ALGORITHMS FOR INCREASING THE EFFICIENCY AND FIDELITY OF FLUID SIMULATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Frank William Losasso Petterson
May 2007
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

______________________________
(Ronald Fedkiw)  Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

______________________________
(Patrick Hanrahan)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

______________________________
(Marc Levoy)

Approved for the University Committee on Graduate Studies.
Abstract

This dissertation presents algorithms for the simulation of fluids. Realistic simulations of smoke, water and fire are amongst the most desired in the special effects industry. These phenomena often contain complex motions and rich visual details that are difficult or impossible to animate by hand. These phenomena are also extremely costly to simulate using current technology.

The first part of this thesis will focus on increasing the efficiency of these simulations. In order to optimize the use of computational resources, we present an adaptive octree mesh or a level of detail approach where more grid cells are placed in visually interesting regions. The enabling technology for this approach is a new technique for discretizing the Poisson equation on an unrestricted octree grid. We will conclude this section by demonstrating the flexibility of the algorithm by several examples.

The rest of the thesis will focus on simulating new phenomena. We will describe a novel technique for melting and burning solid materials including the simulation of the resulting liquid and gas. Our method allows for state of the art techniques to be used for both the solid and the fluid without compromising simulation quality when coupling them together or converting one into the other. In addition, an approach to extend the particle level set method to the simulation of an arbitrary number of regions is presented. Using a separate particle level set method for each region, we present a novel projection algorithm that can be used to provide a dictionary that translates between the vector of level set values and the standard single-valued level set representation. This dictionary allows the standard level set techniques to be used without any additional special treatment.
Acknowledgements

I want to thank my parents, Lasse and Clara, whose influence on my life is obvious to anyone who knows me. They have provided support, encouragement and guidance for the past twenty six years in the best way I could imagine, and for that I owe them more than I could ever give them. I also want to thank my sister, Cecilia, you are smart, kind and fun, and I’ve enjoyed all the time we have spent together.

Next I want to thank my advisor, Ron, for five great years. He is definitely one of the most unique individuals I’ve ever met. Ron has been an advisor, teacher, friend and at times enemy, but throughout it all, I’ve learnt an incredible amount about a surprising array of topics including research, fluids, physics, graphics, visual effects, life, politics and people. Past, present and future, Ron has, is and will be an invaluable advisor.

Having been in Ron’s group, I have had the pleasure of working with some of the best people I’ve ever meet. Specifically, I would like to thank each of my coauthors, Tamar Shinar, Andrew Selle, Geoffrey Irving, Jerry Talton, Nipun Kwatra, Fred Gibou, Doug Enright and Eran Guendelman. Special thanks go out to Tamar, Andy, Geoff and Jerry, whom I’ve worked the closest with, and whom I will always consider friends as well as colleagues. I would also like to thank the other members of Ron’s group, past and present, whom I had the pleasure of meeting. The graduate school experience would not have been the same without the benefit of them. Thanks to Robert Bridson, Neil Molino, Igor Neverov, Joseph Teran, Zhaosheng Bao, Eftychios Sifakis, Duc Nguyen, Jeong-Mo Hong, Robert Strzodka, Avi Robinson-Mosher, Jonathan Su, Kevin Der, and Craig Schroeder.

I would like to thank Ron Fedkiw, Pat Hanrahan and Marc Levoy for sitting on
my qualifying, reading and theses committees. Thanks to Heinz Pitsch for chairing
my thesis committee and to Scott Klemmer for sitting on my thesis committee.

I would also like to thank Joe Warren for introducing me to computer graphics
and convincing me to go to graduate school. The opportunity that Joe gave me and
other computer science undergraduate students was invaluable. My coauthors from
Rice University, Scott Schaefer and Tao Ju, were also an integral part of my choice
to follow in their footsteps with respect to going to get a doctorate, so I thank them
for that.

I have had the pleasure of working with a number of remarkable people during
my time at Industrial Light+Magic. Although there are too many to mention, I’d
like to thank Nick Rasmussen, Kim Libreri, Mohen Leo, Willi Geiger, John Knoll,
David Meny, Joakim Arnessson, Lynwen Brennan and others for showing me that
visual effects and specifically ILM is where I want to work.

Finally, thanks to my fiancee, Rachel, for all of our time together. The last few
years have been incredible, and I’m looking forward to many many more.
# Contents

Abstract iv  
Acknowledgements v  

1 Introduction 1  

2 Fluids on an Octree Data Structure 3  

2.1 Introduction 3  
2.2 Previous Work 6  
2.3 The Octree Data Structure 7  
2.4 Navier Stokes Equations on Octrees 8  

2.4.1 The Divergence Operator 9  
2.4.2 The Pressure Gradient 10  
2.4.3 Accuracy 14  

2.5 Smoke 14  
2.6 Water 15  

2.7 Second order Accuracy on Octrees 17  
2.8 Adaptive Solver Discussion 21  

3 Multiple Interacting Liquids 22  

3.1 Introduction 22  
3.2 Previous Work 24  
3.3 Multiple Level Sets 24  

3.3.1 Projection Method 24
### 3.3.2 Particle Level Set Method

**3.4 Multiple Liquids**

**3.4.1 Poisson Equation**

**3.4.2 Viscosity**

**3.4.3 Viscoelasticity**

**3.5 Adding Air or Empty Regions**

**3.6 Surface Tension**

**3.7 Surface Reactions**

**3.8 Examples**

**3.9 Conclusions and Future Work**

### 4 Melting and Burning Solids into Liquids and Gases

**4.1 Introduction**

**4.2 Previous Work**

**4.3 Eroding Thin Shells**

**4.4 Eroding Volumetric Solids**

**4.5 Creating Fluids from Solids**

**4.6 Creating Liquid**

**4.7 Creating Gas**

**4.8 Two Way Solid Fluid Coupling**

**4.9 Examples**

**4.10 Conclusions and Future Work**

### 5 Coupled SPH and Particle Level Set Simulation

**5.1 Introduction**

**5.2 Previous Work**

**5.3 Particle Level Set Method**

**5.4 Diffuse SPH**

**5.5 Two-way Coupling**

**5.6 Examples**

**5.7 Conclusions**
# 6 Conclusions and Future Work

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Simulation of Fluids on Octree Grids</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Multiple Interacting Liquids</td>
<td>89</td>
</tr>
<tr>
<td>6.3</td>
<td>Melting and Burning</td>
<td>90</td>
</tr>
<tr>
<td>6.4</td>
<td>Large Scale Simulation</td>
<td>90</td>
</tr>
</tbody>
</table>

**Bibliography**

91
List of Figures

2.1 Simulation of smoke past a sphere. The bottom two figures are close up views. The effective resolution is $1024^3$ and the computational time is about 4-5 minutes per frame. 4

2.2 Left: One large cell neighboring four smaller cells. The $u_i^*$ represent the $x$ components of the intermediate velocity $\mathbf{u}^*$ defined at the cell faces. Right: Zoom of one computational cell. The velocity components are defined on the cell faces, while the pressure is defined at the center of the cell. The density, temperature and level set function $\phi$ are stored at the nodes. 8

2.3 Discretization of the pressure gradient. 11

2.4 An ellipsoid slips along through shallow water illustrating our method’s ability to resolve thin sheets. The effective resolution is $512^3$ and the computational time is about 4-5 minutes per frame. 12

2.5 Formation of a milk crown demonstrating the effect of surface tension. We take $\sigma = 0$ (left) and $\sigma = .0005$ (right). The effective resolution is $512^3$ and the computational time is about 4-5 minutes per frame. 17

2.6 Initial quadtree refinement used in Poisson solver accuracy test. 19

3.1 Each of the three regions is independently evolved in time, after which the interface locations do not agree. There are vacuums where all $\phi_i$ are positive, and overlaps where more than one $\phi_i$ is negative. The dotted black line shows the new interface locations after our projection step. 36

3.2 A kinematically controlled sphere splashing into a multi-layer pool ($300 \times 300 \times 200$ grid, 4 phases). 36
3.3 (Left) Two level sets initialized so that properties 1 and 2 hold. (Right) After evolving in time, we obtain the solid lines with overlap (both negative in the middle). The dotted lines show an example result after projection. Not only has the overlap been removed, but the interface location is preserved.

3.4 (Left) The solid lines have overlap on the left (two $\phi_i$ negative) and a vacuum on the right (all $\phi_i$ positive). The average of the smallest two level sets at any point is subtracted from $\vec{\phi}$ to obtain the dotted lines. Not only has the overlap and vacuum been removed, but the smallest $\phi_i$ is preserved and negative at each point preserving interface locations and inside/outside information. (Right) Results after reinitializing each level set to a signed distance function.

3.5 Rayleigh-Taylor instability ($300^3$ grid, 4 phases).

3.6 Viscous letters splash into a pool of water, then change into low density inviscid fuel bubbling up and burning when they hit the surface ($350 \times 200 \times 350$ grid, 10 phases).

3.7 Different viscosity liquids interacting on an inclined plane. ($300 \times 150 \times 240$ grid, 5 phases).

3.8 (Left) One way coupling from liquid to air. (Right) Two way coupling of liquid and air. Note the surface ripples and the unstable stream of liquid in the fully coupled simulation.

3.9 Two drops with high surface tension collide. Green has low density, red high density ($350^3$ grid, 3 phases).

3.10 Two submerged liquids meeting and reacting to create air ($150^3$ grid, 4 phases).

3.11 Oil pouring into water, then catching on fire. Note that the fiery ball is a separate phase of fluid, and that it deforms into the shape of a droplet as it falls ($200^3$ grid, 4 phases).
3.12 An armadillo that starts out viscoelastic, becomes viscous and more dense than the water, then inviscid and lighter than the water, and finally viscoelastic again before another viscoelastic liquid is dropped onto it (250 × 275 × 250 grid, 4 phases). 45

4.1 A sheet of material constrained at its four corners catches fire. When the top corners burn away from the rest of the material, it falls to the ground (121x193x121 grid, 13K triangles). 47

4.2 Red and green refinement of BCS triangles. 50

4.3 A burning sheet of material rendered without the fire to illustrate the smooth boundary of the solid (13K triangles). 51

4.4 Starting with a uniform body-centered square mesh in material space (upper left), we apply red-green refinement and discard triangles completely outside the object (upper right). The simulation is then performed on the red-green parent mesh (lower left). Level set operations are performed by overlaying a quadtree grid on top of the red-green structure, noting that each mesh vertex is also a node in the quadtree grid (lower right). A major advantage of this approach is that the front evolution is carried out on a quadtree mesh in the two-dimensional material space even as the object deforms in three spatial dimensions. 63

4.5 A portion of the BCC lattice. The blue and the green connections depict the two interlaced grids, and the eight red connections at each node lace these two grids together. 64

4.6 Red refinement produces eight children that reside on a BCC lattice of half the size (left). Three types of green refinement are allowed in order to remove T-junctions (right). 64

4.7 An elastic torus bounces and rolls on the ground as it melts, generating thin sheets of liquid (242x121x121 grid, 22K tetrahedra). 65

4.8 Ten rigid body simulated ice cubes are melted by a stream of hot water (100³ grid, 600K total surface triangles). 66
4.9 The melting speed of the ice cubes is derived from a temperature field advected with the liquid velocity. .......................... 67
4.10 Six elastic tori fall onto a hot surface causing them to quickly melt into liquid (181x61x181 grid, 160K total tetrahedra). .......................... 68
4.11 A melting solid partially immersed in liquid. Particles are seeded throughout the solid with the same distribution and properties as the particle level set particles (left). As the solid melts, particles leave the solid and are converted to removed liquid particles. Some of these particles are close enough to grid nodes to convert them to liquid (middle). More liquid is generated as the solid continues melting (right). .... 69
4.12 A predefined level set is used to prescore part of a thin material sheet as inflammable (121x193x85 grid, 37K triangles). ................. 70
4.13 Rigid ice cubes floating and melting in water with full two-way force coupling (100^3 grid, 600K total surface triangles). ............ 71
4.14 Burning cloth draped over an armadillo figurine (89x122x78 grid, 18K triangles). ................................................. 72

5.1 (Left) We start an SPH simulation targeting a uniform particle number density. (Center) We then increase the target particle density causing the liquid to compress. (Right) Finally, we decrease the target particle density, and the fluid expands. ......................... 75
5.2 SPH liquid flowing into a 120 × 240 box. The target particle number density is set to be low everywhere except in a region delineated by outlines of the SIGGRAPH logo. ......................... 76
5.3 Two-way coupling between SPH and the particle level set method to pour water into a glass. ................................. 77
5.4 (Top) A thin level set source pouring onto an upturned bowl on a $150 \times 200 \times 200$ grid. The grid cannot resolve the thin film at the point of impact and catastrophically loses mass. (Middle) The simulation with the removed negative particles visualized. The particles have no notion of volume and thus end up compressed on the bottom right edge of the domain. (Bottom) Our SPH solver treats the removed negative particles as an integral part of the liquid representation, and convincingly conserves volume.

