The idea with ENO methods is to find higher order approximations for ϕ_x^+ and ϕ_x^-. The approximations are chosen based on the smoothest possible polynomial interpolants of ϕ at each grid point.

For example, a third order accurate scheme would calculate $(\phi_x^-)_i$ by first constructing three polynomial interpolants. The first one is constructed from the values $\{\phi_{i-3}, \phi_{i-2}, \phi_{i-1}, \phi_i\}$, the second from $\{\phi_{i-2}, \phi_{i-1}, \phi_i, \phi_{i+1}\}$ and the third from $\{\phi_{i-1}, \phi_i, \phi_{i+1}, \phi_{i+2}\}$. The smoothest of these polynomials is used to approximate $(\phi_x^-)_i$. Note that each of the potential polynomials includes the values ϕ_{i-1} and ϕ_i to capture the information coming from the left.

If an application uses a normal velocity as in the level set equation of the form

$$\phi_t + V_{\mathbf{n}}|\nabla\phi| = 0, \tag{2.2.19}$$

then a slightly more sophisticated scheme must be used. We can rewrite the equation in a form similar to equation (2.1.1) as

$$\phi_t + \left(\frac{V_1 \phi_x}{|\nabla \phi|}, \frac{V_2 \phi_y}{|\nabla \phi|} \right) \cdot \nabla \phi = 0. \tag{2.2.20}$$

Here, the term $\left(\frac{V_1 \phi_x}{|\nabla \phi|}, \frac{V_2 \phi_y}{|\nabla \phi|} \right)$ is like our $\mathbf{V}$ in equation (2.1.1) except that it depends on ϕ_x and ϕ_y. This dependance makes choosing the upwind direction more complicated in certain situations. At an area where ϕ_x changes signs, ϕ_x^+ and ϕ_x^- will have different signs, so $V_1 \phi_x$ will have different signs if we use ϕ_x^+ or ϕ_x^- to approximate ϕ_x. These situations are where the level set function has a "V" shape and special care must be taken to get the correct solution. A common approach is to add a small amount of artificial viscosity at these points to obtain the correct vanishing solution. Schemes that handle these situations have been designed for general Hamilton-Jacobi equations of the form

$$\phi_t + H(\nabla \phi) = 0 \tag{2.2.21}$$

Level set equations (2.1.1) and (2.1.9) are both of this type. The former is obtained with $H(\nabla \phi) = \mathbf{V} \cdot \nabla \phi$ and the latter with $H(\nabla \phi) = V_{\mathbf{n}} |\nabla \phi|$.

Curvature driven flows like in equation (2.1.12) cannot use an upwinding scheme since the $\epsilon \kappa |\nabla \phi|$ term is parabolic (similar to $\epsilon \Delta \phi$). A diffusive term like this sends information in all directions, so it requires a central differencing scheme to capture the information from all directions. If central differencing for the spacial discretization is coupled with the explicit forward euler time discretization then a restrictive time step of $\Delta t \sim (\Delta x)^2 / \epsilon$ is required to achieve stability.

Using the implicit Crank-Nicolson scheme (backward Euler time, trapezoidal rule with central differencing spacial) will allow a time step of $\Delta t \sim \Delta x$ and have second order accuracy in time and space. The cost of doing this is solving a system of equations on each time step. Also, the $\kappa^{n+1} |\nabla \phi^{n+1}|$ term is not linear, so the system of equations needs to be linearized and solved iteratively (i.e. using Newton's method).

2.3 Finite Element Based Level Set Methods

For the internal level set equation

$$\phi_t + V_{\mathbf{n}}|\nabla\phi| = 0, \tag{2.3.22}$$

we can devise a simple scheme on a uniform grid of triangles. We assume $V_{\mathbf{n}}$ is defined at all vertices. We approximate the $|\nabla\phi|$ term at a vertex v_i by taking an average over a patch of triangles around v_i (see Figure 2.1). There are two ways the average can be taken: either taking the magnitude of ϕ before the sum,

$$|\nabla\phi|_i = \frac{\sum_{\omega_i}\int_{\omega_i}|\nabla\phi|\,dx\,dy}{\sum_{\omega_i}\text{area}(\omega_i)}, \tag{2.3.23}$$

or by summing ϕ over each triangle, then taking the resultant's magnitude,

$$|\nabla\phi|_i = \frac{\left|\sum_{\omega_i}\int_{\omega_i}\nabla\phi\right|\,dx\,dy}{\sum_{\omega_i}\text{area}(\omega_i)}. \tag{2.3.24}$$

If ϕ is at a local minimum at vertex v_i, then $\nabla\phi$ over opposite triangles will tend to cancel and our approximation could end up being zero. Cancelation could also happen if v_i was at the bottom of a trough. This is not desirable since ϕ_i would then not change in time. For this reason, the first definition (2.3.23) is better.

This coupled with a simple explicit time scheme such as forward Euler gives a scheme

$$\phi_i^{n+1} = \phi_i^n - \Delta t(V_{\mathbf{n}})_i|\nabla\phi^n|_i. \tag{2.3.25}$$

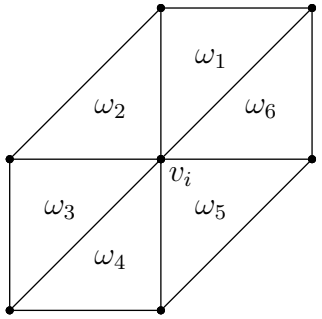


Figure 2.1: A patch of triangles around a vertex v_i.

Chapter 3

An Interface-fitted Finite Element Level Set Method

In this chapter we describe details of our interface-fitted finite element based level set method. The method has two main parts. The first part involves tracking the interface implicitly using the level set method on a uniform *base mesh*. At each time step the original base mesh is refined to explicitly locate the interface. The second part of the method is all the steps that take place on the *interface-fitted mesh*.

We recall that the level set method uses a level set function ϕ defined on a domain Ω to implicitly locate a moving interface. The interface Γ is represented by the zero level set of ϕ,

$$\Gamma = \{\mathbf{x} \in \Omega \mid \phi(\mathbf{x}, t) = 0\}. \tag{3.0.1}$$

The level set function ϕ evolves in time according to the level set equation

$$\phi_t + V_{\mathbf{n}}|\nabla\phi| = 0, \tag{3.0.2}$$

where $V_{\mathbf{n}}$ is the normal velocity.

3.1 Description of Algorithm

Most applications follow the steps of the algorithm given below.

Step 1: Initialize. Define the base mesh. Define the level set function so that the zero level set corresponds to the initial position of the interface. Define any other problem-dependent parameters.

Step 2: Interface-fit the mesh. Explicitly locate the interface and refine the base mesh to create an interface-fitted mesh. This explicitly separates the domain Ω into Ω_+ and Ω_-.

Step 3: Solve field equations. The velocity of the moving interface Γ may depend on the solution to field equations, or it may depend on knowing the precise location of the interface. Since Γ forms the boundary of Ω_- and part of the boundary of Ω_+, boundary conditions can be imposed on Γ.

Step 4: Calculate interface velocity. The velocity of the interface can depend on the solution to field equations (solved on the previous step) or may be highly depend on the precise location of the interface. In either case, on this step the velocity is determined along the edges defining the interface.

Step 5: Extend velocity. The velocity that was determined on the interface needs to be extended to the entire domain in order to solve the level set equation and thus propagate the interface. How this is done in explained in section 3.6.

Step 6: Unrefine the mesh. Once the velocity is smoothly extended away from the interface, the refined mesh is no longer needed. The calculation returns to the original base mesh.

Step 7: Move the interface. The interface is propagated using the velocity defined on the domain by solving the level set equation on the base mesh. This is accomplished using one of the methods described in Chapter 2.

Step 8: Repeat the process. Go to step 2.

3.2 Base Mesh

In all of our problems, we start with a rectangular domain $\Omega = [a, b] \times [c, d]$. We define a lattice of points on Ω with spacing sizes $h = (b - a)/M_1$ and $k = (d - c)/M_2$ by letting

$$x_i = a + ih, \quad i = 0, 1, \ldots, M_1 \tag{3.2.3}$$

$$y_j = c + jk, \quad j = 0, 1, \ldots, M_2 \tag{3.2.4}$$

and then defining the points

$$v_{i,j} = (x_i, y_j) \tag{3.2.5}$$

These lattice points become the vertices of a regular and uniform mesh of triangles of the type shown in Figure 3.1. We denote this triangulation as $\mathcal{T}_h$, the set of triangles in the mesh. The level set function ϕ as well as any problem dependent functions defined on Ω are discretized on this mesh. Time is discretized as $t_n = n\Delta t$. Let

$$\phi^n_{i,j} = \phi(v_{i,j}; t_n) = \phi(x_i, y_j; t_n) \tag{3.2.6}$$

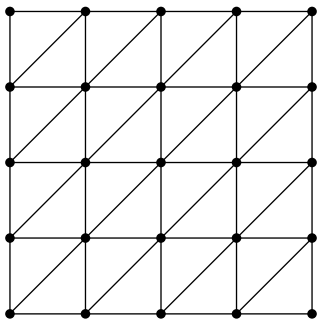


Figure 3.1: The type of base mesh pattern used in this thesis, denoted as $\mathcal{T}_h$.

3.3 Level Set Function Initialization

In any problem the initial location of the interface Γ is a given. A level set function ϕ is then defined on the base mesh such that the zero level set of ϕ represents Γ. The difficulty in constructing ϕ depends on the initial shape of the

interface. For example, if Γ is a circle of radius r with center (x_0, y_0), then ϕ can be initialized by a simple formula such as

$$\phi(x, y) = \sqrt{(x - x_0)^2 + (y - y_0)^2} - r \tag{3.3.7}$$

This would give a signed distance function. A signed distance function at a point has value equal (in magnitude) to the distance from the closest point on the interface, and on one side of the interface it is positive, and on the other side it is negative. More complicated initial shapes of Γ would require more complicated formulas.

If Γ is specified as an arbitrary shape by a discrete set of points, then to make ϕ a signed distance function we would need define ϕ at each mesh point by explicitly calculating the distance to the nearest point on Γ. A less expensive method would be to employ a fast marching method.

The level set function ϕ is represented discretely as

$$\phi_h = \sum_j \phi_j \psi_j \tag{3.3.8}$$

where $\{\psi_j\}$ are linear basis functions for the space

$$S_h = \{v \in \mathcal{C}(\Omega) \colon v \text{ linear in } K \text{ for each } K \in \mathcal{T}_h\}. \tag{3.3.9}$$

That is

$$\psi_j(v_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{3.3.10}$$

3.4 Interface-fitted Mesh Refinement

Each edge has a vertex at each end. Say these vertices are at points v_1 and v_2 as in Figure 3.2. For each edge, the value of the level set function at each vertex, $\phi_1 = \phi(v_1)$ and $\phi_2 = \phi(v_2)$, is examined. When the value is positive at one end and negative at the other end we know that the interface crosses that particular edge. We locate the point v^* along that edge where the level set function is zero

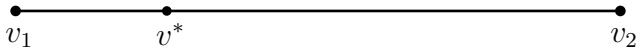


Figure 3.2: A generic edge.

and add a vertex there, splitting the edge into two edges. This point is given by the equation

$$v^* = \frac{\phi_2 v_1 - \phi_1 v_2}{\phi_2 - \phi_1}. \tag{3.4.11}$$

We then call this edge *refined*.

Once all of these new vertices are added, we examine each triangle. For each triangle, there can be zero, one or two edges that were refined.