5.5 Two-way coupled SPH and particle level set fluid simulation. The level set is depicted in green. The far left image shows a few negative removed particles generated from the level set that are subsequently simulated with SPH. In the second figure, we have added a source of further SPH particles. After turning off the source in the third figure, we turn on reincorporation of particles in dense regions so that they transition to a level set representation when possible providing for a smoother interface representation.

5.6 A large ocean scene simulated with two-way coupling between our SPH method and the particle level set method. Besides the full two-way coupling, a secondary air simulation is used to generate a second layer of fine-detail mist and foam sourced from the SPH particles.
Chapter 1

Introduction

The increase in computational power at a low cost has in the last few years made simulation of fluid dynamics a viable alternative to manual animation for the special effects industry. The amount of research into fluid simulation in the computer graphics community has also increased significantly as fluid simulation has moved from the domain of super computers to standard desktop personal computers.

This dissertation presents a number of algorithms that make fluid simulation more useful in computer graphics. The focus is on making simulations more efficient, allowing for larger domains and higher resolutions, and on higher fidelity, allowing for simulations of new phenomena and interactivity. The four main contributions presented are:

- An octree based fluid solver that supports arbitrary refinement. A simple first order pressure solver allows for high resolution smoke simulations, and the addition of an octree particle level set method allows for free surface flows. An second order pressure solver is also discussed.

- A multiphase method that extends the particle level set method to support an arbitrary number of phases.

- A technique for melting and burning volumetric and thin shell solids into liquid or gas. A simple method to couple the the solids and fluids is also presented.
• A novel smoothed particle hydrodynamics (SPH) method that uses a background grid to achieve the stability of grid based pressure solvers. We also show how to fully couple the SPH based fluids with a standard particle level set based fluid allowing for simulations of spray and other diffuse water phenomena.

Although the primary target for the methods presented is the visual effects industry, the techniques themselves are general and can be used in any field from computational physics to computer gaming.

Most of the contents in this dissertation is extracted from previous publications in which I am the main or primary author. These publications include [67, 69, 68, 66, 70].
Chapter 2

Fluids on an Octree Data Structure

2.1 Introduction

Realistic simulations of smoke and water are among the most desired in the special effects industry, since they provide the director with explicit control over the environment enabling the creation of otherwise impossible content. These phenomena contain highly complex motions and rich visual detail, especially when they interact with inanimate objects or the actors themselves. Moreover, a significant portion of the entertainment value and much of the believability relies on an adequate representation and presentation of the small scale visual details such as thin films in water, small rolling vortices in smoke, droplets and sprays, etc. Thus, it is desirable to have both simulation and rendering techniques that can deal with levels of detail.

Recent improvements in simulation techniques have led to impressive simulations of both smoke and water on uniform grids. Empowered by the semi-Lagrangian work of [112], [26] used vorticity confinement to simulate smoke with visually rich small scale rolling motions. Similarly using both semi-Lagrangian methods and hybridized particle and implicit surface techniques, [29, 21] simulated splashing water with both smooth surfaces and thin sheets. While these methods have achieved great success, their application is limited by the computational hardware (i.e. CPU, RAM, disk space) required for the simulations. In an attempt to alleviate this, [98] proposed a method that combines interpolation and two-dimensional simulation to obtain
Figure 2.1: Simulation of smoke past a sphere. The bottom two figures are close up views. The effective resolution is $1024^3$ and the computational time is about 4-5 minutes per frame.
highly detailed simulations of large scale smoke-like phenomena. While stunning results were obtained, this method does not faithfully reproduce the three-dimensional Navier-Stokes equations and thus is unable to obtain results for various fully three-dimensional phenomena. Moreover, water was not addressed.

In order to optimize the use of computational resources, we use an adaptive mesh or a level of detail approach where more grid cells are placed in visually interesting regions with rolling smoke or sheeting water. Although adaptive mesh strategies for incompressible flow are quite common, see e.g. [43], implementations based on recursive structures, such as the octrees we propose here, are less common. In fact, [95] claims to have the first octree implementation of incompressible flow, although there are certainly similar works such as the nested dyadic grids used for parabolic equations in [100]. We extend the work of [95] in two ways. First, we extend octrees to free surface flows allowing the modeling of a liquid interface. Second, we consider unrestricted octrees whereas [95]'s octrees were restricted.

Adaptive meshing strategies lead to nonuniform stencils and thus a nonsymmetric system of linear equations when solving for the pressure, which is needed to enforce the divergence free condition. Although [95] solved this nonsymmetric linear system with a multilevel Poisson solver, [17] pointed out that these multigrid approaches can be problematic in the presence of objects with high frequency detail. Moreover, the situation worsens in the presence of interfaces (such as that between water and air), especially since the faithful coarse mesh representation of watertight isosurfaces is a difficult research problem in itself, see e.g. [63]. Although multigrid solvers can be efficiently applied if the density is smeared out across the interface as in [120] (resembling a one-phase variable density flow as in [1]), this damps out the surface wave generation that relies on horizontal pressure differences caused by stacking different heights of high-density fluid. That is, damping these high frequency pressure differentials makes multigrid efficient, but also damps the wave motion leading to visually uninteresting overly viscous flows.

More recently, [119] departed from a smeared out density approach and instead solves a free surface problem as in [21]. Moreover, [119] switches from multigrid to a preconditioned conjugate gradient (PCG) method stating that the pressure can be
robustly solved for with PCG since the matrix is symmetric. However, the symmetry requirement limits his work to uniform non-adaptive grids. Our new formulation alleviates this restriction by providing a symmetric positive definite discretization of the Poisson equation on an unrestricted octree data structure allowing fast solvers such as PCG to be applied, even in the presence of interfaces.

2.2 Previous Work

[57] solved a linearized form of the three dimensional Navier-Stokes equations, and [13] solved the two dimensional Navier-Stokes equations using the pressure to define a height field. The full three dimensional Navier-Stokes equations were solved in [30, 31, 32] for both water and smoke. Large strides in efficiency were made when [112] introduced the use of semi-Lagrangian numerical techniques, and [26] advocated using vorticity confinement in order to preserve the small scale structure of the flow. [29, 21] proposed hybridizing particle and level set methods for water. The incompressible form of the Navier-Stokes equations has been used and augmented to model fire [62, 88], clouds [78], particle explosions [28], variable viscosity [11], bubbles and surface tension [47], splash and foam [121], etc. [125] proposed a method for control and used it to make letters out of smoke, and [113] solved these equations on surfaces creating beautiful imagery. The compressible version of these equations were used to couple fracture to explosions in [131]. There are also other approaches such as SPH methods [96, 82].

The representation of implicit surfaces on octree data structures has a long history in the marching cubes community, see the recent papers of [55, 92] and the references therein. Moreover, [33, 93] popularized the use of signed distance functions on octree grids. In order to simulate water, we need to solve the partial differential equations that govern the motion of the signed distance function. [117] advocated using quadtrees and semi-Lagrangian methods to solve these equations. Reinitialization for maintaining the signed distance property was addressed in [116], and extrapolation of velocities was considered in [118]. One difficulty with semi-Lagrangian methods for solving level set equations is that extreme mass loss (and thus visual artifacts)
usually occurs, however [20] recently showed that the particles in the particle level
set method alleviate this difficulty. A quadtree structure for level set evolution was
also proposed in [110]. However, none of these authors considered level sets in the
context of incompressible flows with interfaces such as water.

Starting with the seminal works of [4, 3], adaptive mesh refinement (AMR) typ-
ically utilizes uniform overlapping Cartesian grids of various sizes. This is because
AMR originally focused on compressible flow with shock waves, and a block struc-
tured approach is better able to avoid spurious shock reflections from changing grid
levels (since there are less of them). However, in the absence of shocks, a more optimal
unrestricted octree approach can be used for incompressible flow.

2.3 The Octree Data Structure

Figure 2.2 illustrates our unrestricted octree data structure (see e.g. [103]) with a
standard MAC grid arrangement [44], except that all the scalars except the pressure
are stored on the nodes or corners of the cell. This is convenient since interpolations
are more difficult with cell centered data (see e.g. [116]).

Coarsening is performed from the smaller cells to the larger cells, i.e. from the
leaves to the root. When coarsening, nodal values are either deleted or unchanged,
and the new velocity components at the faces are computed by averaging the old
values from that face. Refinement is performed from the larger cells to the smaller
cells. The value of a new node on an edge is defined as the average of its two
neighbors, and the value of a new node at a face center is defined as the average
of the values on the four corners of that face. The velocities on the new faces are
defined by first computing the velocities at the nodes, and then averaging back to
the face centers. Nodal velocities are computed by averaging the four values from the
surrounding cell faces as long as the faces are all the same size. Otherwise, using the
coarsest neighboring face as the scale, we compute temporary coarsened velocities on
the other faces to be used in the averaging.

For all variables, we constrain T-junction nodes on edges to be linearly interpo-
lated from their neighbors on that edge. Similarly, T-junction nodes on faces are
2.4 Navier Stokes Equations on Octrees

We use the inviscid, incompressible Navier-Stokes equations for the conservation of mass and momentum

\[
\begin{align*}
\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} &= -\nabla p + \mathbf{f}, \\
\nabla \cdot \mathbf{u} &= 0,
\end{align*}
\]

(2.1) (2.2)

where \( \mathbf{u} = (u, v, w) \) is the velocity field, \( \mathbf{f} \) accounts for the external forces, and the spatially constant density of the mixture has been absorbed into the pressure \( p \).

Equation 5.1 is solved in two steps. First we compute an intermediate velocity \( \mathbf{u}^* \) ignoring the pressure term, and then we compute the velocity update via

\[
\mathbf{u} = \mathbf{u}^* - \Delta t \nabla p
\]

(2.3)
where the pressure is defined as the solution to the Poisson equation,

$$\nabla^2 p = \nabla \cdot \mathbf{u}^*/\Delta t. \quad (2.4)$$

The external forces are discretized at the cell faces and we postpone the details of their discretization to sections 2.5 and 2.6. The convective part of the velocity update is solved using a semi-Lagrangian stable fluids approach as in [112]. First we compute nodal velocities, and then we average these values to the cell faces (see section 2.3). The cell face values are used to trace back characteristics, and trilinear interpolation of nodal values is used to define the new intermediate value of the velocity component on the face in question.

### 2.4.1 The Divergence Operator

Equation 2.4 is solved by first evaluating the right hand side at every cell center in the domain. Then a linear system for the pressure is constructed and inverted. Consider the discretization of equation 2.4 for a large cell with dimensions $\Delta x$, $\Delta y$ and $\Delta z$ neighboring small cells as depicted in figure 2.2. Since the discretization is closely related to the second vector form of Green’s theorem that relates a volume integral to a surface integral, we first rescale equation 2.4 by the volume of the large cell to obtain $V_{\text{cell}} \Delta t \nabla^2 p = V_{\text{cell}} \nabla \cdot \mathbf{u}^*$. The right hand side now represents the quantity of mass flowing in and out of the large cell within a time step $\Delta t$ in $m^3 s^{-1}$. This can be further rewritten as

$$V_{\text{cell}} \nabla \cdot (\mathbf{u}^* - \Delta t \nabla p) = 0. \quad (2.5)$$

This equation implies that the $\nabla p$ term is most naturally evaluated at the same location as $\mathbf{u}^*$, namely at the cell faces, and that there is a direct correspondence between the components of $\nabla p$ and $\mathbf{u}^*$. Moreover, substituting equation 2.3 into equation 2.5 implies $V_{\text{cell}} \nabla \cdot \mathbf{u} = 0$ or $\nabla \cdot \mathbf{u} = 0$ as desired.
Invoking the second vector form of Green's theorem, one can write

\[ V_{\text{cell}} \nabla \cdot u^* = \sum_{\text{faces}} (u_{\text{face}}^* \cdot n) A_{\text{face}}, \]

where \( n \) is the outward unit normal of the large cell and \( A_{\text{face}} \) represents the area of a cell face. In the case of figure 2.2, the discretization of the \( x \) component \( \partial u^*/\partial x \) of the divergence reads \( \Delta x \Delta y \Delta z \partial u^*/\partial x = u_2^* A_2 + u_3^* A_3 + u_4^* A_4 + u_5^* A_5 - u_1^* A_1 \), where the minus sign in front of \( u_1^* A_1 \) accounts for the fact that the unit normal points to the left. Then \( \partial u^*/\partial x = ((u_2^* + u_3^* + u_4^* + u_5^*)/4 - u_1^*)/\Delta x \). The \( y \) and \( z \) directions are treated similarly.

Once the divergence is computed at the cell center, equation 2.4 is used to construct a linear system of equations for the pressure. Invoking again the second vector form of Green's theorem, one can write

\[ V_{\text{cell}} \nabla \cdot (\Delta t \nabla p) = \sum_{\text{faces}} ((\Delta t \nabla p)_{\text{face}} \cdot n) A_{\text{face}}. \] (2.6)

Therefore, once the pressure gradient is computed at every face, we can carry out the computation in a manner similar to that of the velocity divergence above. There exist different choices in the discretization of \( (\nabla p)_{\text{face}} \), and we seek to discretize the pressure gradient in a fashion that yields a symmetric linear system. Efficient iterative methods such as PCG (see e.g. [102]) can be applied to symmetric positive definite matrices offering a significant advantage over methods for nonsymmetric linear systems. Moreover, since data access for the octree is not as convenient as for regular grids, there is a strong benefit in designing a discretization that leads to a symmetric linear system.

### 2.4.2 The Pressure Gradient

Consider the configuration in figure 2.3. In the case where two cells of the same size juxtapose each other, standard central differencing defines the pressure gradient at the face between them, as is the case for \( p_y = (p_{10} - p_1)/\Delta y \).

Consider the discretization of the pressure gradient in the \( x \) direction at the face...
between cell 1 and cell 2. A standard approach is to first compute a weighted average value $p_a$ for the pressure, by interpolating between the pressure values $p_1$ and $p_{10}$. Then, since standard differencing of $\hat{p}_x = (p_2 - p_a)/(0.75\Delta x)$ does not define $\hat{p}_x$ at the cell face but midway between the locations of $p_a$ and $p_2$, one usually resorts to more complex discretizations. A typical choice is to pass a quadratic interpolant through $p_a$, $p_2$ and $p_6$ and evaluate its derivative at the cell face, see e.g. [14]. However, this approach yields a nonsymmetric linear system that is slow to invert. The nonsymmetric nature of the linear system comes from the non-locality of the discretization, i.e. $p_a$ depends on $p_{10}$ and the quadratic interpolation would depend on $p_6$. Consequently, the equation for cell 1 involves both $p_{10}$ and $p_6$. It is unlikely that the equation for cell 6 depends on $p_1$, since cell 6 juxtaposes another cell of the same size, namely cell 2. And even if it did, the coefficients of dependence would not be symmetric.

Our approach is based on the fact that $O(\Delta x)$ perturbations in the pressure location still yield consistent approximations as in [37]. Therefore defining $p_x = (p_2 - p_a)/(0.75\Delta x)$ at the cell face still yields a convergent approximation, since the location of $\hat{p}_x$ is perturbed by a small amount proportional to a grid cell. Moreover, we can avoid the dependence of $p_a$ on values other than $p_1$ by simply setting $p_a = p_1$. This corresponds to an $O(\Delta x)$ perturbation of the location of $p_1$, and therefore still yields a convergent approximation. Thus, our discretization of $p_x$ is simply $p_x = (p_2 - p_1)/(0.75\Delta x)$. Moreover, since only $p_1$ and $p_2$ are considered, one can define
Figure 2.4: An ellipsoid slips along through shallow water illustrating our method’s ability to resolve thin sheets. The effective resolution is $512^3$ and the computational time is about 4-5 minutes per frame.
\[ p_x = \frac{(p_2 - p_1)}{\triangle} \]

where \( \triangle \) can be defined as the size of the large cell, \( \triangle x \), the size of the small cell, \( .5\triangle x \), the Euclidean distance between \( p_1 \) and \( p_2 \), etc. We have carried out numerical tests against known analytic solutions to the Poisson equation demonstrating that all these choices converge. Currently, we are investigating the impact of different \( \triangle \) definitions on smoke and water simulations.

In light of equation 2.6, \( p_x \) contributes to both row 1 and row 2 of the matrix representing the linear system of equations, since it is located at the cell face between cell 1 and cell 2. More precisely, the contribution to row 1 occurs through the term

\[ \triangle t p_x n_1 A_{\text{face}} = \frac{\triangle t}{\triangle} \frac{p_2 - p_1}{\triangle}, \]

since \( n_1 \), the x component of the outward normal to cell 1, points to the right (hence \( n_1 = 1 \)). Likewise, the contribution to row 2 occurs through the term

\[ \triangle t p_x n_1 A_{\text{face}} = \frac{\triangle t}{\triangle} \frac{p_2 - p_1}{\triangle}(-1), \]

since \( n_1 \), the x component of the outward normal to cell 2, points to the left (hence \( n_1 = -1 \)). Therefore, the coefficient for \( p_2 \) in row 1 and the coefficient for \( p_1 \) for row 2 are identical, namely \( \frac{\triangle t A_{\text{face}}}{\triangle} \). The same procedure is applied to all faces, and the discretization of the y and z components of the pressure gradient are carried out in a similar manner. Hence, our discretization yields a symmetric linear system that can be efficiently inverted with a PCG method. The preconditioner we use is based on an incomplete \( LU \) Cholesky factorization that we modify to ensure that the row sum of \( LU \) is equal to the row sum of the original matrix (see [102]). This yields a significant speed up in the matrix inversion.

The matrix constructed above is negative definite, as is usual when discretizing equation 2.4. We simply multiply all equations by \(-1\) to make it positive definite. We also note that Dirichlet or Neumann boundary conditions do not disrupt the symmetry. In the case of a Neumann boundary condition, the term \( (p_2 - p_1)/\triangle \) disappears from both row 1 and row 2. In the case of a Dirichlet boundary condition, e.g. for \( p_2 \), the equation for \( p_2 \) drops out of the system and all the terms involving \( p_2 \)
are moved to the right hand side of the linear system.

2.4.3 Accuracy

We stress that the dominant errors are due to the first order accurate semi-Lagrangian advection scheme. The velocity averaging is second order accurate and is required in all MAC grid methods in order to define a full velocity vector at a common location for the semi-Lagrangian advection. Dropping the Poisson solver from second to first order accuracy merely puts it on par asymptotically with the semi-Lagrangian scheme. However, we still solve for a fully divergence free velocity field to machine precision just as in a non-adaptive setting. We tested our Poisson solver on many exact solutions and readily obtain several digits of accuracy indicating that the errors from this part of the algorithm are small. See table 2.1 for a typical result.

2.5 Smoke

The external forces due to buoyancy and heat convection are modeled as $f_{\text{buoy}} = -\alpha \rho z + \beta (T - T_{\text{amb}}) z$, where $z = (0, 0, 1)$, $T_{\text{amb}}$ is the ambient temperature and $\alpha$ and $\beta$ are parameters controlling the influence of the density and the temperature. The density and the temperature are passively advected with the flow velocity and are updated with the semi-Lagrangian method using velocities defined at the nodes (see section 2.3). Both the density and the temperature are then averaged to the faces in order to evaluate the forcing term.

The vorticity confinement force is calculated as follows. First we define velocities at the centers of cells by using area weighted averaging of face values. Then all the derivatives needed to compute the vorticity, $\omega = \nabla \times \mathbf{u}$, are computed on cell faces using the same method used to compute pressure derivatives. Area weighted averaging is used (again) to define all these derivatives at the cell center, and then we compute the vorticity and its magnitude (at the cell center). Next, the gradients of the vorticity are computed at the cell faces, and averaged back to the cell center to define $\mathbf{N} = \nabla |\omega|/|\nabla |\omega||$. Finally, the unscaled force can be computed at the cell
\begin{table}
\begin{tabular}{|c|c|c|c|}
\hline
$L^1$ error & order & $L^\infty$ error & order \\
\hline
$4.083 \times 10^{-2}$ & --- & $6.332 \times 10^{-2}$ & --- \\
$8.713 \times 10^{-3}$ & 2.22 & $2.203 \times 10^{-2}$ & 1.523 \\
$2.952 \times 10^{-3}$ & 1.56 & $1.292 \times 10^{-2}$ & .770 \\
$9.980 \times 10^{-4}$ & 1.56 & $7.745 \times 10^{-3}$ & .739 \\
$4.010 \times 10^{-4}$ & 1.31 & $4.249 \times 10^{-3}$ & .866 \\
$1.820 \times 10^{-4}$ & 1.14 & $2.287 \times 10^{-3}$ & .894 \\
\hline
\end{tabular}
\caption{Poisson solver accuracy on an unrestricted octree grid.}
\end{table}

centers as $N \times \omega$. Cell face values of this term are obtained by averaging the values from the two cells that contain the face. Then this term is scaled by the diagonal of the face $h$ and a tunable parameter $\epsilon$.