If only one edge was refined, then the value of the level set function must be zero on the opposite vertex. In this case a new edge is added joining the new vertex with the opposite vertex, splitting the triangle into two triangles. (See upper-left triangle in Figure 3.3)

The more common situation is when two edges of a triangle were refined. In this case, this splits the triangle into a triangle and a quadrilateral. We then split this quadrilateral into two triangles by adding an edge along one of the diagonals. (See lower-right triangle in Figure 3.3) This splits the original triangle into three triangles.

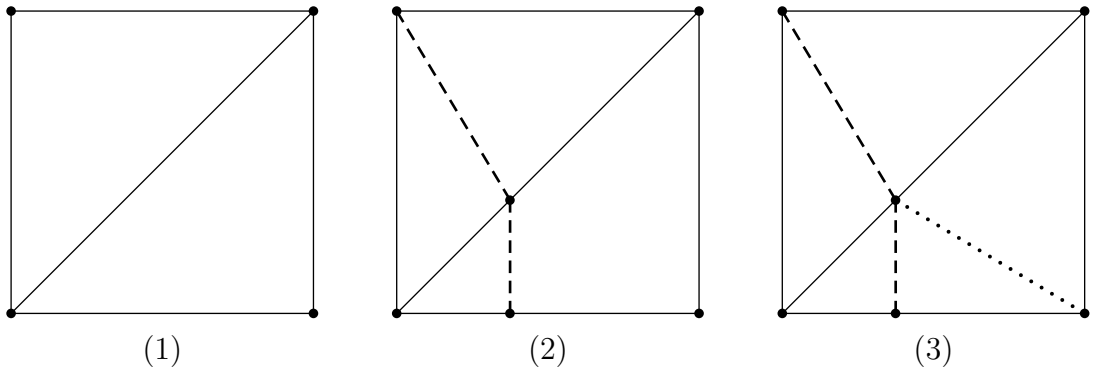


Figure 3.3: Example of two types of interface-fitted mesh refinement. (1) Original triangles in base mesh. (2) Edge (dashed) added along interface. (3) Edges (dotted) added to split lower triangle into three triangles.

We denote a base triangulation $\mathcal{T}_h$ that is refined to an interface-fitted triangulation at time t_n as $\mathcal{T}_{h,\Gamma}^n$. The mesh refinement process is illustrated in Figures 3.4, 3.5 and 3.6.

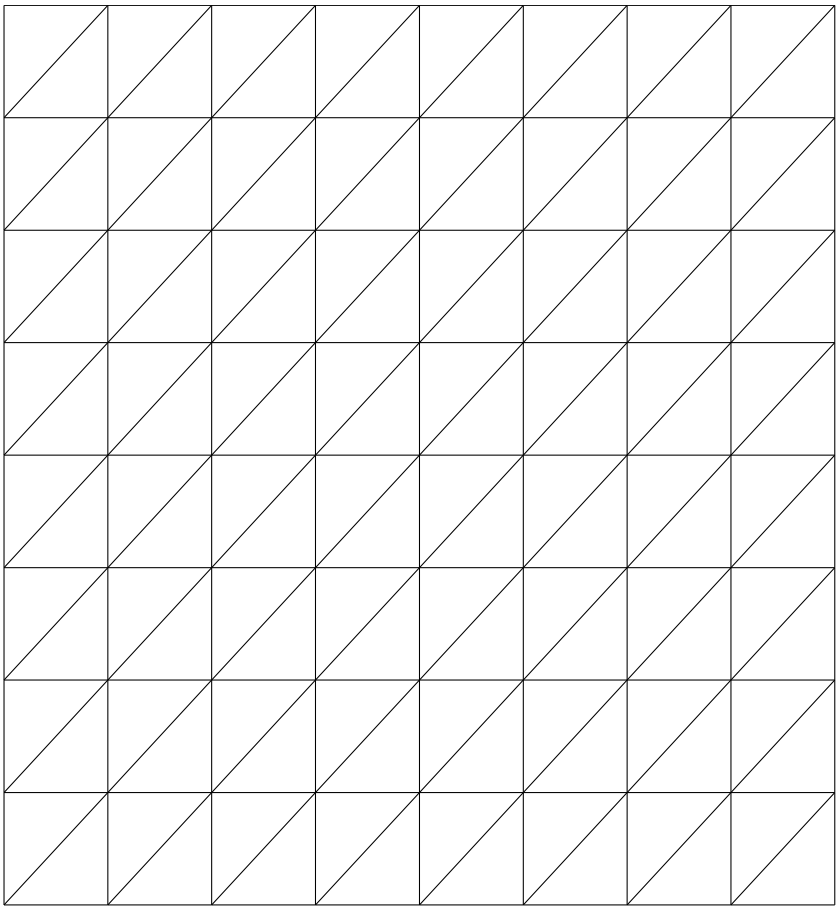


Figure 3.4: Example of base mesh $\mathcal{T}_h$.

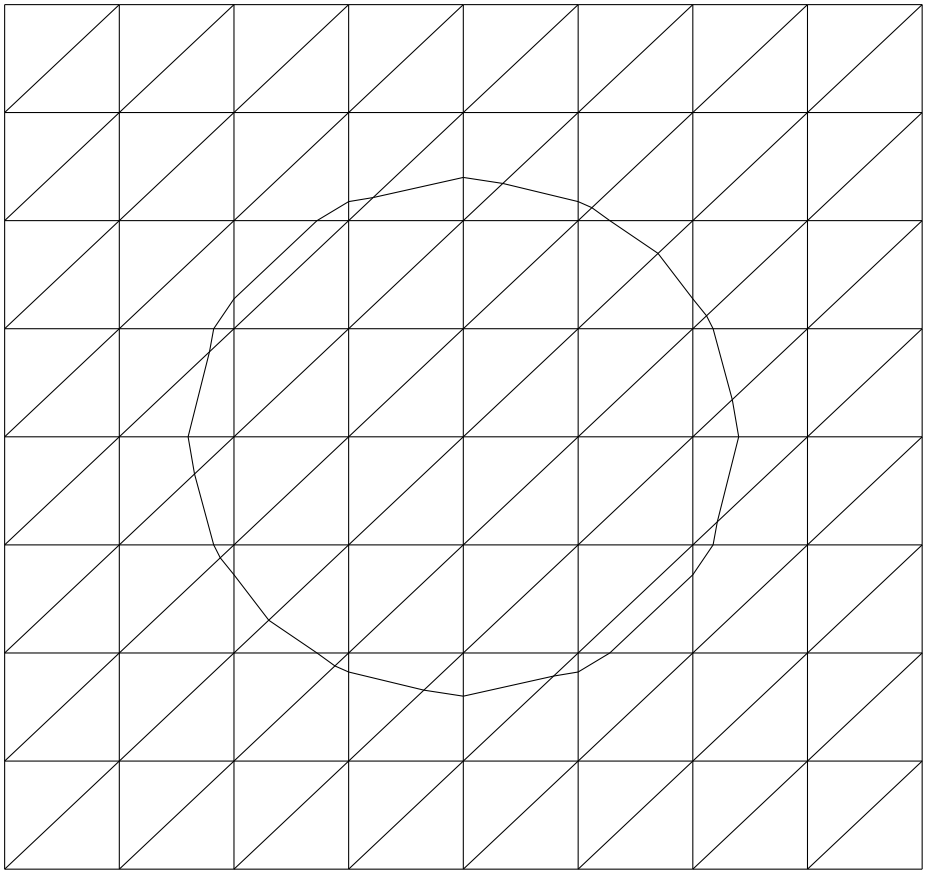


Figure 3.5: Example of base mesh $\mathcal{T}_h$ with interface Γ shown.

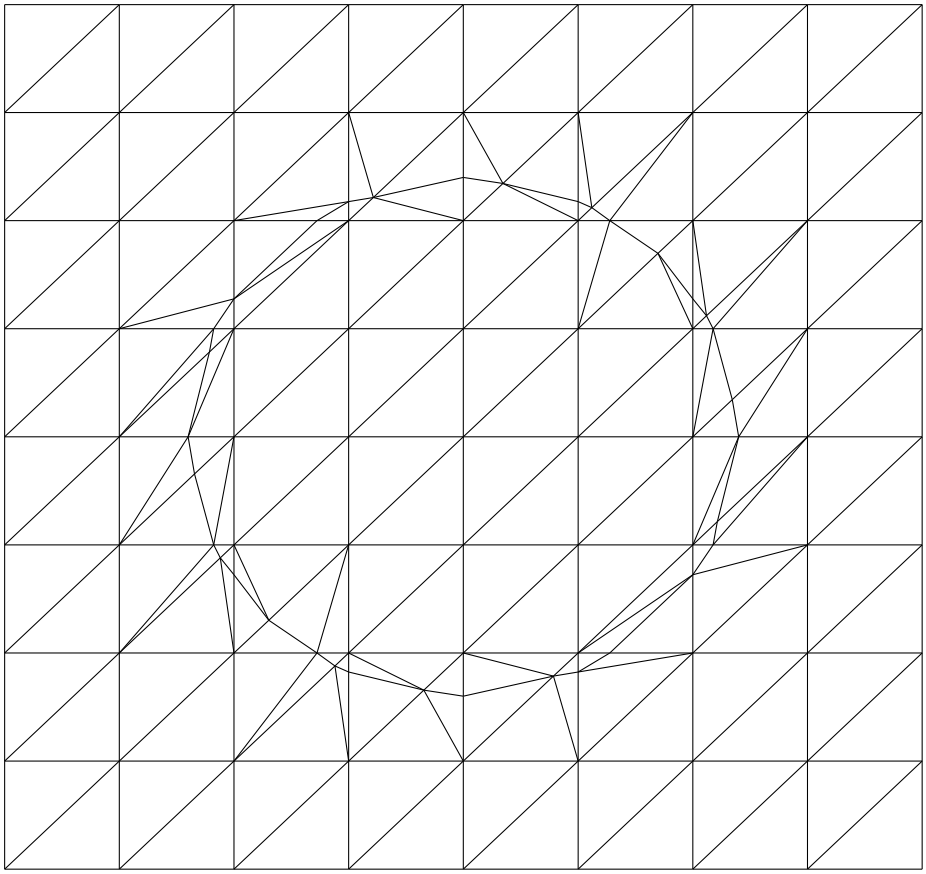


Figure 3.6: Example of mesh after interface-fitted refinement, denoted $\mathcal{T}_{h,\Gamma}^n$.

At time t_n we define the linear finite element basis functions $\{\psi_j^n\}_j$ for the space

$$S_h^n = \{v \in \mathcal{C}(\Omega) \colon v \text{ linear in } K \text{ for each } K \in \mathcal{T}_{h,\Gamma}^n\} \tag{3.4.12}$$

where

$$\psi_j^n(v_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{3.4.13}$$

Functions are represented on $\mathcal{T}_{h,\Gamma}^n$ as a linear combination of these basis functions.

We store the original configuration ($\mathcal{T}_h$), so the unrefinement step is just reverting back to the original parameters that define our base mesh.

It is quite likely that some of the new triangles in the refined mesh will be skinny which can decrease the accuracy of any calculation on the refined mesh. To avoid this an alternative approach would be to move some of the original vertices slightly so that they lie precisely on the (calculated) interface. But this approach would complicate the algorithm because the base mesh would be changed. If we want to revert back to the original base mesh we would need to do additional work to calculate quantities at the vertices that were moved and moved back.

3.5 Reinitialization

As the interface propagates, the level set function changes in time. Although the velocity at the interface determines how the interface moves, how the velocity is defined off of the interface determines how the other (nonzero) level sets move. The level sets can bunch up making the level set function steeper, or they can spread apart making the level set function flatter. For example, motion by mean curvature has a flattening effect on the level set function. Moreover, the level set function can become less smooth. These things can eventually lead to catastrophic numerical errors.