Inside an object, we set the temperature to the object temperature and the density to zero. For velocity, we clip the component normal to the object so that it is guaranteed to be separating. Furthermore, we apply Neumann boundary conditions to the cell faces that intersect the object when solving for the pressure. This keeps these velocities fixed.

In the case of smoke, we utilize three different refinement criteria. First, we refine near objects since their interactions with smoke will introduce small scales features that enhance believability. Second, we refine near concentrations of high vorticity. Third, we refine in a band of density values (for example $0.1 < \rho < 0.3$). This last criteria prunes out both the low densities that cannot be seen as well as the high densities interior to the smoke which are self-occluded.

\section{Water}

We use the particle level set method of \cite{19} with $\phi \leq 0$ designating the water and $\phi > 0$ representing the air. When solving for the pressure, one only needs to consider cells in the water. Dirichlet boundary conditions of $p_l = p_{\text{air}} + \sigma \kappa$ are set in the air cells bordering the water, where $\sigma$ is a surface tension coefficient and $\kappa = \nabla \cdot (\nabla \phi / |\nabla \phi|)$ is the local interface curvature. We note that \cite{47} considered surface tension in the case
of bubbles, but not for films. \( \kappa \) is computed by averaging nodal values of \( \phi \) to the cell center, computing derivatives of \( \phi \) on the cell faces, averaging these back to the cell center, using these cell centered values to obtain the normal, computing derivatives of the normal on the cell faces, averaging these values back to the cell center, and finally using these cell centered values to obtain the curvature. The only external force we account for is gravity via \( \mathbf{u} = \Delta t \mathbf{g} \). The interaction with objects is similar to that of smoke. We apply adaptive refinement to a band about the interface (focusing more heavily on the water side), noting that the signed distance property of \( \phi \) makes this straightforward.

Recently, [20] showed that the particle level set method relies on particles for accuracy and the level set for connectivity. Moreover, they showed that one could use a simple semi-Lagrangian method on the level set with no significant accuracy penalty as long as the particles are evolved with at least second order Runge-Kutta. Thus, we update \( \phi \) with the semi-Lagrangian method using velocities defined at the nodes (see section 2.3). The particles are advected using second order Runge-Kutta and trilinearly interpolated nodal velocities.

We use the fast marching method [126, 107] to maintain the signed distance property of \( \phi \). First, the signed distance is computed at all the nodes around the interface, and they are marked as updated. The nodes adjacent to the updated nodes are tagged trial. Then we compute potential values of \( \phi \) at all trial nodes using only updated nodes. The smallest of these is tagged as updated, and all its non-updated neighbors are tagged trial. This process is repeated to fill in a band of values near the interface. For many grid nodes there are neighboring values of \( \phi \) in all six directions, but at T-junctions there are directions where \( \phi \) is missing some of its neighbors. Since we coarsen as we move away from the interface, these directions will generally not contribute to the potential value of \( \phi \). Thus, we trivially ignore them.

Velocity extrapolation is carried out by first defining nodal velocities, extrapolating them, and then computing the face velocities. If we perform this algorithm as in [19], we will occasionally encounter grid points that have no neighboring values of velocity and cannot be updated. While this is rare (but not impossible) for uniform grids, T-junctions exacerbate their occurrence on octree grids. When this happens,
we skip over these nodes until one of their neighbors is updated (and then we update them in the usual manner).

2.7 Second order Accuracy on Octrees

A potential difficulty with the method proposed above is that it computes a different pressure gradient on each cell face, and also allows a different velocity on each face. The velocity on a face of a large cell is computed as the area weighted average of the velocities on the faces of all the adjacent small cells. Alternatively, in order to properly constrain the velocity field for interpolation, etc., it is desirable to have the velocity on the large cell face identical to all the velocities on the adjacent small cell faces. Area weighted averaging can still be used to compute the velocity on the large cell face, but this velocity then needs to be assigned to all the small cell faces as well. This also simplifies the treatment of Neumann (fixed velocity) boundary conditions, since an entire large cell face can be constrained as opposed to only portions of it.

If all the small cell faces incident on a large cell face have the same velocity, it is also important that they have the same pressure gradient when enforcing incompressibility so that they maintain the same velocity after being projected to be divergence free.
Unfortunately, the method proposed above yields different pressure gradients for these incident small cells. The pressure gradient on the large cell face is given by the area weighted average of the pressure gradients computed on each of the small cell faces, i.e.

\[
(p_x)_L = \sum_s \frac{A_s}{A_L} \left( \frac{p_s - p_L}{\Delta_s} \right)
\]

where \((p_x)_L\) is the pressure gradient on the large cell face, \(A_L\) is the area of the large cell face, \(p_L\) is the pressure in the large cell, the sum is over all incident small cell faces, \(A_s\) is the area of a small cell face, \(p_s\) is the pressure in a small cell, and \(\Delta_s\) is the distance associated with a small cell face pressure gradient (e.g. half the size of the large cell plus the size of the associated small cell). To compute the pressure gradient flux into the large cell, \((p_x)_L\) is multiplied by \(A_L\). For an incident small cell, the first order accurate method above uses

\[
(p_x)_s = \frac{p_s - p_L}{\Delta_s}
\]

as the pressure gradient and computes the flux through the small cell as \(A_s(p_x)_s\). Instead, we propose using the pressure gradient of the large cell for all the small cells as well, i.e. setting \((p_x)_s = (p_x)_L\) so that the velocities of the small and large cells agree after the velocity field is projected to be divergence free.

For symmetry, first consider the coupling between the large cell and an incident small cell. When discretizing the large cell with \(A_L(p_x)_L\), the coefficient of a small cell pressure is \(A_s/\Delta_s\). When discretizing a particular small cell with \(A_{s_o}(p_x)_L\), the coefficient of the large cell pressure is \(-A_{s_o}/A_L \sum_s (-A_s/\Delta_s)\) where the extra minus sign comes from the fact that the normal points to the left (assuming the small cells are to the right of the large cell). If we use the same \(\Delta_s\) for all incident small cells, e.g. we use an area weighted average

\[
\Delta = \sum_s \frac{A_s}{A_L} \Delta_s
\]
then $\triangle s$ factors out of the sum, $\sum_s A_s$ is equal to and cancels out $A_L$, and we arrive at $A_s/\triangle$ yielding symmetry for the coupling between the large and small cells. We must also consider symmetry between pairs of small cells incident on the same large cell face, as this new scheme couples them together as well. When discretizing a particular small cell with $A_s(p_x)_L$, the coefficient of another small cell pressure is $-A_sA_s/(A_L\triangle)$. Similarly, the contribution cell $s_o$ makes to another small cell via $A_s(p_x)_L$ is $-A_sA_s/(A_L\triangle)$ so the coupling is symmetric.

Although the discretization proposed in [67] is first order accurate, preliminary tests with this new discretization seem to indicate that it is second order accurate. For example, consider $\nabla^2 p = e^x + e^y$ with an exact solution of $p = e^x + e^y$. We use the initial grid shown in Figure 2.6, and uniformly refine a number of times to obtain the results shown in Table 2.2. Obviously, the discretization is second order accurate in both the average and max norms. The method is second order accurate in three spatial dimensions as well, as illustrated in table 2.3 which demonstrates this for $\nabla^2 p = e^x + e^y + e^z$ with an exact solution of $p = e^x + e^y + e^z$.

Although we were at first surprised by the second order accurate numerical results,
CHAPTER 2. FLUIDS ON AN OCTREE DATA STRUCTURE

Table 2.2: Poisson solver accuracy on a quadtree grid.

<table>
<thead>
<tr>
<th>( L^1 ) error</th>
<th>order</th>
<th>( L^\infty ) error</th>
<th>order</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 4.465 \times 10^{-3} )</td>
<td>( ____ )</td>
<td>( 5.012 \times 10^{-3} )</td>
<td>( ____ )</td>
</tr>
<tr>
<td>( 6.720 \times 10^{-4} )</td>
<td>2.73</td>
<td>( 9.777 \times 10^{-4} )</td>
<td>2.36</td>
</tr>
<tr>
<td>( 1.200 \times 10^{-4} )</td>
<td>2.49</td>
<td>( 2.050 \times 10^{-4} )</td>
<td>2.25</td>
</tr>
<tr>
<td>( 2.447 \times 10^{-5} )</td>
<td>2.29</td>
<td>( 4.626 \times 10^{-5} )</td>
<td>2.15</td>
</tr>
<tr>
<td>( 5.466 \times 10^{-6} )</td>
<td>2.16</td>
<td>( 1.092 \times 10^{-5} )</td>
<td>2.08</td>
</tr>
<tr>
<td>( 1.289 \times 10^{-6} )</td>
<td>2.09</td>
<td>( 2.648 \times 10^{-6} )</td>
<td>2.04</td>
</tr>
</tbody>
</table>

Table 2.3: Poisson solver accuracy on an octree grid.

<table>
<thead>
<tr>
<th>( L^1 ) error</th>
<th>order</th>
<th>( L^\infty ) error</th>
<th>order</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 3.241 \times 10^{-3} )</td>
<td>( ____ )</td>
<td>( 6.990 \times 10^{-3} )</td>
<td>( ____ )</td>
</tr>
<tr>
<td>( 7.219 \times 10^{-4} )</td>
<td>2.17</td>
<td>( 1.596 \times 10^{-3} )</td>
<td>2.13</td>
</tr>
<tr>
<td>( 1.700 \times 10^{-4} )</td>
<td>2.08</td>
<td>( 3.841 \times 10^{-4} )</td>
<td>2.06</td>
</tr>
<tr>
<td>( 4.121 \times 10^{-5} )</td>
<td>2.04</td>
<td>( 9.460 \times 10^{-5} )</td>
<td>2.02</td>
</tr>
<tr>
<td>( 1.014 \times 10^{-5} )</td>
<td>2.02</td>
<td>( 2.354 \times 10^{-5} )</td>
<td>2.01</td>
</tr>
</tbody>
</table>

[64] and the references therein indicate the plausibility of this. In fact, it was [64] (and personal communications with the authors) that directly led to our first attempts to define unique pressure gradients on cell faces that could be used for both the large cell and all the small cells incident on a particular face.

Finally, we point out to the reader that uniformly refining an octree may not be the best way to numerically check accuracy. Repeated refinement of an octree eventually yields an octree where every large cell has face neighbors with uniform size, although the ratio between the large and small cells is no better or worse than that in the initial unrefined octree, i.e. it does not become graded. To see this trivially, consider a large cell in the initial octree mesh next to any number of smaller cells. Once enough uniform refinements occur to subdivide the large cell into smaller cells at most the size of its smallest original neighbor, then all neighbors incident on a given face of a subdivided child of the large cell are trivially equal in size.
2.8 Adaptive Solver Discussion

Our new symmetric formulation reduces the pressure solver to approximately 25% of the total simulation time requiring only about 20 iterations to converge to an accuracy of machine precision. This leaves little room for improvement and even a zero cost pressure solver would only make the code 25% faster. On the other hand, nonsymmetric formulations requiring BiCGSTAB or GMRES and nonoptimal preconditioners easily lead to an order of magnitude slowdown, or in the worst case scenario problems with robustly finding a solution at all. The symmetric formulation enables a full octree discretization of the equations that govern both the flow of smoke and water. Moreover, we achieved reasonable computational costs on grids as fine as $1024^3$ allowing us to capture fine scale rolling motion in smoke and thin films for water.
Chapter 3

Multiple Interacting Liquids

3.1 Introduction

Earlier works on fluid simulation focused on single phase flows such as smoke [32, 112, 26] or free surface flows such as water [29, 21]. More recently, researchers have considered more complex phenomena including fire [62, 88], bubbles [48], viscoelasticity [39], etc. Although the level set method allows for the simulation of two distinct fluids, such as fuel and products [88] or water and air [48], it does not handle the complex phenomena associated with the interactions of more than two fluids. In this chapter, we propose a novel method that allows one to simulate multiple (more than two) liquids (and gases), including complex interactions between the different fluids.