For these reasons the level set function needs to be *reinitialized* occasionally. An important consideration when reinitializing is to try to keep the zero level set in the same location. Ideally, the reinitialized function would be smooth and its gradient would have magnitude approximately equal to one.

It has been suggested the the best level set function would be a signed distance function. That is, a function whose absolute value at a point is equal to the distance from the nearest point of Γ. The function would be negative on Ω_- and positive on Ω_+.

One way to reinitialize is to solve Poisson's equation on Ω with appropriate boundary conditions as follows:

$$\Delta\phi = G \qquad \text{in } \Omega_- \tag{3.5.14}$$

$$\Delta\phi = 0 \qquad \text{in } \Omega_+ \tag{3.5.15}$$

$$\phi = 1 \qquad \text{on } \partial\Omega \tag{3.5.16}$$

$$\phi = 0 \qquad \text{on } \Gamma \tag{3.5.17}$$

where $G < 0$ is a constant. This system can be solved using standard finite element techniques.

$$\int_\Omega \Delta\phi v \, dx = \int_\Omega f v \, dx \quad \text{for all } v \in S_h^n \tag{3.5.18}$$

$$-\int_\Omega \nabla\phi \cdot \nabla v \, dx = \int_\Omega f v \, dx \quad \text{for all } v \in S_h^n \tag{3.5.19}$$

where

$$f = \begin{cases} 0, & \text{on } \Omega_+ \\ G, & \text{on } \Omega_- \end{cases} \tag{3.5.20}$$

Since we reinitialize on an interface-fitted mesh $\mathcal{T}_{h,\Gamma}^n$, the interface does not move. That is, we can set the values of ϕ on Γ to zero. But we need to consider the location of the interface (zero level set) after we unrefine the mesh. That is, if the new level set function is not smooth over Γ, then unrefinement would implicitly move the zero level set, and consequently, the interface. To solve this problem we can require that the jump in the gradient of ϕ across the interface is zero; $[\![\frac{\partial\phi}{\partial n}]\!]_\Gamma = 0$. This gives

$$\Delta\phi = 0 \qquad \text{in } \Omega_- \tag{3.5.21}$$

$$\Delta\phi = 0 \qquad \text{in } \Omega_+ \tag{3.5.22}$$

$$\phi = 1 \qquad \text{on } \partial\Omega \tag{3.5.23}$$

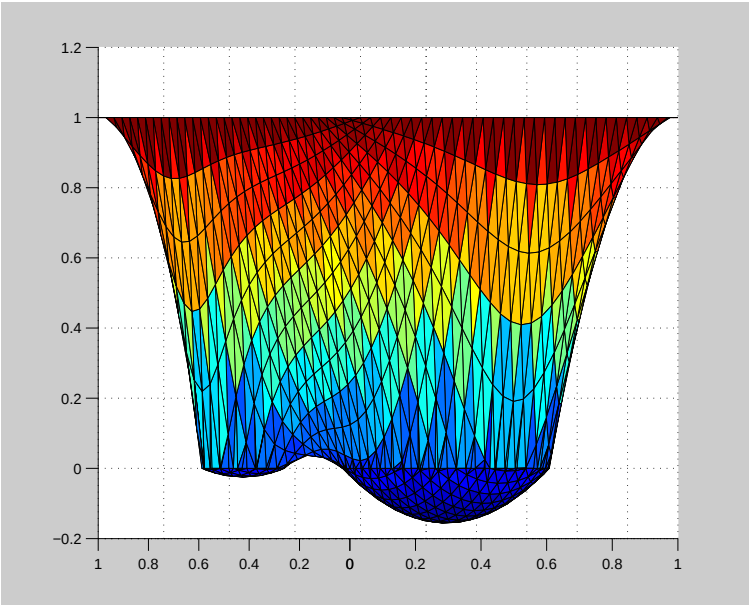
$$\phi = 0 \qquad \text{on } \Gamma \tag{3.5.24}$$

$$\left[\!\left[\frac{\partial\phi}{\partial n}\right]\!\right]_\Gamma = 0 \qquad \text{on } \Gamma \tag{3.5.25}$$

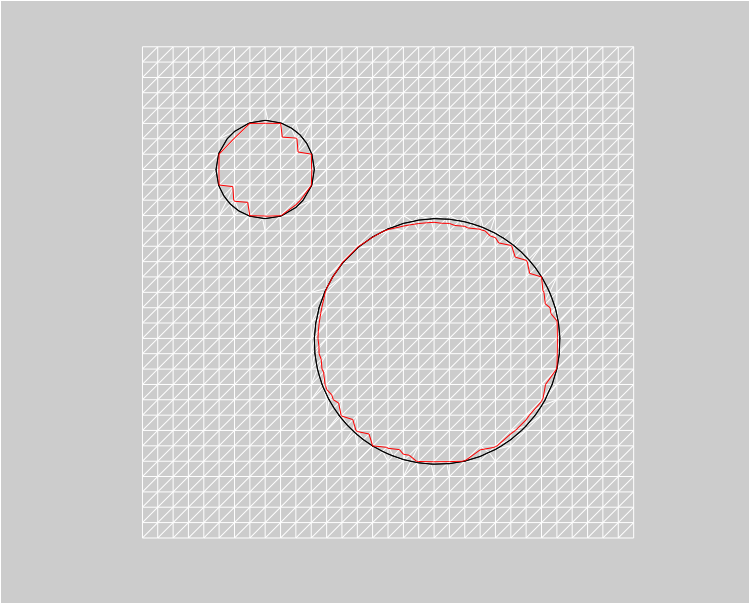
By imposing the gradient jump condition, the system is overdetermined. We choose to maintain the conditions (3.5.23) and (3.5.24) and solve a least square system for the other conditions. This produces a smooth level set function ϕ.

Figure 3.7 shows reinitialization without the gradient jump condition. Figure 3.7 (a) shows the level set function after reinitialization. It is shown from the side view to highlight the jump in the gradient. Figure 3.7 (b) shows the interface location before reinitialization and the location after reinitialization and unrefinement. After unrefining the mesh, the location of Γ moves since Γ is then interpolated from the values of ϕ on the base mesh.

Figure 3.8 shows reinitialization with the gradient jump condition. Figure 3.8 (a) shows the level set function, from the side view, after reinitialization. There is no jump in the gradient as there was in Figure 3.7 (a). Figure 3.8 (b) shows that the interface stays in the same location after mesh unrefinement.

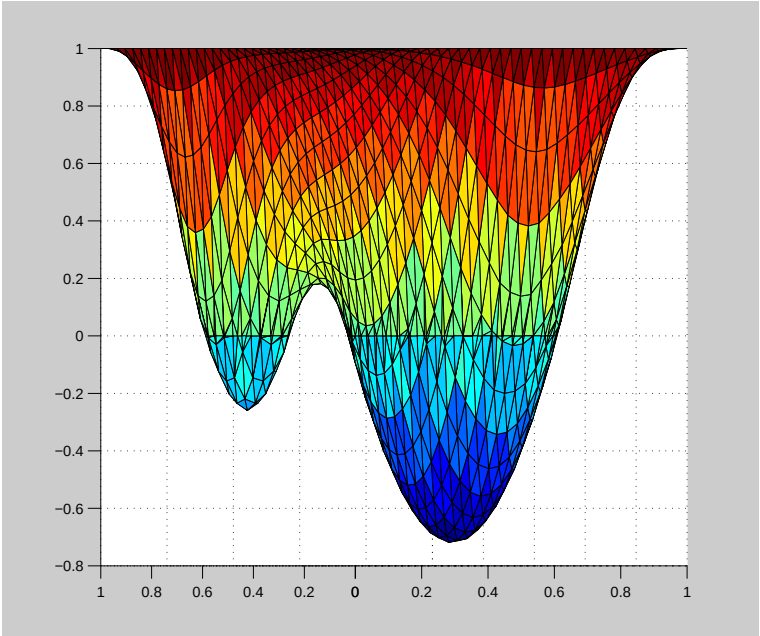


(a)

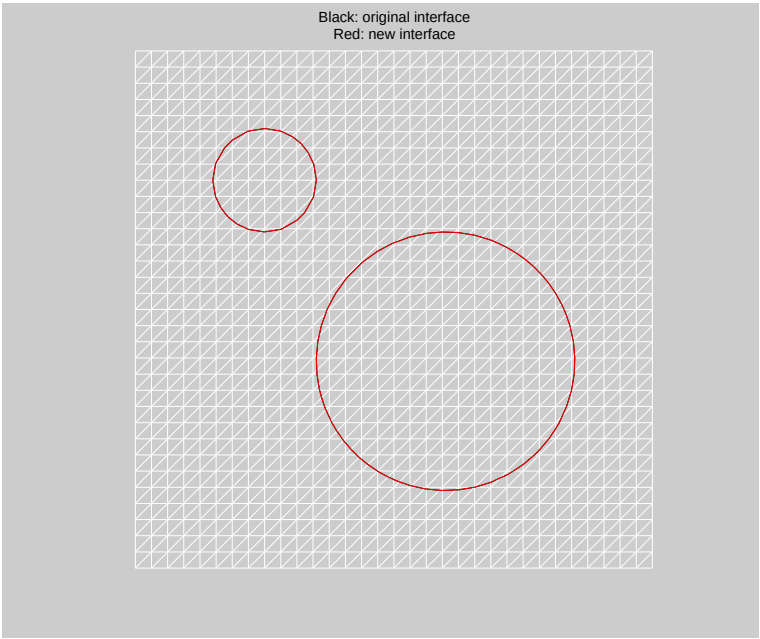


(b)

Figure 3.7: Example of reinitialization without gradient jump condition. (a) Level set function (side view) shown after reinitialization. (b) Interface movement during reinitialization. Interface is a circle before but becomes ragged after unrefinement.



(a)



(b)

Figure 3.8: Example of reinitialization with gradient jump condition. (a) Level set function shown after reinitialization. (b) Interface movement during reinitialization with gradient jump condition. Interface remains a circle after unrefinement.

3.6 Velocity Extension

The velocity must be defined on the entire domain to be able to solve the level set equation, so if the velocity of our interface is only defined on the interface, we need to extend this velocity to the rest of the domain on each time step. Even though only the zero level set of our level set function determines our interface, the velocity field will determine how all level sets of our level set function propagate.

How we extend the velocity away from the interface is important for several reasons. We want the interface to propagate with the correct velocity. Velocity extensions that are not smooth enough can cause the level set function to deteriorate quickly over time.

A typical approach to velocity extension is to set the velocity at mesh points off of the interface to the velocity at the closest interface point. A somewhat sophisticated method is required for this approach to be efficient. This typically involves defining the velocity extension at mesh points closest to the interface and progressing to other points away from the interface.

Our method of velocity extension is to think of the velocity at the interface as diffusing into the rest of the domain. That is, at each time step, after the velocity on the interface is determined, we solve Laplace's equation (i.e. steady-state heat equation) on the domain. This is easy to implement since the mesh is interface-fitted.

$$\Delta V = 0 \qquad \text{in } \Omega_- \tag{3.6.26}$$

$$\Delta V = 0 \qquad \text{in } \Omega_+ \tag{3.6.27}$$

$$V_{\mathbf{n}} = V_\Gamma \qquad \text{on } \Gamma \tag{3.6.28}$$

$$\frac{\partial V_{\mathbf{n}}}{\partial n} = 0 \qquad \text{on } \partial\Omega \tag{3.6.29}$$

The velocity on the interface is V_Γ. Note that we are essentially solving Laplace's equation separately on Ω_- and Ω_+. On Ω_- we use the interface velocity as the boundary condition along Γ. On Ω_+ we use the interface velocity as the boundary condition along Γ and have a free boundary condition along $\partial\Omega$.

The drawback to this approach is that it will produce a velocity extension $V_{\mathbf{n}}$ that will not have a continuous derivative along Γ. Near the points on the

interface with the highest velocity the extension will have lower velocity values on either side. This is a problem since after unrefinement (of the interface-fitted mesh) the explicit values of the velocity on the interface are lost, and the interpolated velocity values on the interface will be lower than they should be. This is similar to the problem encountered during reinitialization of ϕ discussed in Section 3.5 where the values on the interface are lost after unrefinement.

When we unrefine the interface-fitted mesh, we lose the values of the velocity at the nodes on the interface, so the extended velocity should capture the correct velocity of the interface implicitly. That is, the representation of the velocity on the unrefined mesh should be such that at the location of the interface it should still equal what we want the velocity to be on the interface. This can be accomplished by requiring the extension to be differentiable. That is, we impose the condition that the jump of the gradient of the velocity across Γ should be zero ($[\![\frac{\partial V}{\partial n}]\!]_\Gamma = 0$). See Figure 3.9 for an illustration of this point in one dimension.

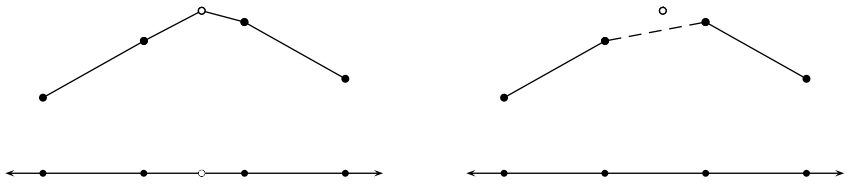


Figure 3.9: Illustration of how nonsmooth velocity extension can create errors. (a) Velocity value on the interface vertex (circle) is extended away to the base mesh vertices (solid dots) to create a continuous function. (b) When the interface vertex is removed and only base mesh vertices remain, the interpolated (dashed line) interface value is lower than it should be.

If we want to use this method but constrain $V_{\mathbf{n}}$ so that its derivative is continuous across Γ, then the problem is overdetermined. We can impose our constrain at the expense of only solving Laplace's equation in a least squares way.

The system of equations is then

$$\Delta V = 0 \qquad \text{in } \Omega_- \tag{3.6.30}$$

$$\Delta V = 0 \qquad \text{in } \Omega_+ \tag{3.6.31}$$

$$V = V_\Gamma \qquad \text{on } \Gamma \tag{3.6.32}$$

$$\left[\!\left[\frac{\partial V}{\partial n}\right]\!\right]_\Gamma = 0 \qquad \text{on } \Gamma \tag{3.6.33}$$

$$\frac{\partial V}{\partial n} = 0 \qquad \text{on } \partial\Omega \tag{3.6.34}$$

This is similar to our method of reinitialization discussed in Section 3.5/

3.7 Solve Field Equations

In many problems the velocity of the moving interface is determined by some additional quantity more than simple geometry (e.g. curvature). For example, a temperature T can be defined on the entire domain. The velocity may be defined as the jump in the gradient of T over the interface in the normal direction, designated as $[\![\frac{\partial T}{\partial n}]\!]_\Gamma$. The temperature itself changes in time according to the field equations.

After the interface-fitting step, the domain is partitioned onto the outer region Ω_+ and the inner region Ω_- separated by the interface-fitted edges that represent the interface Γ. This allows us to impose boundary conditions along the interface when we solve the field equations.

The boundary conditions of the field equations may be given on the interface, so once we interface-fit the mesh to the interface we can solve the equations in the usual way. We don't have to smooth any parameters across the interface as is common with purely finite difference level set methods.

If the gradient jump across the interface in the normal direction contributes to the velocity, then we calculate this on each of the interface edges. So we get $[\![\frac{\partial T}{\partial n}]\!]$ on each of the interface edges. If we want values on the interface vertices, then we convert the edge jump values to the values on the vertices by averaging the values on the edges that meet at that particular vertex.

Chapter 4

Numerical Analysis

4.1 Curvature Approximation

In this section we present a way to calculate the curvature of the level sets of ϕ. In particular, this allows us to calculate the curvature of our interface which is represented by the zero level set of ϕ. We use a local L^2-projection method that results in $\mathcal{O}(h^2)$ accuracy.

Let ω be an arbitrary patch of triangles that share the common vertex v_0. (See figure 4.1.) Let N_0 be the piecewise linear function that is one at vertex v_0, zero on other vertices and linear on each triangle. Let ϕ_h be the piecewise linear approximation of a function ϕ on the mesh. The idea is to approximate the curvature of the level set of ϕ passing through vertex v_0 by using an L^2-projection on ω. This method only requires information on the local patch ω. We approximate the curvature at vertex v_0 by finding κ_0 to make the following equation true:

$$\int_\omega \kappa_0 N_0 \, dx = \int_\omega \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|} N_0 \, dx = -\int_\omega \frac{\nabla \phi}{|\nabla \phi|} \cdot \nabla N_0 \, dx + \int_{\partial \omega} \left(\frac{\nabla \phi}{|\nabla \phi|} \cdot \mathbf{n} \right) N_0 \, dx \tag{4.1.1}$$

Note that the boundary integral is zero since N_0 is zero on $\partial\omega$, so the above equation can be simplified to give a formula for the curvature approximation at vertex v_0:

$$\kappa_0 = -\frac{\int_\omega \frac{\nabla \phi}{|\nabla \phi|} \cdot \nabla N_0 \, dx}{\int_\omega N_0 \, dx}. \tag{4.1.2}$$

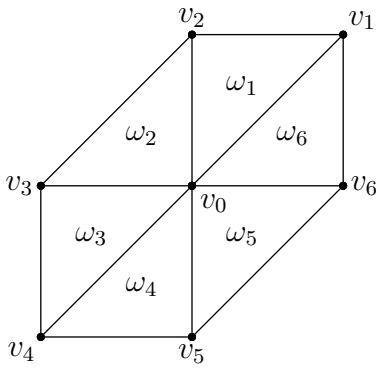


Figure 4.1: A patch of triangles around a vertex v_0 where $\omega = \cup_{i=1}^{6}\omega_i$

.

On a regular grid this gives a $\mathcal{O}(h^2)$ approximation to the curvature where h is the distance between grid points.

Consider the piecewise linear approximation to ϕ on $\omega = \cup_{i=1}^{6}\omega_i$. Let $\phi_i = \phi(v_i)$. The integral in the numerator of (4.1.2) can be broken down into integrals over each of the triangles ω_i on ω.

$$
\begin{aligned}
-\int_{\omega_1} \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx &= \frac{(\phi_2-\phi_0)h}{2\sqrt{(\phi_1-\phi_2)^2+(\phi_2-\phi_0)^2}}\\
-\int_{\omega_2} \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx &= \frac{(\phi_2+\phi_3-2\phi_0)h}{2\sqrt{(\phi_0-\phi_3)^2+(\phi_2-\phi_0)^2}}\\
-\int_{\omega_3} \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx &= \frac{(\phi_3-\phi_0)h}{2\sqrt{(\phi_0-\phi_3)^2+(\phi_3-\phi_4)^2}}\\
-\int_{\omega_4} \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx &= \frac{(\phi_5-\phi_0)h}{2\sqrt{(\phi_5-\phi_4)^2+(\phi_0-\phi_5)^2}}\\
-\int_{\omega_5} \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx &= \frac{(\phi_5+\phi_6-2\phi_0)h}{2\sqrt{(\phi_6-\phi_0)^2+(\phi_0-\phi_5)^2}}\\
-\int_{\omega_6} \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx &= \frac{(\phi_6-\phi_0)h}{2\sqrt{(\phi_6-\phi_0)^2+(\phi_1-\phi_6)^2}}
\end{aligned}
\tag{4.1.3}
$$

The $\{\phi_i\}_{i=1}^{6}$ in these equations can all be put in terms of ϕ_0 and the derivatives of ϕ at v_0 using Taylor series expansions. For convenience let $\phi_x = \phi_x(v_0)$, $\phi_y = \phi_y(v_0)$, $\phi_{xy} = \phi_{xy}(v_0)$, etc. Additionally, each formula can be expanded in a power series

in h to give

$$-\int_{\omega_1}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = \frac{\phi_y}{2(\phi_x^2+\phi_y^2)^{1/2}}h - \frac{(\phi_x\phi_y\phi_{xx}+2\phi_x\phi_y\phi_{xy}-\phi_x^2\phi_{yy})}{4(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^3)$$

$$-\int_{\omega_2}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = -\frac{\phi_x-\phi_y}{2(\phi_x^2+\phi_y^2)^{1/2}}h + \frac{(\phi_x^2\phi_{yy}+\phi_x\phi_y\phi_{xx}+\phi_y^2\phi_{xx}+\phi_x\phi_y\phi_{yy})}{4(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^3)$$

$$-\int_{\omega_3}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = -\frac{\phi_x}{2(\phi_x^2+\phi_y^2)^{1/2}}h - \frac{(\phi_x\phi_y\phi_{yy}+2\phi_x\phi_y\phi_{xy}-\phi_y^2\phi_{xx})}{4(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^3)$$

$$-\int_{\omega_4}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = -\frac{\phi_y}{2(\phi_x^2+\phi_y^2)^{1/2}}h - \frac{(\phi_x\phi_y\phi_{xx}+2\phi_x\phi_y\phi_{xy}-\phi_x^2\phi_{yy})}{4(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^3)$$

$$-\int_{\omega_5}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = -\frac{\phi_y-\phi_x}{2(\phi_x^2+\phi_y^2)^{1/2}}h + \frac{(\phi_x^2\phi_{yy}+\phi_x\phi_y\phi_{xx}+\phi_y^2\phi_{xx}+\phi_x\phi_y\phi_{yy})}{4(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^3)$$

$$-\int_{\omega_6}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = \frac{\phi_x}{2(\phi_x^2+\phi_y^2)^{1/2}}h - \frac{(\phi_x\phi_y\phi_{yy}+2\phi_x\phi_y\phi_{xy}-\phi_y^2\phi_{xx})}{4(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^3)$$

Summing these equations gives

$$-\int_{\omega}\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx = \frac{\phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-2\phi_x\phi_y\phi_{xy}}{(\phi_x^2+\phi_y^2)^{3/2}}h^2+\mathcal{O}(h^4). \tag{4.1.4}$$

(The $\mathcal{O}(h^3)$ terms cancel along with the $\mathcal{O}(h)$ terms.) This and the fact that $\int_\omega N_0\,dx = h^2$ gives the approximation of the curvature at v_0:

$$\kappa_0 = -\frac{\int_\omega \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx}{\int_\omega N_0\,dx} = \frac{\phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-2\phi_x\phi_y\phi_{xy}}{(\phi_x^2+\phi_y^2)^{3/2}} + \mathcal{O}(h^2) \tag{4.1.5}$$

$$= \nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right) + \mathcal{O}(h^2) \tag{4.1.6}$$

It is interesting to note that the largest curvature that can be calculated is inversely proportional to the grid spacing h. Since we always have

$$\left|\frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\right| \le \frac{\sqrt{2}}{h}, \tag{4.1.7}$$

this means

$$\left|\frac{\int_\omega \frac{\nabla\phi}{|\nabla\phi|}\cdot\nabla N_0\,dx}{\int_\omega N_0\,dx}\right| \le \left|\frac{\int_\omega dx}{\int_\omega N_0\,dx}\right|\cdot\frac{\sqrt{2}}{h} = \frac{3h^2}{h^2}\cdot\frac{\sqrt{2}}{h} = \frac{3\sqrt{2}}{h} \tag{4.1.8}$$

See figure 4.6 which illustrates this limit on various mesh sizes.