A few researchers have begun to tackle the difficulties associated with using multiple level sets (without particles) to represent multiple regions. One approach is to use a different level set for each region as in [75, 101, 109]. Each level set is independently evolved forward in time leading to contradictions in the representation of the interface, which are resolved via projection (or slowly, via a penalty method as in [134]) eliminating points that have been classified as inside more than one region or not inside any region. The other main approach uses $n$ level sets to represent up to $2^n$ regions [127]. For example, two level sets can be used to represent four regions classified via all possible sign combinations (i.e. “++”, “+-”, “-+” and “--”). This approach intrinsically removes the need for projection, but typically
suffers from biasing artifacts especially where more than two regions intersect. So while it is quite useful in computer vision especially when there are many regions, it has not enjoyed similar success in physics-based applications where the number of distinct materials is typically small enough that it is not inefficient to use a separate level set for each allowing for a non-biased simulation of the underlying physics. In fact, level set methods are typically only applied in a lower dimensional band near the interface making them cheap as compared to the physical equations that need to be solved everywhere. Moreover, regardless of the number of level sets used, the particle level set method still requires one set of particles for each region, so its cost remains unchanged. Finally, we note that all these approaches are predated by [34], which proposed a method for removing overlaps of implicitly represented deformable objects, although gaps between objects were not addressed making the work inapplicable to fluids where vacuum regions need to be properly addressed.

At each point in the domain, we have a vector $\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \ldots, \phi_n(\vec{x}))$. Since the individual level set functions (the $\phi_i$’s) will generally give contradictory geometric information, a consistent approach to interpreting the vector-valued level set function is needed. We do this by creating a level set “dictionary” that translates between the vector $\vec{\phi}$ and the traditional single level set representation at each point in the domain. Our novel projection method makes this translation straightforward for both theoretical and practical purposes (e.g. allowing the incorporation of previous level set simulation techniques). Moreover, our method is purely geometric and thus does not interfere with the underlying physics. It also preserves the signed distance property of the various level set functions (unlike for example [75]).

Our method provides for the straightforward simulation of multiple liquids (and air) with varying densities, viscosities, or viscoelastic properties. We also consider complex interactions between fluids such as surface tension forces and reactions (e.g. the burning of a premixed fuel as in [88]). Such interactions typically involve discontinuous material properties across the interface, e.g. pressure jumps due to surface tension. [88] and [48] advocated using the ghost fluid method (GFM) to avoid the visual errors associated with nonphysically smearing out these discontinuities. We propose a novel paradigm that automatically detects when discontinuous information
is combined across (any number of) interfaces, computes jump conditions and ghost values “on the fly,” and returns appropriate values. This reduces the memory requirements associated with storing ghost values for multiple region interactions, and furthermore makes the implementation of the algorithms straightforward. We note that it also simplifies the treatment of complex solid objects (see e.g. [42]).

3.2 Previous Work

Besides the works already mentioned above, earlier computer graphics research on the Navier-Stokes equations includes [57, 13, 30, 31]. There has also been work on explosions [87, 131, 28, 98], flow on surfaces [113], chemically reacting gases [50], octree implementations [67], RLE implementations [49], tetrahedral meshes [27] hybridized vortex particle approaches [106] and sand [136]. Various authors have also addressed viscosity [11, 97, 48], surface tension [22, 47, 67, 48] and fire [114, 62, 88, 74]. Some of the most recent interesting areas include control [125, 72, 24, 97, 76, 108], solid fluid coupling [12, 42, 128, 68], and SPH [96, 58]. In fact, [85] tackles the problem of interacting multiple fluids from the SPH standpoint.

3.3 Multiple Level Sets

Each level set function is independently evolved in time, after which the interface locations do not agree (because of numerical errors) as shown for example in Figure 3.1. We propose a novel method for fixing the level set functions removing overlaps and vacuums while preserving an accurate interface location.

3.3.1 Projection Method

We first make the following observations about an arbitrary vector \( \vec{\phi} \) of level set values at a point \( \vec{x} \). (O1) If \( \phi_j \) is the smallest element, \( \vec{x} \) is in region \( j \). This assigns \( \vec{x} \) to the region it is deepest inside when it is inside more than one region (overlap), or the region it is closest to when it is outside every region (vacuum). (O2) If O1 holds and
\( \phi_k \) is the second smallest element, only \( \phi_j \) and \( \phi_k \) are needed to locally represent the interface. Basically, \( \vec{x} \) is in region \( j \), and the closest point on an interface lies between region \( j \) and region \( k \). Region \( k \) is the region \( \vec{x} \) is closest to not counting the region it is in.

Given these observations, we desire the following properties for numerical robustness and backward compatibility with the standard single level set function for two phases. (P1) If \( \phi_j \) is the smallest element, it is the only negative element and its magnitude represents the distance to the interface. This is consistent with observation 1, but also makes \( \phi_i \) a signed distance function in region \( i \) for all \( i \). Moreover, it removes overlaps since only one \( \phi_i \) is negative, and removes vacuums since the smallest \( \phi_i \) is negative. (P2) If P1 holds and \( \phi_k \) is the second smallest element, \( \phi_k = -\phi_j \). This is consistent with observation 2, and it makes the level set for the region a point is closest to but not inside a signed distance function as well. Moreover, all other \( \phi_i \) are positive and bigger than \( \phi_k \) and not relevant.

These observations and properties are consistent with the standard single level set function methodology. A standard single level set function \( \phi \) can be broken into two separate functions \( \phi_1 = \phi \) and \( \phi_2 = -\phi \), and be shown to satisfy the above observations and properties. This readily gives us a dictionary that translates between \( \vec{\phi} \) and \( \phi \). That is, once we take an arbitrary level set vector \( \vec{\phi} \) and project it to satisfy properties 1 and 2, we can use \( \phi_j \) and \( \phi_k \) as if they were \( \phi_1 \) and \( \phi_2 \) in the appropriate order. The only discrepancy lies in how the exact interface is handled where \( \phi_j = \phi_k = 0 \). For the standard level set function, this is equivalent to \( \phi_1 = \phi_2 = \phi = 0 \) and is typically nominally assigned to the negative level set, i.e. we define the regions via \( \phi \leq 0 \) and \( \phi > 0 \). This is equivalent to assigning the point to region 1 when \( \phi_1 \) and \( \phi_2 \) are both zero. To extend this to multiple level sets, we assign a point where both \( \phi_j \) and \( \phi_k \) are zero to region \( j \) or \( k \) depending on whether \( j < k \).

We illustrate our method in one spatial dimension. Figure 3.3a shows two level set functions that satisfy properties 1 and 2, and Figure 3.3b shows a property violating version after evolving in time. Based on property 1, the interface location is defined as the point where the minimum \( \phi_i \) changes from one level set to the other. This is
the location where the two level sets intersect in the figure, and we want our projection method to preserve the interface location to avoid biasing. Thus, our projection method computes the average value of the two level set functions and subtracts this average from both of them. At points where the two level sets intersect, their average equals their individual values, and thus subtracting off their averages sets them both to zero preserving the interface location. Otherwise, at points where one level set is larger than the other, subtracting their average makes them the same magnitude but opposite sign preserving the region a given point is inside. The result is shown as dotted lines in the figure. If both level sets are reinitialized to signed distance functions, we obtain the result shown in Figure 3.3a which satisfies all desired properties. The same projection method can be generalized to an arbitrary vector of level sets by subtracting the average of the smallest two $\phi_i$ from all of the $\phi_i$. An example of this is shown in Figure 3.4.

Notably our method is unbiased preserving signed distance information. For example, consider Figure 3.3a and Figure 3.4b where the level sets are all signed distance functions to begin with. Because property 2 holds, the two smallest $\phi_i$ are equal and opposite in sign making their average identically zero at every point. Thus, subtracting the average leaves all the the $\phi_i$ unchanged preserving signed distance. This surprisingly simple algorithm has all the properties we desire. Notably, [75] is similar in spirit to our own, but while they preserve the interface location and inside/outside information, they do not preserve signed distance thus introducing biasing into the algorithm.

For two regions (hence two level sets) the result produced by our method is identical to that of the traditional single level set (or particle level set) method. If the two level set functions at time $n$ are negatives of each other, i.e. $\phi_1^n = -\phi_2^n$, after advection we still have $\phi_1^* = -\phi_2^*$ because they are evolved with the same method. Then projection leaves $\phi_1^*$ and $\phi_2^*$ unchanged, since their identically zero average is subtracted off. This is also true for more than two regions away from multiple junctions (e.g., triple points).
3.3.2 Particle Level Set Method

Each level set has an associated set of particles that are seeded near the boundary of its interior region as shown in the figure to the right. Following the standard particle level set algorithm, for each level set function we rebuild $\phi^-$ using that level set’s particles, and rebuild $\phi^+$ using all the particles from all other regions. For efficiency, we ignore particles that are far from the interface of the region in question. Typically, particles are used to correct the level set function both after advection and after reinitialization. We apply our projection method to every grid point after each of these particle correction steps. Then for each grid point, all geometric information can be computed from the level set function that is negative at that point. When level set values are needed in between grid points, we interpolate $\vec{\phi}$ to that location and apply our projection method on the fly to find the resulting negative $\phi_i$. Note that the first projection step is important because reinitialization preserves the interface location of each level set individually, but not their intersections which correspond to the pre-projected interface locations. The second projection removes any numerical drift introduced by reinitialization, and we note that a method such as [75] could not be used for this step because it does not preserve signed distance.