4.2 Examples of Curvature Approximation

We test this method by calculating the curvature of a circular interface using different grid sizes. The domain is a square with side length one. The level set function ϕ is a cone centered in the center of the domain (see figure 4.2). Thus, the level sets of ϕ are circles whose curvatures are $\kappa = 1/\text{radius}$. The zero level set of ϕ (representing our interface) is a circle of radius $1/4$ with $\kappa = 4$.

We first calculate the curvature on the domain. This gives a function whose value at a particular point is the approximate curvature of the level set of ϕ at that point. We then refine the mesh to an interface-fitted mesh (as explained in section 3.4). The value of the curvature on the interface nodes is interpolated from the pre-refinement curvature calculation.

We repeat this procedure on several grids from coarse to fine. Figure 4.2 shows the level set function on a particular mesh. Figure 4.3 shows the calculated curvature on the domain. Note that near the center of the domain the level sets become small circles whose curvatures become infinite, so we cannot expect high

accuracy in this area. Figure 4.4 shows the absolute error in the curvature calculation across the domain. Note how the error is higher on the domain's boundary which we expect because the approximation is not $\mathcal{O}(h^2)$ there. Also, the error is higher near the center where the curvature should become infinite. The interface is not located near these areas so the error is smaller on and near the interface. Figure 4.5 is a log-log graph of the maximum relative error,

$$\text{Relative Error} = \frac{|\kappa - \kappa_{calculated}|}{\kappa}, \tag{4.2.9}$$

on the interface for several grid sizes. A simple linear regression on the data in the graph gives a best fit line with a slope of 2.065. This suggests, as expected, the relative error is $\mathcal{O}(h^2)$, where h is the grid spacing.

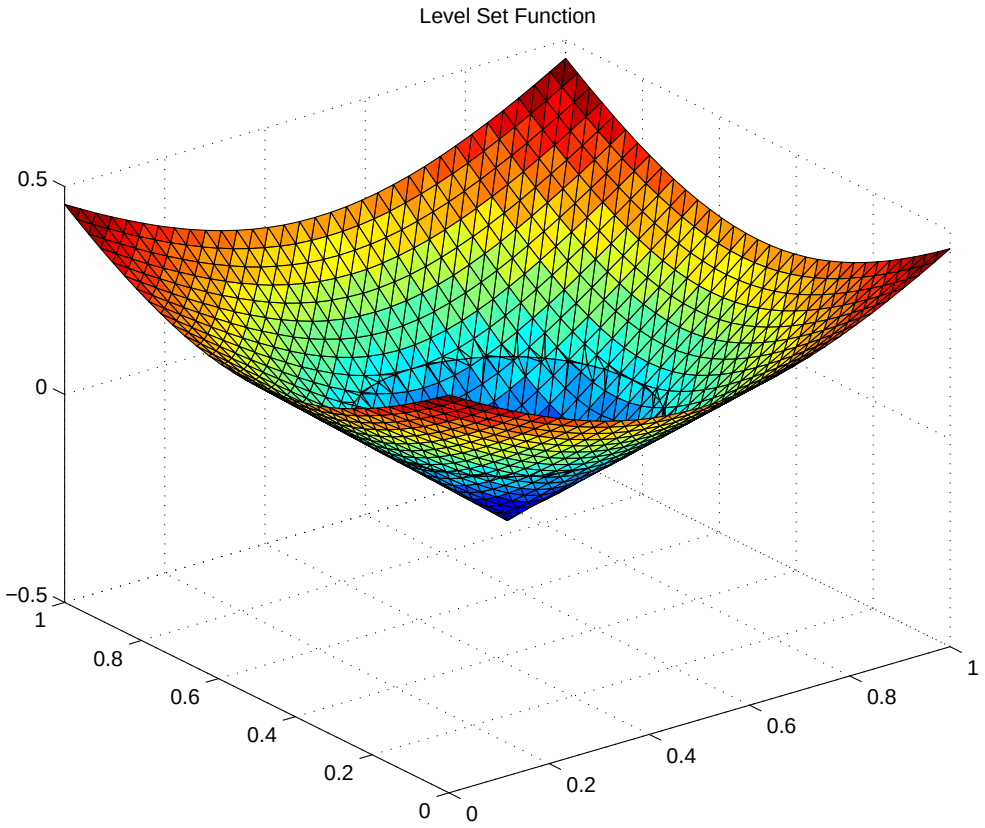


Figure 4.2: A cone shaped level set function used to test the curvature approximation. The interface is a circle of radius $\frac{1}{2}$.

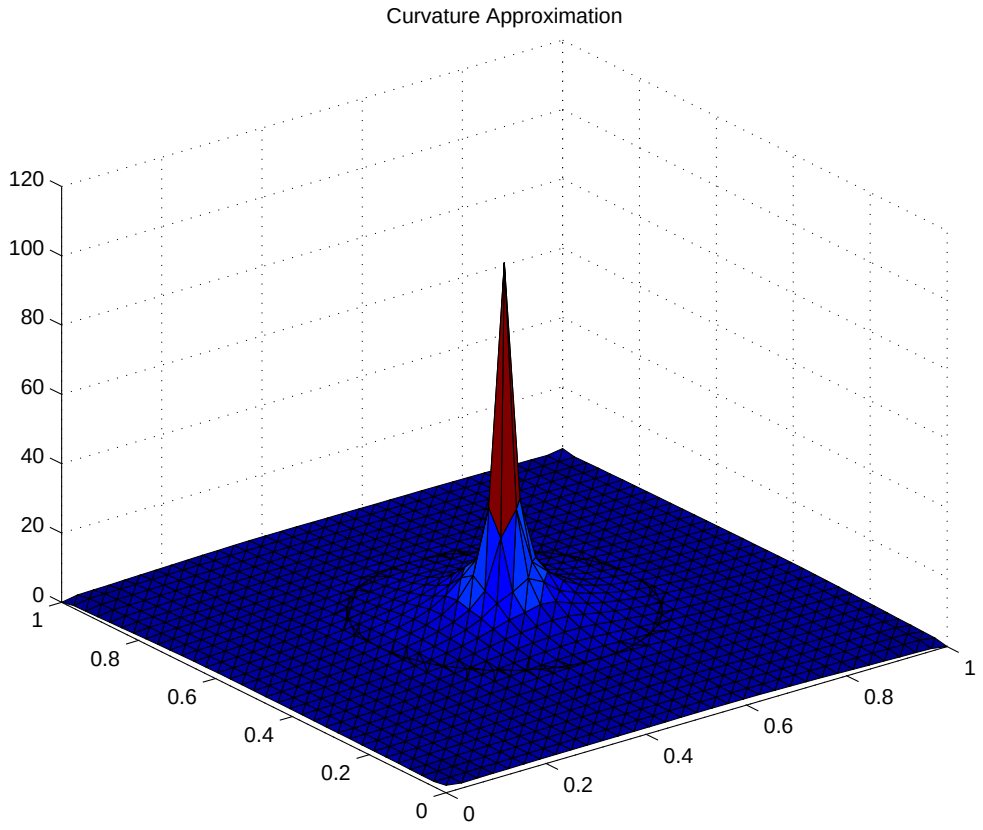


Figure 4.3: Calculated curvature of the level set function. $\kappa = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right)$. Grid spacing $h = 1/32$

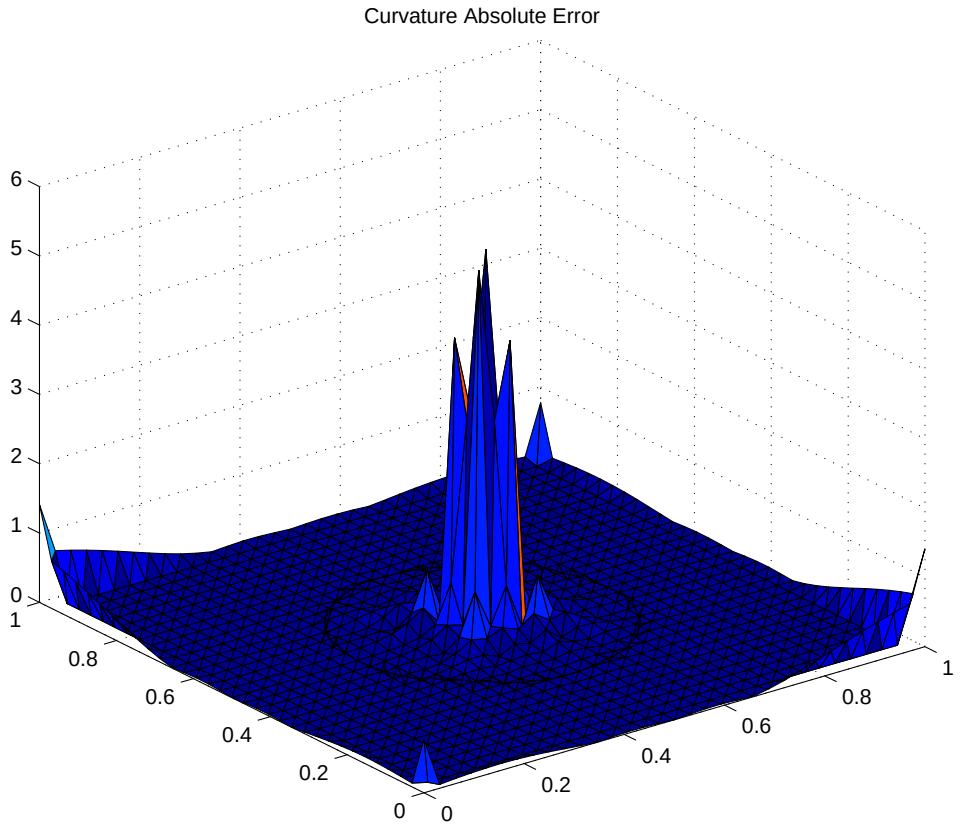


Figure 4.4: Calculated curvature absolute error on the domain. Grid spacing $h = 1/32$

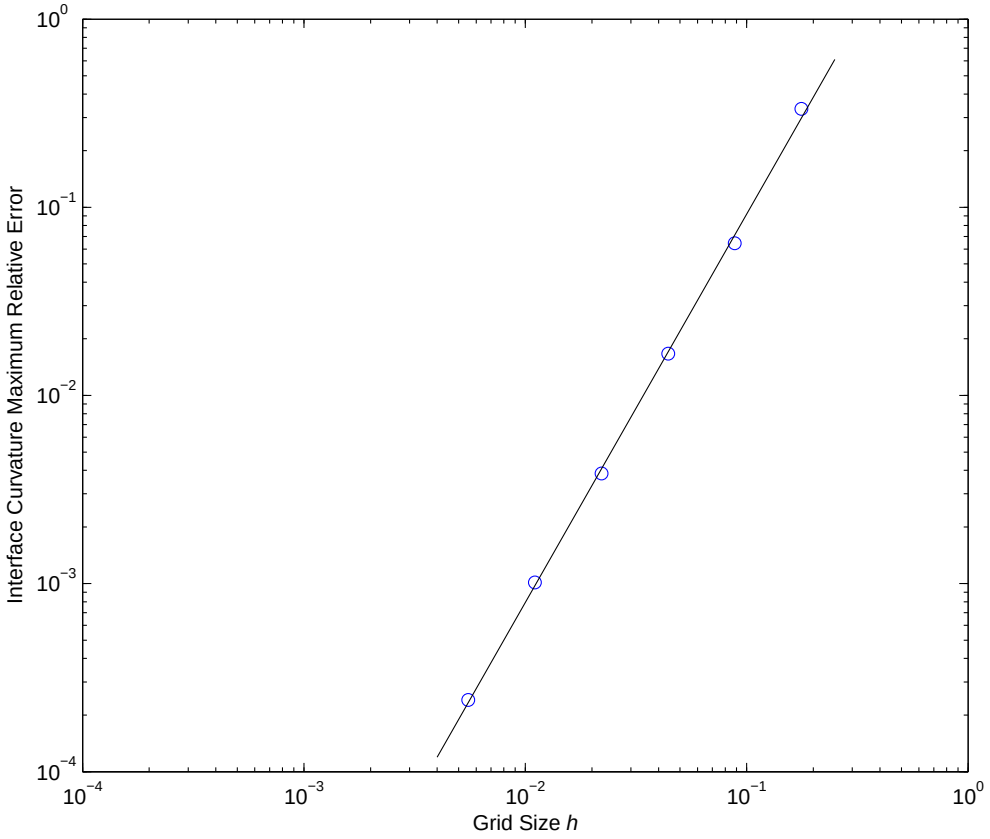


Figure 4.5: The maximum relative error of the calculated curvature on a circular interface versus grid spacing h. The simple linear regression line is plotted and it has slope 2.065

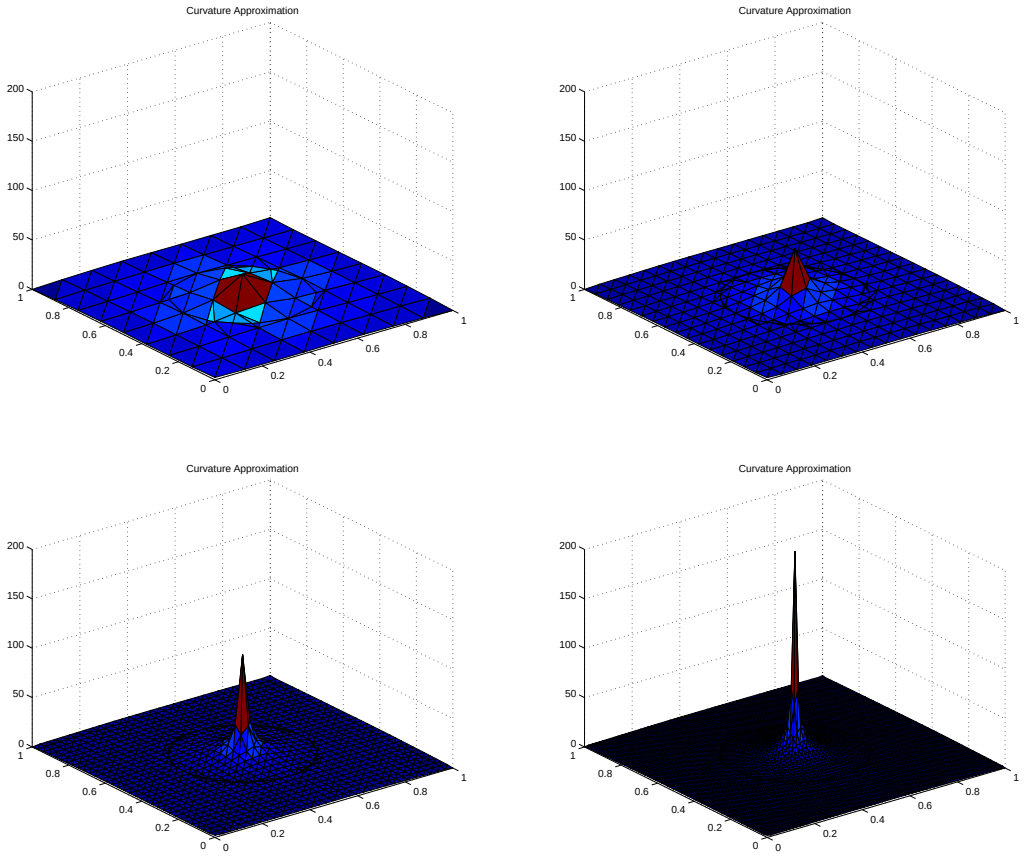


Figure 4.6: Calculated curvature of the level set function on various grid sizes: $h = 1/8,\ 1/16,\ 1/32,\ 1/64$.

Chapter 5

Applications: Solidification

Here we demonstrate the algorithm by using it to solve Stefan problems. The examples model dendritic solidification. This demonstrates that using our interface-fitted finite element level set method produces similar results to those previously produced by a strictly finite difference scheme without interface-fitting [CMOS97].

We simulate a frozen seed placed into a supercooled liquid. The frozen seed is represented by the area Ω_-, and the supercooled liquid is represented by the area Ω_+. The interface between these areas is Γ. As liquid solidifies around the frozen seed, Γ changes shape. Γ moves in its normal direction in proportion to the jump in the gradient of the temperature T across Γ. This is called the Stefan condition, written mathematically as $V_{\mathbf{n}}\Big|_\Gamma = -\left[\!\left[\frac{\partial T}{\partial n}\right]\!\right]_\Gamma$. The temperature changes according to the heat equation, and it is influenced by the changing shape of Γ. The temperature on the interface Γ is governed by the Gibbs-Thompson relation $T(x,t) = -\varepsilon_C\kappa - \varepsilon_V V_{\mathbf{n}}$ where ε_C is the surface tension coefficient, κ is the interface curvature, ε_V is the molecular kinetic coefficient, and $V_{\mathbf{n}}$ is the normal velocity of the interface. It is the dependence between the evolution of Γ and T that makes this problem interesting.

Mathematically, the problem is stated as

$$T_t(x,t) = \Delta T \qquad \text{in } \Omega_- \cup \Omega_+, \tag{5.0.1}$$

$$T(x,t) = -\varepsilon_C \kappa - \varepsilon_V V_{\mathbf{n}} \qquad \text{on } \Gamma \text{ (Gibbs-Thompson relation)}, \tag{5.0.2}$$

$$\frac{\partial T}{\partial n} = 0 \qquad \text{on } \partial\Omega, \tag{5.0.3}$$

$$V_{\mathbf{n}} = -\left[\!\left[\frac{\partial T}{\partial n}\right]\!\right] \qquad \text{on } \Gamma \text{ (Stefan condition)} \tag{5.0.4}$$

$$T(x,0) = 0, \qquad \text{on } \Omega_- \tag{5.0.5}$$

$$T(x,0) = -\frac{1}{2}, \qquad \text{on } \Omega_+ \tag{5.0.6}$$

5.1 Algorithm

We follow the algorithm as explained in section 3.1 and summarize the details of how it applies to this problem.

Step 1 Choose in initial base mesh with spacing appropriate for the problem. Input parameters and initialize the level set function. The input parameters are the values or formulas for ε_C and ε_V that defined the Gibbs-Thompson relation and the initial temperature T on Ω_- and Ω_+. The level set function ϕ is initialized to have the zero level set in the desired shape of Γ.

Step 2 Calculate the curvature κ as described in section 4.1

Step 3 Interface-fit the grid to Γ. Vertices and edges are added where $\phi = 0$ and additional edges are added to make the mesh conforming. This explicitly separates the grid into the Ω_- region and the Ω_+ region. The calculated curvature κ is interpolated onto Γ since it is used in the Gibbs-Thompson relation.

Step 4 Solve the heat equation

$$T_t(x,t) = \Delta T \qquad \text{in } \Omega_- \cup \Omega_+, \tag{5.1.7}$$

$$T(x,t) = -\varepsilon_C \kappa - \varepsilon_V V_{\mathbf{n}} \qquad \text{on } \Gamma, \tag{5.1.8}$$

$$\frac{\partial T}{\partial n} = 0 \qquad \text{on } \partial\Omega \tag{5.1.9}$$

for one time step.

Step 5 Calculate the normal velocity $V_{\mathbf{n}} = -\left[\!\left[\frac{\partial T}{\partial n}\right]\!\right]_{\Gamma}$ on Γ.

Step6 Extend $V_{\mathbf{n}}$ onto the rest of the domain using the method explained in Section 3.6

Step 7 Unrefine the mesh back to the base mesh by simply throwing away the interface-fitted refinements.

Step 8 Evolve ϕ on the base mesh by solving the level set equation $\phi_t + V_n|\nabla\phi| = 0$ for one time step.

Step 9 Reinitialize the level-set function ϕ, if necessary.

Step 10 Go to *Step 2.*

5.2 Varying the Surface Tension Coefficient ε_C

We vary our choice of ε_C and the initial shape of Γ to produce different results for comparison. ε_C can be zero, representing no surface tension; ε_C can be a constant, representing isotropic surface tension; or ε_C can vary based on the direction of the normal to Γ, representing anisotropic surface tension. We let $\varepsilon_V = 0$, so the Gibbs-Thompson relation becomes simply $T(x,t) = -\varepsilon_C\kappa$. In the examples the initial frozen seed is a small pentagon.

Figure 5.1 shows the time progression of the case where $\varepsilon_C = 0$, representing no surface tension effects. As time progresses, the pentagon initially expands to form five dendritic arms, but these arms unstably split as the growth proceeds.

Figure 5.2 shows the case where $\varepsilon_C = 0.001$, representing isotropic surface tension effects. The initial frozen seed expands to form five dendritic arms, but unlike the previous case with no surface tension, the dendritic arms do not split. Instead, they expand outward with a smooth edge.

Figure 5.3 shows the case where $\varepsilon_C = \overline{\varepsilon}_C\left(\frac{8}{3}\sin^4\left(2(\theta - 45°)\right)\right)$ with $\overline{\varepsilon}_C = 0.001$, representing anisotropic surface tension effects. Growth is favored on the

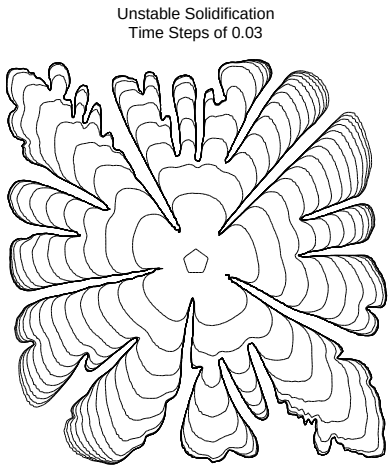


Figure 5.1: Unstable solidification, isotropic surface tension $\varepsilon_C = 0$

interface where the normal direction is diagonal while growth is inhibited where the interface normal direction is vertical or horizontal.

In each case we take the domain to be a square, $\Omega = [-1.5, 1.5] \times [-1.5, 1.5]$. We use a 129×129 grid for the base mesh. The time step is $\Delta t = 0.00025$ and the final time is 0.4.