Advancing the incompressible velocity field to the next time step and particle advection are the most expensive parts of our algorithm. These steps depend mostly on the fluid volume and surface area, respectively, rather than the number of regions. The cost of advecting and reinitializing the level set functions remains fixed at twice the usual cost, since two level sets are updated locally near each interface instead of one. Thus, the cost of updating the level set function scales with the surface area just as in the standard single level set method regardless of the number of regions.
CHAPTER 3. MULTIPLE INTERACTING LIQUIDS

3.4 Multiple Liquids

We model the fluids using the incompressible Navier-Stokes equations

\[ \nabla \cdot \bar{\mathbf{u}} = 0 \] (3.1)
\[ \bar{\mathbf{u}}_t + (\bar{\mathbf{u}} \cdot \nabla)\bar{\mathbf{u}} + \nabla p/\rho = (\nabla \cdot \mathbf{\tau})/\rho + \mathbf{f} \] (3.2)

where \( \bar{\mathbf{u}} = (u, v, w) \) is the velocity, \( \rho \) is the density, \( \mathbf{\tau} \) is the viscous stress tensor, and \( \mathbf{f} \) accounts for body forces, e.g. gravity, vorticity confinement, etc. For simplicity, we first consider the inviscid case. First, an intermediate velocity field \( \bar{\mathbf{u}}^* \) is computed

\[ \frac{(\bar{\mathbf{u}}^* - \bar{\mathbf{u}}^n)}{\Delta t} + (\bar{\mathbf{u}}^n \cdot \nabla)\bar{\mathbf{u}}^n = \mathbf{f} \] (3.3)

using a semi-Lagrangian advection scheme as in \([112]\). Next, we compute the pressure via

\[ \nabla \cdot (\nabla p/\rho) = \nabla \cdot \bar{\mathbf{u}}^*/\Delta t. \] (3.4)

and use it to make the velocity field divergence free

\[ \frac{(\bar{\mathbf{u}}^{n+1} - \bar{\mathbf{u}}^*)}{\Delta t} + \nabla p/\rho = 0. \] (3.5)

3.4.1 Poisson Equation

We follow the method of \([88]\). For multiple fluid regions, equation (5.1) is a Poisson equation with discontinuous coefficients. The equation is separable so we can consider each dimension independently. A standard second order accurate discretization of the left hand side in one spatial dimension at a grid node \( i \) is

\[ \left( \beta_{i+1/2}(p_{i+1} - p_i)/\Delta x - \beta_{i-1/2}(p_i - p_{i-1})/\Delta x \right)/\Delta x \] (3.6)
where $\beta = 1/\rho$. For inviscid flow, [56] showed that the flux in equation (5.1) is continuous across the interface satisfying

$$\beta^- p_x^- = \beta^+ p_x^+ \quad (3.7)$$

where the $-$ and $+$ superscripts represent values from different sides of the interface. Thus if $\rho$ (and hence $\beta$) varies across the interface then so must $p_x$. Consider the case where an interface lies between nodes $x_i$ and $x_{i+1}$. We define $\theta = |\phi(x_i)|/(|\phi(x_i)| + |\phi(x_{i+1})|)$ and approximate equation (3.7) with one-sided differences as

$$\beta^- (p_I - p_i)/(\theta \Delta x) = \beta^+ (p_{i+1} - p_I)/((1 - \theta)\Delta x) \quad (3.8)$$

and solve for the interface pressure $p_I = (\theta \beta^+ p_{i+1} + (1 - \theta)\beta^- p_i)/((\theta \beta^+ + (1 - \theta)\beta^-)$ which can be substituted into either the left or the right hand side of equation (3.8) to obtain $\hat{\beta}(p_{i+1} - p_i)/\Delta x$ where $\hat{\beta} = (\beta^- \beta^+)/(\theta \beta^+ + (1 - \theta)\beta^-)$. Thus, the discontinuity between grid nodes $i$ and $i+1$ is readily handled by replacing $\beta_{i+1/2}$ with $\hat{\beta}$ in equation (3.6). $\beta_{i-1/2}$ is treated similarly.

### 3.4.2 Viscosity

The viscous stress tensor for incompressible flow is $\tau = \mu(\nabla \bar{u} + (\nabla \bar{u})^T)$. As discussed in [97], a spatially constant $\mu$ (within each region) implies that $\nabla \cdot \tau = \mu \Delta \bar{u}$. Renaming $\bar{u}^{n+1}$ in equation (5.2) to be $\bar{u}^{**}$, we next solve the three systems of linear equations given by

$$\bar{u}^{***} = \bar{u}^{**} + \Delta t \nabla \cdot (\nu \nabla \bar{u}^{***}) \quad (3.9)$$

where $\nu = \mu/\rho$. Note that we moved $\rho$ under the divergence operator, under the assumption that it is spatially constant in each region. Since the viscosity to density ratio is discontinuous across the interface, we replace $\nu$ with $\hat{\nu}$ for differences that cross the interface in the same manner as $\beta$ is adjusted to $\hat{\beta}$ when solving equation (5.1). Then, we again solve for the pressure and make the flow divergence free using $\bar{u}^{***}$ in place of $\bar{u}^*$ in equations (5.1) and (5.2).

Each component of equation (3.9) should conserve momentum, so we require a
unique flux between every two velocity values. The physically correct flux in the incompressible flow context is rather complicated (see its derivation in [56]). However, considering equation (3.9) in isolation admits a simple approximation which [48] showed was sufficient for visual accuracy. In isolation, the flux is given by $\nu \nabla \vec{u}$, and we assume $\nu^- \nabla \vec{u}^- = \nu^+ \nabla \vec{u}^+$ which is not actually true. Instead, correction terms should be added for stencils that cross the interface, but these terms couple the $u$, $v$, and $w$ diffusion equations together making them difficult to solve with a fully implicit method. For example, see [97] where variable viscosity couples the three equations together. They explicitly add correction terms before solving for the velocity implicitly. We could take a similar approach allowing for a larger time step than a fully explicit method, but it is still less efficient than a fully implicit method. [56] showed that the viscosity jump causes a jump in the pressure and its derivatives as well, but [48] showed that these too can be ignored for graphical purposes.

### 3.4.3 Viscoelasticity

[39] incorporated viscoelastic effects by adding $(\mu_e/\rho) \nabla \cdot \epsilon$ to the Navier-Stokes equations, where $\mu_e$ is the elastic modulus and $\epsilon$ is the elastic strain tensor evolved in time via $\epsilon_t + \vec{u} \cdot \nabla \epsilon = (\nabla \vec{u} + (\nabla \vec{u})^T)/2 - \epsilon^{Plastic}$. They solved this last equation by first using semi-Lagrangian advection, and then incorporating the right hand side which is the total strain rate minus the plastic strain rate. [51] points out that this ignores the rotation of the strain tensors, yielding incorrect results when the fluid rotates. Thus, [51] proposes rotating the strain tensor by the curl of the velocity field after the advection step. This is accomplished by computing an explicit rotation matrix in the center of each cell, and using it to rotate the strain tensor also stored in the center of each cell. This has enabled high quality detailed viscoelastic fluid computations, see e.g. Figure 3.12.
3.5 Adding Air or Empty Regions

Often times, only the liquid region is of interest and the gas flow can be ignored. Our system allows for the standard treatment of this by setting an entire region to be empty, and subsequently extrapolating velocity into that region from the liquid regions and using Dirichlet boundary conditions during the pressure solve (see e.g. [21]). However, when the gas flow is important, our method trivially extends to simulate gas regions just as though they were other liquid regions. This allows for straightforward incorporation of smoke, fire, and even reactive gases as in [50]. Besides empty regions and air regions, there is yet a third way to model non-liquid regions. The animator may wish to simulate air, but not have the air affect the liquid. This requires one way coupling from liquid to air, but not vice versa. Figure 3.8 shows one way coupling (left) as compared to two way coupling (right). One way coupling is accomplished by using extrapolated velocities from the liquid to the gas as boundary conditions for the liquid region, while gas advection is carried out normally. In addition, these extrapolated velocities are used to overwrite the gas velocity at any grid points that become liquid as the interface moves. The Poisson equation is first solved for the liquid region setting Dirichlet boundary conditions in the gas so that it has no effect (as is usual for empty regions), and then a second Poisson equation is solved for the gas using Neumann fixed velocity boundary conditions in the liquid so that it properly drives the gas.

In free surface flow, the air region is modeled as empty allowing it to vanish. Thus, characteristics coalesce as shown in the single grid cell in the figure to the right. Eventually, liquid rushes into the cell from all sides, and the air should disappear. However, air particles faithfully follow these characteristics ending up trapped in the center of the cell. Since this cell should be a sink for air, we simply delete the air particles as they approach the center of the sink. Note that these sinks are easily detected by finding local minima level set values in empty regions.
3.6 Surface Tension

The ghost fluid method (GFM) of [25] uses the physically correct interfacial jump conditions to define ghost values for discontinuous quantities which are then incorporated into finite difference or interpolation stencils. [48] used the GFM to discretize the jumps in pressure caused by surface tension effects, and [46] showed that the method produces far better visual results than a smeared out delta function approach. Surface tension causes a jump in pressure across the interface equal to $\sigma \kappa$, where $\sigma$ is a surface tension coefficient (defined pairwise for the regions) and $\kappa = -\nabla \cdot (\nabla \phi / |\nabla \phi|)$ is the interface curvature. Consider the case where an interface lies between $x_i$ and $x_{i+1}$. We adjust $p_{i+1}$ in equation (3.6) to account for the jump in pressure,

$$\left( \hat{\beta}((p_{i+1} + \sigma \kappa_{\Gamma}) - p_i)/\Delta x - \beta_{i-1/2}(p_i - p_{i-1})/\Delta x \right)/\Delta x$$

where $\kappa_{\Gamma} = \theta \kappa_{i+1} + (1 - \theta) \kappa_i$ is computed with respect to the region containing $x_i$, and $\beta_{i+1/2}$ has been replaced by $\hat{\beta}$ as explained in section 3.4.1. The $(\hat{\beta} \sigma \kappa_{\Gamma})/\Delta x^2$ term can be moved to the right hand side, so that the resulting matrix is unaffected allowing for the use of fast symmetric linear system solvers such as the preconditioned conjugate gradient method.

3.7 Surface Reactions

Our method incorporates surface reactions allowing one material to turn into another. One example of this is the work on fire by [88], but our framework allows for a more generalized treatment, e.g. Figure 3.10 shows two materials (submerged beneath a third) coming together and reacting to form a fourth.

[88] used the GFM to model fire where the expansion of fuel into products admits jumps in both velocity and pressure.\(^5\) As in the surface tension case, the pressure jump is incorporated directly into the Poisson equation. The velocity jump needs to be handled whenever information is combined from different sides of the interface.

\(^4\)page 36  
\(^5\)See equations (2) and (3) for the jump conditions
They implement this by storing two velocity fields, one for fuel and one for products, that inherently store the jump conditions. While this only doubles their storage, the storage requirements would scale linearly with the number of different materials. We instead compute ghost values on the fly by generalizing the concept of a scalar or vector field to encapsulate the application of the jump condition. Our implementation wraps the data in a lookup class that maintains a state variable indicating the region for which values are being looked up. For example, when using semi-Lagrangian advection to update a face velocity in region $i$, we create a lookup class instance and set its internal state to indicate that any queried values should be returned with respect to region $i$. Then the lookup class ensures that any data used to construct the ray or interpolate is retrieved with the proper jump conditions already applied. In this manner, existing interpolation and discretization code is generalized with relative ease to account for discontinuities. The lookup classes can also be used to incorporate object intersections in the same manner as jump discontinuities. For example, for thin objects, a nested lookup class can be used to check for object intersections and return the appropriate object ghost velocity as described in [42], simplifying object interaction implementations significantly.

[88] used only the normal component of the velocity to advect the fuel level set, which is sufficient for their WENO scheme that was applied without particles. We instead use semi-Lagrangian advection for the level set equation, and this requires the tangential component of velocity as well in order to properly trace characteristics. Since the tangential component is continuous across the interface, we form the normal component as usual and simply add the tangential component from the local fluid velocity. The tangential component is required for particle advection as well.

### 3.8 Examples

To demonstrate the effectiveness of our approach, we simulated a number of examples that range in resolution from $150^3$ to $350^3$ on a number of 4 processor Opteron machines. The computational cost for the examples range from 5 to 50 minutes per frame. Surface tension was the main cause for the examples with slower simulation
times. We augmented a standard ray tracer to use the same projection based querying of the level set functions that the simulation uses, since rendering each level set independently can lead to multiple intersections per interface (from numerical error).

Figure 3.9 depicts two drops suspended in liquid. The lower drop has a lower density than the surrounding fluid and the upper drop has a higher density. Both drops have surface tension. Figure 3.2 depicts a kinematic sphere splashing into a number of liquids. The air region is simulated as a Dirichlet region, and the liquids are of increasing density from top to bottom. Figure 3.5 shows four layers of fluid where the lower middle liquid is lighter than the top middle liquid. This causes a Rayleigh-Taylor instability as the two liquids switch places.

Figure 3.7 shows a number of different fluids on an inclined plane. The liquid with the highest viscosity is blue, then green, then silver, and finally the clear water is simulated as inviscid. Figure 3.12 depicts a viscoelastic armadillo in a pool of water. The viscoelastic property is then removed making it viscous only, then the viscosity is turned off and the density is turned down making it bubble up to the surface. The former armadillo is then changed back to viscoelastic and a newly introduced viscoelastic liquid is dropped on top of it.

Figure 3.11 depicts flammable oil being released into a tank of water. The oil rises to make a layer on the surface, which is then ignited. This example uses a temperature based ignition model where the temperature in actively burning regions is $T_{\text{max}}$, but a lower $T_{\text{ignition}}$ is needed to cause ignition of surrounding fluid. Figure 3.10 shows two viscous liquids that react with each other creating a third (air) that bubbles up to the surface. In Figure 3.6, eight letters with various high densities and viscosities splash into a pool of water and sink to the bottom. The air is treated as an empty region. In the second part of the simulation, the letters are changed to be low density fuels with surface tension, and the air is fully simulated. The letters then rise through the water, bursting into flames as they break the surface.
3.9 Conclusions and Future Work

Notably, the new technique does exacerbate the limitations of the original particle level set method with regards to volume loss by facilitating increased scene complexity. This is evident in the Rayleigh-Taylor simulation depicted in Figure 3.5. In that example, the majority of the mass loss occurs away from triple points, in regions where our method is identical to the original particle level set method (see section 3.3.1). Of course, this can be addressed by increasing the particle count for the particle level set method or using an adaptive octree or run length encoded type level set simulation, at the expense of increased code complexity and/or CPU time.
Figure 3.1: Each of the three regions is independently evolved in time, after which the interface locations do not agree. There are vacuums where all $\phi_i$ are positive, and overlaps where more than one $\phi_i$ is negative. The dotted black line shows the new interface locations after our projection step.

Figure 3.2: A kinematically controlled sphere splashing into a multi-layer pool (300 $\times$ 300 $\times$ 200 grid, 4 phases).
Figure 3.3: (Left) Two level sets initialized so that properties 1 and 2 hold. (Right) After evolving in time, we obtain the solid lines with overlap (both negative in the middle). The dotted lines show an example result after projection. Not only has the overlap been removed, but the interface location is preserved.

Figure 3.4: (Left) The solid lines have overlap on the left (two $\phi_i$ negative) and a vacuum on the right (all $\phi_i$ positive). The average of the smallest two level sets at any point is subtracted from $\vec{\phi}$ to obtain the dotted lines. Not only has the overlap and vacuum been removed, but the smallest $\phi_i$ is preserved and negative at each point preserving interface locations and inside/outside information. (Right) Results after reinitializing each level set to a signed distance function.
Figure 3.5: Rayleigh-Taylor instability (300$^3$ grid, 4 phases).
Figure 3.6: Viscous letters splash into a pool of water, then change into low density inviscid fuel bubbling up and burning when they hit the surface (350 × 200 × 350 grid, 10 phases).
Figure 3.7: Different viscosity liquids interacting on an inclined plane. (300×150×240 grid, 5 phases).
Figure 3.8: (Left) One way coupling from liquid to air. (Right) Two way coupling of liquid and air. Note the surface ripples and the unstable stream of liquid in the fully coupled simulation.
Figure 3.9: Two drops with high surface tension collide. Green has low density, red high density (350³ grid, 3 phases).
Figure 3.10: Two submerged liquids meeting and reacting to create air (150³ grid, 4 phases).
Figure 3.11: Oil pouring into water, then catching on fire. Note that the fiery ball is a separate phase of fluid, and that it deforms into the shape of a droplet as it falls (200³ grid, 4 phases).
Figure 3.12: An armadillo that starts out viscoelastic, becomes viscous and more dense than the water, then inviscid and lighter than the water, and finally viscoelastic again before another viscoelastic liquid is dropped onto it (250 × 275 × 250 grid, 4 phases).
4.1 Introduction

Simulations of water, smoke, and fire are becoming increasingly important in computer graphics applications such as feature films, since it is often costly, dangerous, or simply impossible to film the desired interactions between these fluids and their surroundings. These difficulties are exacerbated when the surrounding environment is undergoing complex motion or destructive modification, making it advantageous to combine fluid effects with simulations of rigid and deformable objects. Ideally, one would like to simultaneously incorporate the full range of available fluid and solid behaviors in a simulation including interactions between different components. This leads naturally to the concept of handling phase changes between solids and fluids.

Historically, particle methods have been popular for the physical simulation of both solids and fluids as well as phase changes between them, and [122] applied them to softening and melting behavior in graphics early on. These methods avoid the need to maintain connectivity information for the solid phase, which simplifies topological change and transition between different material behaviors. Unfortunately, particle methods have not yet reached the quality and efficiency of specialized techniques for fluid or solid simulation. The lack of exact topological information for solids
Figure 4.1: A sheet of material constrained at its four corners catches fire. When the top corners burn away from the rest of the material, it falls to the ground (121x193x121 grid, 13K triangles).

makes collision and self-collision problematic and thin sheets such as cloth almost impossible to simulate. High particle densities are typically required to achieve good accuracy for fluids, and it is often difficult to generate a smooth renderable surface from the resulting set of points. Moreover, while there has been recent progress in specialized methods for both solids (e.g. [83]) and fluids (e.g. [96]), researchers have not yet coupled together the specialized treatments devised for each. For example, [83] softens solids to the point where they start to melt and flow, but does not consider (or obtain) high quality particle based fluid effects as in [96].

The aim of our approach is to couple the highest quality simulation techniques for
fluids and solids together, and thus we consider Eulerian grid-based techniques for fluids and Lagrangian mesh-based techniques for solids as in [42]. See also [12], although that method is not applicable to the deformable or thin materials treated in [42]. The opposite approaches of Lagrangian mesh-based fluids and Eulerian grid-based solids typically have only limited success, although there has been some interesting recent work with high viscosity and viscoelastic fluids. [11, 97] used high viscosity fluid to model melting and flowing, and [39] compute elastic forces resisting deformation by advecting a strain field along with the fluid. While these techniques allow the simulation of a much greater range of fluid phenomena, they do not support accurate self-collision, rolling, etc. as a tetrahedral mesh can, and cannot be applied to thin objects such as cloth.

For the fluid phase, we use state of the art simulation techniques for smoke [26], water [21] and fire [88]. We use standard Lagrangian mesh-based techniques for deformable objects (in particular [5, 6, 41, 52]), except that we embed the surface geometry of the object in a parent simulation mesh as in [79] to allow smooth erosion of the surface during melting or burning. This is essentially a free form deformation (FFD) [104]. For more on FFD’s applied to simulation, see for example [23, 10, 9, 86, 54].

[73, 74] used level sets to model the erosion of burning rigid objects (see also [135]), but did not consider thin objects, deformable objects or melting. We similarly use level set methods to simulate the erosion of solid material, but desire a direct implementation on the object’s tetrahedral or triangular simulation mesh in material coordinates enabling the treatment of both thin and deformable objects. In order to avoid the difficulties associated with implementing level set algorithms on tetrahedral and triangular meshes, we instead evolve the level set on a background octree (or quadtree) grid in material coordinates interpolating the results to the simulation mesh. This allows us to directly leverage the level set algorithms proposed in [67] for octree data structures. The values of the level set function allow us to regenerate the evolving embedded geometry when necessary, and we use dynamic red-green refinement to increase the resolution of the simulation mesh near the boundary geometry as it changes over time. In fact, due to the special structure of our red-green refinement,
the nodes of the simulation mesh correspond exactly to a subset of the nodes of the octree grid reducing interpolation to the trivial copying of data. Figure 4.1 shows examples of this technique applied to the burning of a material sheet.

4.2 Previous Work

[32] popularized the use of the incompressible three dimensional Navier-Stokes equations in computer graphics, and this work was subsequently extended with semi-Lagrangian advection and vorticity confinement in [112] and [26] respectively. Although it is not possible to discuss all the varied research in this direction, variations have been used to model fire [62, 88] and explosions [28, 98] (with compressible flow in [131]), control [125, 24], and flows on surfaces [113]. There is also the recent work of [106] that hybridizes grid-based methods with vortex particle techniques. Liquids have received a lot of focused attention including the early work of [57, 13], the particle and cell based approach of [30, 31], and the hybridized particle and level set approaches in [29, 21]. Additional work has been done on surface tension [47, 67], control [72, 76, 97, 108], solid fluid coupling [36, 84], contact angles [128], sand [136] and two phase flow [48].

4.3 Eroding Thin Shells

We simulate solids as triangular or tetrahedral meshes, and use standard element-based force computation and collision algorithms properly adjusted for embedding (see e.g. [79]). Since the solid is continuously eroding into fluid during melting or burning, we need to dynamically erode the mesh as material disappears while maintaining a smooth surface for collisions and rendering. Thus throughout the simulation, we adaptively refine the boundary elements of the simulation mesh to improve accuracy while preserving large elements elsewhere for efficiency. For a thin shell such as cloth where every element is on the surface requiring reasonable resolution, this adaptivity is primarily useful to resolve any complicated features at the cloth’s edge (and was not required for our examples). However, for volumetric tetrahedral
meshes, simulations are far more expensive if the invisible interior of the object is not discretized with larger elements. Since the algorithms and data structures required for two dimensional shells are quite similar to those for three dimensional volumetric solids, we focus on the simpler case of thin shells first.

In order to support dynamic mesh adaptation, we begin with the uniform body-centered square (BCS) triangular lattice which is the two dimensional analog of the body-centered cubic (BCC) lattice used in mesh generation by [80, 7] (they also used red-green refinement similar to what we propose here). The BCS lattice is given by taking the vertices of a uniform square grid in space together with the centers of each square (the vertices of the dual grid), and placing four triangles in each square by connecting the four edges to the center node as shown in Figure 4.4 (upper left). We then perform red-green refinement initially labeling all BCS triangles as red. Any red triangle can then be red refined into four red triangles of half the size as shown in Figure 4.2. The resulting smaller triangles are exactly the BCS triangles from a grid of twice the resolution. After a sequence of red refinements, T-junctions are removed with green refinement which replaces a red triangle with two green triangles of possibly lower quality (Figure 4.2). Since we only allow one level of green refinement to ensure high quality elements, the initial red refinement is iterated until neighboring triangles are at most one level apart and no triangle has more than one T-junction. See Figure 4.4 (lower left) for an example of a red-green refined mesh. Note that if a green triangle later requires additional refinement, it and its sibling must be removed and replaced with a red refinement of the parent red triangle.

Since all triangles in a BCS mesh are aligned in a Cartesian grid structure, refinement alone is not enough to match a smooth boundary curve. [80, 7] solved this
problem by compressing the mesh to the boundary before simulation. This is computationally infeasible in the context of melting or burning, since it would have to be performed after every step of the simulation. Instead, we leave the simulation mesh unchanged, and define the actual boundary of the object as the zero-isocontour of a level set function which is stored on the nodes of the simulation mesh. The surface of the object can be extracted directly from the level set function with marching triangles (the two-dimensional analog of marching tetrahedra), and any simulation triangles lying completely outside the level set can be removed. Figure 4.4 (upper right) shows the results obtained for a circular piece of geometry.