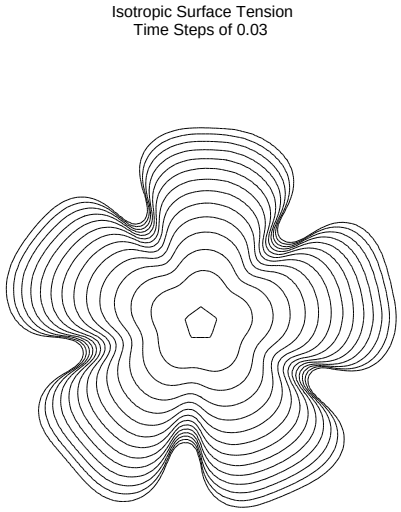


Figure 5.2: Solidification with isotropic surface tension $\varepsilon_C = 0.001$

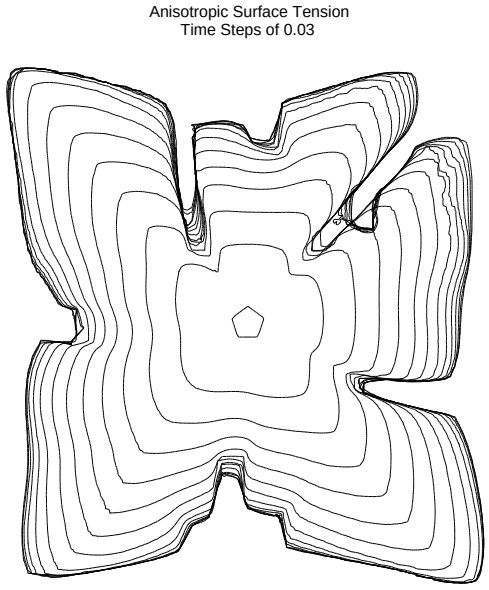


Figure 5.3: Solidification with anisotropic surface tension

5.3 Varying the Phase Angle of an Anisotropy

Here we vary the phase angle θ_0 of the anisotropic surface tension and anisotropic kinetic effects. The Gibbs-Thompson relation $T(x,t) = -\varepsilon_C\kappa - \varepsilon_V V_{\mathbf{n}}$ is defined by

$$\varepsilon_C(\mathbf{n}) = \overline{\varepsilon}_C\left[1 - \cos\left(4(\theta + \theta_0)\right)\right] \tag{5.3.10}$$

$$\varepsilon_V(\mathbf{n}) = \overline{\varepsilon}_V\left[1 - \cos\left(4(\theta + \theta_0)\right)\right] \tag{5.3.11}$$

where θ is the angle formed by the outward normal $\mathbf{n}$ and the positive x-axis measured counterclockwise. These give a fourfold anisotropy. Our initial interface representing the frozen seed is an eight-sided star shape.

Figure 5.4 shows that the growth is favored in the vertical and horizontal directions when the phase angle is 0°. Figure 5.5 shows that the growth is favored in the diagonal directions when the phase angle is 45°.

In our examples we let $\overline{\varepsilon}_C = \overline{\varepsilon}_V = 0.001$. The domain is a square, $\Omega = [-1,1] \times [-1,1]$. We use a 129×129 grid for the base mesh. The time step is $\Delta t = 0.0001$ and the final time is 0.4. In both figures the shape is shown at $t = 0, 0.01, 0.02, 0.03$, and 0.04.

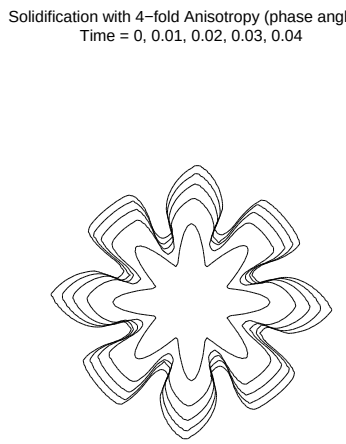


Figure 5.4: Solidification with anisotropic surface tension (phase angle 0°)

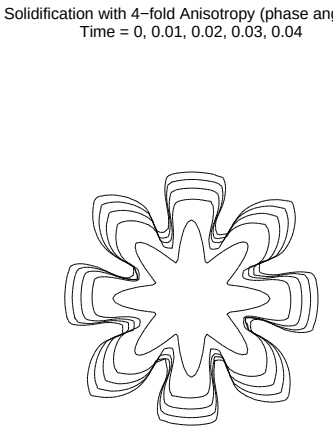


Figure 5.5: Solidification with anisotropic surface tension (phase angle 45°)

Chapter 6

Applications: Variational Implicit Solvation of Molecules

6.1 Molecular Solute-Solvent Interface

Here we demonstrate the algorithm by using it to find the optimal solute-solvent interface of a solvation system.

We use a variational implicit solvent model for equilibrium solvation systems of nonpolar molecules [DSM06a, DSM06b, CDLM08, CXD$^+$09]. Central in this model is an effective free energy functional of the solute-solvent interface and solute particle positions. This free energy includes the interface energy of the solute and the solute-solvent van der Waals type interactions.

Our underlying system of nonpolar molecules in a solution is divided geometrically into three parts: the solute region Ω_s, the solvent (e.g., water) region Ω_w, and the corresponding solute-solvent interface Γ which is the boundary separating the solute region Ω_m and the solvent region Ω_w. We assume that there is a sharp interface that separates the solvent and solutes, and we treat the solvent as a continuum. We also assume that there are N solute atoms in the system that are located at $\mathbf{x}_1, \ldots, \mathbf{x}_N$ inside Ω_s. These solute particles are treated explicitly.

We assume that the experimentally observed equilibrium solvation systems consist of the molecular surface Γ, the solute particles $\mathbf{x}_i$, and electrostatic potential

that together minimize an effective solvation free energy. We define the following effective free energy functional:

$$G = G\left[\Gamma; \mathbf{x}_1, \ldots, \mathbf{x}_N\right] = \int_\Gamma \gamma \, dS + \rho_w \sum_{i=1}^{N} \int_{\Omega_w} U_{sw}\left(|\mathbf{x} - \mathbf{x}_i|\right) \, dV \tag{6.1.1}$$

The first term $\int_\Gamma \gamma \, dS$ is the surface energy, where γ is the surface tension. To simplify the model, we take γ to be a constant. In two dimensions this term is a line integral along Γ.

The second term

$$\rho_w \sum_{i=1}^{N} \int_{\Omega_w} U_{sw}\left(|\mathbf{x} - \mathbf{x}_i|\right) \, dV \tag{6.1.2}$$

is the non-electrostatic, van der Waals type interaction energy between solute particles $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and solvent molecules governed by a Lennard-Jones type potential

$$U_{sw}(r) = 4\varepsilon_{sw}\left[\left(\frac{\sigma_{sw}}{r}\right)^{12} - \left(\frac{\sigma_{sw}}{r}\right)^{6}\right]. \tag{6.1.3}$$

This is an integral over the solvent region. The parameter ρ_w is the solvent density which we take to be constant. Note that the parameters ε_{sw} and σ_{sw} can vary with different particles.

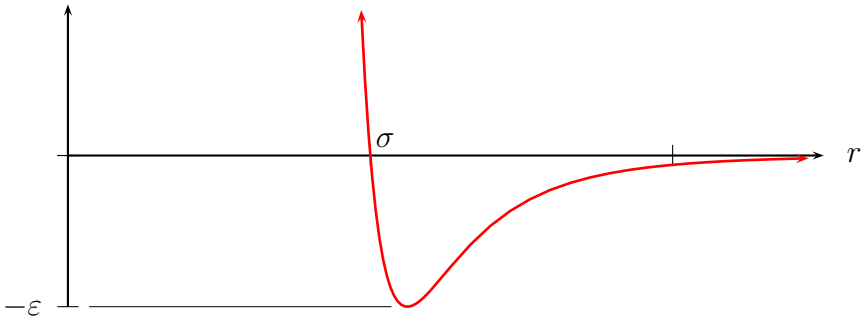


Figure 6.1: The Lennard-Jones Potential $U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$

6.2 Algorithm

We follow the algorithm as explained in section 3.1 and summarize the details of how it applies to this problem.

Step 1 Choose an initial base mesh with spacing appropriate for the problem. Input parameters and initialize the level set function. The input parameters are the surface tension constant γ, the Lennard-Jones energy parameters σ_{sw}, ε_{sw} (for each particle) and the solute particle locations $\mathbf{x}_1, \ldots, \mathbf{x}_N$. The level set function ϕ is initialized as the distance from the nearest particle minus a positive constant. The constant is chosen so that the initial interface defines Ω_m as a single connected region.

Step 2 Calculate the curvature κ as described in section 4.1

Step 3 Interface-fit the grid to Γ. Vertices and edges are added where $\phi = 0$ and additional edges are added to make the mesh conforming. This explicitly separates the grid into an Ω_w region and an Ω_m region.

Step 4 Calculate the Lennard-Jones potential term $\rho_w \sum_{i=1}^{N} U_{sw}(|\mathbf{x} - \mathbf{x}_i|)$ on Γ (at the vertices on Γ).

Step 5 Extend the Lennard-Jones potential term away from Γ as explained in section 3.6

Step 6 Evolve ϕ. The normal velocity V_n is the sum of the curvature term $\gamma\kappa$ and the extended potential term (from step 5). ϕ is evolved on the regular grid by solving the level set equation $\phi_t + V_n|\nabla\phi| = 0$ for one time step.

Step 7 Reinitialize the level-set function ϕ, if necessary.

Step 8 If the energy functional did not decrease, then end; otherwise, go to *Step 2.*

6.3 A Three Atom System

In this example we let $\Omega = [0,1]\times[0,1]$. Two atoms are located at $(0.3, 0.35)$ and $(0.6, 0.65)$ and both have $\varepsilon = 0.005$ and $\sigma = 0.15$. The third atom is located at $(0.5, 0.2)$ and has $\varepsilon = 0.003$ and $\sigma = 0.1$. We initialize ϕ to be at least $1.5\sigma_i$ from atom i. The free energy functional parameters are $\gamma = 0.0001$ and $\rho = 0.5$.

Figures 6.2, 6.3 and 6.4 show the progression of Γ to the steady state that represents the optimal solute-solvent interface.

Figure 6.5 shows the value of the surface energy, van der Waals energy, and the total free energy as the algorithm proceeds. The top line represents the surface energy and we see it increase from time step 75 to time step 250 as the interface becomes longer. As the surface energy increases it is more than offset by the decrease of the van der Waals energy so that the total free energy (surface energy plus van der Waals energy) monotonically decreases to a steady state.

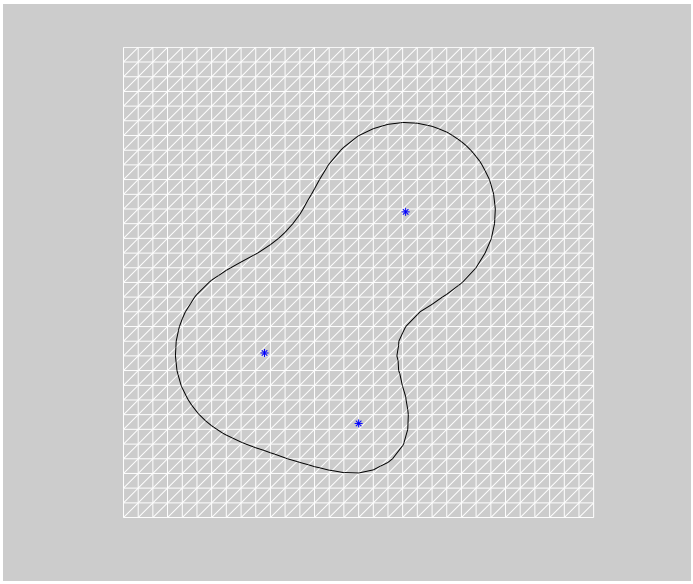


Figure 6.2: A three atom solute-solvent interface example, start position

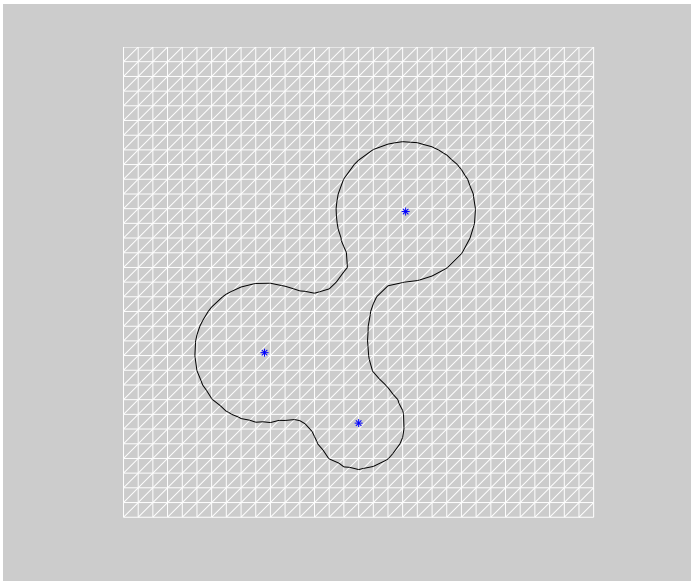


Figure 6.3: A three atom solute-solvent interface example, intermediate position

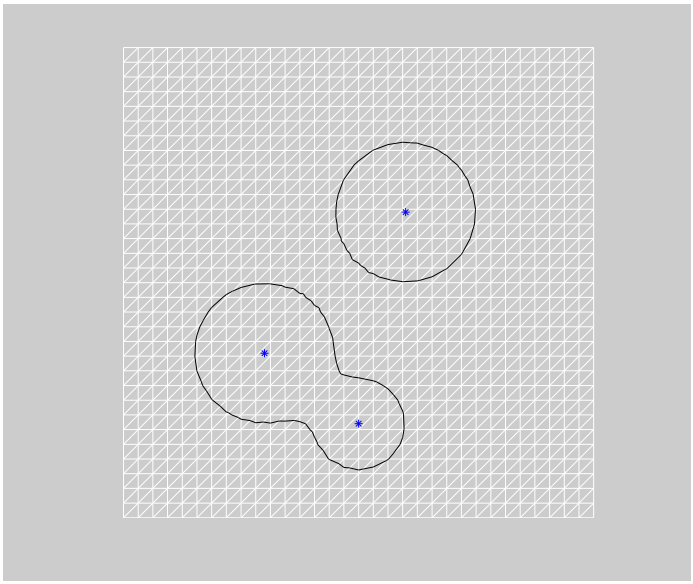


Figure 6.4: A three atom solute-solvent interface example, end position

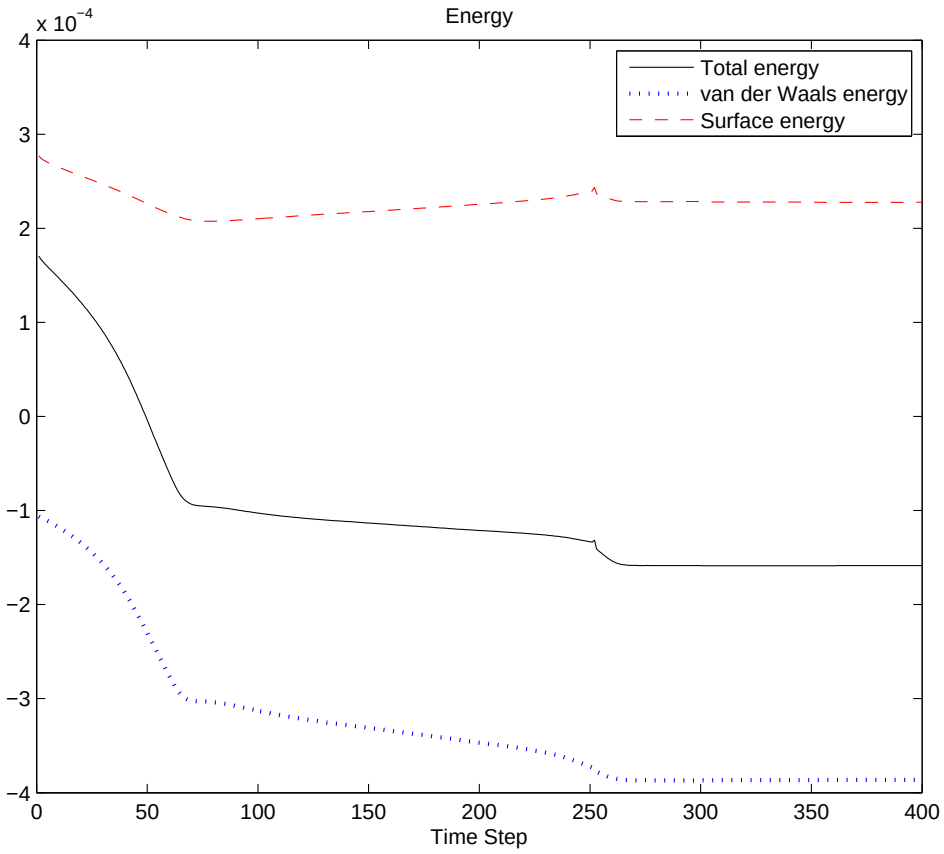


Figure 6.5: Energy vs. step number of a three atom solute-solvent interface example. The top (dashed) line is the surface energy. The bottom (dotted) line is the van der Waals energy. The middle (solid) line is the sum, total free energy.

Chapter 7

Discussions and Conclusions

7.1 Summary of Results

We presented a new level set method. This is an interface-fitted finite element based level set method. We use a base mesh that is regular, but at each time step we refine the mesh to produce an interface-fitted mesh. This allows us to computationally separate the regions separated by the interface. By explicitly separating the regions, we can separately solve field equations with different parameters on each region. Also, by explicitly locating the interface we can calculate certain quantities with greater accuracy, for example, functions that vary rapidly in space such as the Lennard-Jones potential.

We presented a new method to reinitialize the level set function by solving a Poisson's equation type problem on the interface-fitted mesh. Other reinitialization methods tend to move the location of the interface, but our reinitialization method does not move the interface.

A new method of velocity extension was also given. This method was similar to the reinitialization method. It extends the velocity values from the interface to the rest of the mesh by solving a Poisson's equation on the interface-fitted mesh and using the interface velocity values as a boundary condition.

We presented a new way to calculate the curvature of an interface on a regular uniform mesh with linear basis functions. This method uses the quantities $\nabla\phi$ and $|\nabla\phi|$ on each triangle, and it has $\mathcal{O}(h^2)$ accuracy. While other relatively

simple $\mathcal{O}(h^2)$ formulas already exist for curvature calculation, our method would be useful if $\nabla\phi$ and $|\nabla\phi|$ are already calculated, thus increasing the efficiency of an algorithm.

We applied our new method to two types of applications, simulating dendritic solidification and finding optimal solute-solvent interfaces of solvation systems. On the solidification problems, our method produced results similar to those produced previously using a finite difference based level set method [CMOS97]. On the solute-solvent interface problems, our method converged to the optimal minimum energy interface.

7.2 Comparison with Finite Difference Based Level Set Method

Finite difference level set methods are well established. Efficient high order schemes exist to solve the level set equation. With purely finite difference methods, the interface is not explicitly located except when a graph of the interface is needed. Many level set problems involve solving partial differential equations (PDEs) to determine the interface velocity, but jump conditions across the interface are crucial to solving many PDE problems, so special techniques must be employed to solve these PDEs on finite difference grids. A popular technique is to smear the jump condition across several grid points near the interface. This requires some care since too little or too much smearing will degrade the PDE's solution.

With an interface-fitted mesh, the jump conditions can be explicitly enforced and calculated, and the PDEs can be solved using finite element methods, but since our method uses a base mesh that is regular and uniform, we still have the option of using finite difference methods to solve the level set equation. The drawback of interface-fitting is the added computational cost associated with locating the interface and refining the mesh at each time step.

7.3 Outlook

7.3.1 Extension to Three Dimensions

We believe this method can be applied in a three dimensional setting. The main difficulty in three dimensions is interface-fitting the base mesh at each time step. This would require working out and programming the refinements necessary to make the mesh conforming after adding the interface-fitted surfaces. In two dimensions interface-fitting means adding new vertices and edges, and triangles typically are split into three new triangles near the interface. In three dimensions, tetrahedrons would be split by the interface surface and new vertices, edges, and surfaces would need to be added.

The velocity extension and reinitialization methods should be easily adapted to three dimensions since they are based on solving Poisson's equation.

It would appear that a curvature calculation similar to the one presented in chapter 4 could be used on a three dimensional mesh and obtain $\mathcal{O}(h^2)$ accuracy.

7.3.2 Adaptive Finite Element for Solving Field Equations

The use of a finite element method, after interface-fitting the mesh, permits the use of adaptive refinement to increase the accuracy of PDE solutions. Most refinement would be needed near the interface for two reasons: the area near the interface is most important for determining the velocity of the interface, and interface-fitting creates irregularly shaped elements near the interface.

In the process of interface-fitting the mesh, very slender triangles will inevitably form near the interface. This can increase errors of field equation solutions. Moving vertices to improve mesh quality would be prohibitively expensive in our scheme since we rely on keeping the base mesh the same on each time step. This strategy would entail projecting values back onto our base mesh on each time step. Because we'd like to keep our base mesh intact, the best way to refine would appear to be adaptive refinement techniques that split elements rather than move vertices.

Bibliography

[CDLM08] J. Che, J. Dzubiella, B. Li, and J. A. McCammon. Electrostatic free energy and its variations in implicit solvent models. *J. Phys. Chem. B*, 112:3058–3069, 2008.

[CDML07] L.T. Cheng, J. Dzubiella, J.A. McCammon, and B. Li. Application of the level-set method to the implicit solvation of nonpolar molecules. *The Journal of Chemical Physics*, 127:084503, 2007.

[Cho80] A.J. Chorin. Flame Advection and Propagation Algorithms. *Journal of Computational Physics*, 35:1, 1980.

[CMOS97] S. Chen, B. Merriman, S. Osher, and P. Smereka. A simple level set method for solving Stefan problems. *Journal of Computational Physics*, 135(1):8–29, 1997.

[CXD+09] L.T. Cheng, Y. Xie, J. Dzubiella, J.A. McCammon, J. Che, and B. Li. Coupling the Level-Set Method with Molecular Mechanics for Variational Implicit Solvation of Nonpolar Molecules. *Journal of Chemical Theory and Computation*, 5(2):257–266, 2009.

[DSM06a] J. Dzubiella, J.M.J. Swanson, and J.A. McCammon. Coupling hydrophobicity, dispersion, and electrostatics in continuum solvent models. *Phys. Rev. Lett.*, 96:087802, 2006.

[DSM06b] J. Dzubiella, J.M.J. Swanson, and J.A. McCammon. Coupling nonpolar and polar solvation free energies in implicit solvent models. *The Journal of Chemical Physics*, 124:084905, 2006.

[HEOC87] A. Harten, B. Engquist, S. Osher, and S.R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. *Journal of Computational Physics*, 71(2):231–303, 1987.

[HEOC97] A. Harten, B. Engquist, S. Osher, and S.R. Chakravarthy. Uniformly High Order Accurate Essentially Non-oscillatory Schemes, III. *Journal of Computational Physics*, 131(1):3–47, 1997.

[OF03] S. Osher and R.P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003.

[OS88] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations. *J. Comput. Phys*, 79:12–49, 1988.

[Set84] J.A. Sethian. Turbulent combustion in open and closed vessels. *Journal of Computational Physics*, 54:425–456, 1984.

[Set99] 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.

[SO88] C.W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, 1988.

[SO89] C.W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes, II. *Journal of Computational Physics*, 83(1):32–78, 1989.

[SP03] S.V. Shepel and S. Paolucci. Finite element level set formulations for modelling multiphase flows. In *Proceedings of the international conference on Computational methods in sciences and engineering*, pages 593–597. World Scientific Publishing Co., Inc. River Edge, NJ, USA, 2003.