With the ability to spatially adapt our mesh based on the values of a level set

Figure 4.3: A burning sheet of material rendered without the fire to illustrate the smooth boundary of the solid (13K triangles).
function, simulating the loss of solid material (to the liquid or gas phase) is readily accomplished by smoothly increasing the nodal values of the level set function (and remeshing on the fly). However, calculating the change in level set values due to combustion, phase change, etc. can be daunting on our adaptive red-green data structure, since it typically requires the solution of partial differential equations on an only partially structured non-Cartesian BCS (or BCC) mesh. A key aspect of our approach is to avoid the solution of partial differential equations on arbitrary meshes with moving points, instead solving the governing equations for level set evolution in material space where the grid is static (i.e. grid points do not move). Moreover, we solve these equations on a structured Cartesian background grid interpolating data back and forth to the red-green simulation mesh. In fact, we choose a quadtree grid that exactly matches the red-green adaptive structure of the BCS mesh.

We emphasize that the match is exact, i.e., the vertices of the red-green simulation mesh lie exactly on top of the grid points of the quadtree grid as shown in Figure 4.4 (lower right). This means that there are no interpolation errors for moving data back and forth between the red-green mesh and the quadtree grid, and the computational cost is limited to the simple copying of data. Although all the red-green vertices correspond to quadtree nodes, there are some extra nodes in the quadtree grid that do not correspond to red-green vertices. Before computations can be carried out on the quadtree grid, values for these nodes must be interpolated or extrapolated from valid data on the red-green grid or the already defined quadtree grid nodes. If a quadtree node lies inside a red-green triangle, its value can be interpolated using barycentric coordinates. The remaining nodes can be filled in using the standard level set reinitialization and extrapolation equations on the quadtree grid. This background quadtree grid allows us to leverage all level set algorithms implemented on quadtree data structures including advection, motion by mean curvature, reinitialization, extrapolation of data across an interface or isocontour, the particle level set method, elliptic solvers, etc., see e.g. [67] and the references within. Note that if adaptivity is not needed, the use of an unrefined BCS mesh allows us to solve level set algorithms on an even simpler uniform grid.
While the red-green refinement strategy applies to any mesh, the quadtree correspondence applies only to grid aligned meshes. In particular, it does not hold for equilateral triangle meshes. Also, while the back and forth interpolation strategy used for updating the level set function could be implemented with a different background grid, the matching quadtree grid gives an optimal match for vertices and refinement structure minimizing computational cost and interpolation error.

Once the red-green mesh and boundary are constructed, the simulation proceeds along standard lines. Values such as world position and velocity are interpolated to new nodes as they are generated during refinement. Values from the simulation mesh are also interpolated to the embedded boundary, which is used for collision, self-collision and fluid interaction.

4.4 Eroding Volumetric Solids

Two-dimensional geometric methods do not typically carry over elegantly to three spatial dimensions. However, we designed our two-dimensional approach with three dimensions in mind, and thus the method carries over almost entirely. In three spatial dimensions, we tile space with a body-centered cubic (BCC) lattice and form tetrahedral elements by considering the vertices of the uniform cubic grid together with the cell centers. Space is divided into octahedrons by considering the region connecting each cubic face to its neighboring cell centers, and four tetrahedra are placed in each octahedron. See Figure 4.5. Red refinement of a tetrahedron produces eight smaller BCC tetrahedra, and T-junctions are removed using several varieties of green refinement which result in two or four green children. See Figure 4.6. Overall, our approach is similar to that in [80, 7], and we refer the reader to their work for further details.

Once again, we define a level set function on the nodes of the red-green simulation mesh, and use marching tetrahedra and embedded simulation technology similar to that proposed in [79]. Most importantly, we implement all our level set methods on an overlayed background octree grid in material space. Just as in two spatial dimensions, the nodes of the octree grid correspond exactly to those of the tetrahedra in
the adaptive red-green simulation mesh. This makes the method both computationally efficient and straightforward to implement leveraging on [67] and the references therein.

For ease of implementation, rigid body surfaces are generated in the same manner as the deformable case, even though the tetrahedral mesh is not used during rigid body dynamics. This incurs no significant computational cost, since the vast majority of the total simulation time is spent solving for the fluid dynamics.

### 4.5 Creating Fluids from Solids

In the case of melting, the solid and liquid phase have similar densities, so a given volume of solid material is converted into approximately the same volume of liquid. In contrast, the density ratio between a solid and its gaseous products is typically several orders of magnitude, and a large volume of gas is released before any noticeable amount of solid disappears. Thus, we simulate the production of fluid from solid in very different ways for the melting and burning cases. Moreover, we do not consider burning volumetric solids, since it takes a long time to see changes in the solid. Instead, we consider burning shells where there is very little solid material, and thus it erodes away quickly. In addition, we only consider the melting of volumetric objects, since melting shells produce only a small amount of liquid that is difficult to resolve on a computational fluid dynamics grid.

### 4.6 Creating Liquid

We model the liquid phase using the particle level set method of [21]. The bulk of the fluid is represented by a level set function defined on the grid with negative values denoting liquid and positive values denoting air. The level set provides clean handling of topological merging and breaking, but suffers from volume loss in regions of high curvature due to numerical diffusion. To alleviate this problem, [29] proposed the use of particles passively advected with the fluid velocity to replace liquid lost to dissipation. After each advection step, each particle was treated as a small sphere
and added to the level set with a CSG union. [21] later extended this technique by adding a layer of positive particles outside the fluid in addition to the negative particles inside, which reduces the opposite problem of volume gain (air loss) due to dissipation in the other direction. That and other proposed modifications made the method accurate enough for scientific calculations as well as visually compelling. Since particles exist only in a thin band around the interface and do not need to interact, they can be seeded at higher resolution than the fluid grid to improve accuracy. The particle radii used for level set modification are therefore much smaller than a grid cell, which may result in particles failing to convert a surrounding node to liquid and incorrectly crossing over the the level set zero isocontour. These removed particles move ballistically and collide with objects until they occur in high enough density to convert grid nodes into fluid. [29] rendered these removed particles as spray droplets.

As the solid melts, we generate liquid at the surface of the solid using techniques synergistic with the particle level set method. At the beginning of the simulation, we seed liquid particles inside the solid in material space using the same particle density and radius distribution used for the particle level set method when modeling the liquid. Note that this seeding is readily implemented on the quadtree or octree grid in material space. Then, as the level set evolves on the material space quadtree or octree grid, we make note of any particles that cross completely over the level set based on their radii effectively moving outside the solid. These particles are moved from material space to world space (in the same way embedded particles are carried along by the simulation mesh) and used to generate fluid. Fluid is generated by adding these particles directly to the particle level set simulation as removed particles with an initial velocity determined as if the particle were an embedded particle. Some of these particles will be in free flight and generate new fluid (by modifying the level set function in the usual manner), while others will already be interior to the fluid and thus immediately reincorporated into the particle level set simulation as negative particles. This allows thin sheets of melted liquid to form on the object surface before it would be possible to resolve the flow on the fluid dynamics grid. This process is illustrated in Figure 4.11.

The negative particles in the particle level set method are used only to resolve
the liquid interface, and do not otherwise influence the physics of the liquid. Thus they have no mass or volume information, and do not interact with each other except through the Eulerian grid. If these particles venture too far across the interface into the air region and do not occur in dense enough populations to create new liquid volume, they become removed particles and are treated ballistically as droplets in free flight. This could likely be improved by modeling these removed particles with a smoothed particle hydrodynamics (SPH) method such as that proposed in [82, 96]. Additionally, the negative particles interior to the liquid volume could be used as boundary conditions for the removed particle SPH simulation. Regardless, even without this, our approach produces compelling animations.

4.7 Creating Gas

We use the fire simulation method of [88], which models fire as a blue core of unburnt fuel undergoing a combustion reaction into hot gaseous products. A level set is used to track the location of the blue core, and the products are described by temperature and density fields. Since the density of the gaseous fuel is higher than the resulting products, the gas expands as it crosses the level set interface and is converted from fuel to products. The rate of this expansion can be derived by solving the jump conditions for conservation of mass and momentum as described in [88]. To model the sourcing of fuel from kinematic solid objects, they constrain the velocities at the boundary of the object to inject fuel at a given rate (which can also be determined from the jump conditions).

When a region of solid catches fire and begins burning, we continuously source gaseous fuel around the burning region while maintaining a reaction coordinate on the vertices of the solid describing the progress of the combustion reaction. This can also be used for browning effects at render time. We model the expansion from solid into gaseous fuel by generating a thin band of fuel level set around the burning region and setting the divergence in this region to a positive value when solving for the pressure forcing the fuel to move outwards. The amount of gas produced can be directly linked to the rate at which the solid is disappearing, although the animator
can of course control this effect as desired. Divergence sourcing was also used in [28] to model suspended particle explosions by making each suspended dust particle a small divergence source.

4.8 Two Way Solid Fluid Coupling

We follow the solid fluid coupling strategy proposed in [42] which allows one to use their favorite (and previously implemented) methods for simulating the solid and the fluid independent of the coupling strategy, unlike [12] which requires special treatment of the solid, i.e. treating it as some sort of fluid in order to obtain two way coupling. Thus while [12] can only handle rigid volumetric bodies that can be rasterized and represented on the fluid grid, [42] showed full two way coupling for deformable bodies as well as very thin objects such as cloth. The main philosophical difference between the two approaches is that [42] computes the effect of the fluid on the solid via a force per vertex that can be added to any solid simulation technique, while [12] allows the fluid to completely determine the solid’s velocity overwriting all internal dynamics (such as elastic deformation). In addition, the final projection step of [12] takes the continuous velocity field computed for the combined solid/fluid domain and constrains the grid cells within the solid to have a velocity consistent with rigid body motion. This makes the velocity field discontinuous at the surface of the solid, allowing fluid to flow directly into the solid and disappear. In contrast, [42] applies Neumann boundary conditions at the surface of the solid enforcing the fluid’s normal velocity to be identical to that of the solid at the solid/fluid interface.

[42] focused on thin objects, whereas we are also interested in volumetric objects. Thus, we review some of their techniques as well as propose a couple of algorithmic improvements. Recall that the projection method for incompressible flow first computes an intermediate velocity \( \mathbf{u}^* \) and then solves for the pressure \( p \) needed to make
the final velocity $u^{n+1}$ divergence free:

$$u^* = u^n - \Delta t(u^n \cdot \nabla)u^n + \Delta t g$$  \hspace{1cm} (4.1)$$
$$\nabla \cdot (\nabla p/\rho) = \nabla \cdot u^* / \Delta t$$  \hspace{1cm} (4.2)$$
$$u^{n+1} = u^* - \Delta t \nabla p/\rho$$  \hspace{1cm} (4.3)$$

We use the new node based fluid solver of [42] which stores the permanent velocity on the cell faces of a MAC grid. At the beginning of each time step, the face velocities are averaged to the nodes and used in Equation 4.1 to compute $u^*$ at the nodes. Then we calculate a scaled force $\Delta u = u^* - u^n$ at each node, average this back to the faces, and use the scaled force at each face to increment the face velocity to its intermediate value. Finally, the standard MAC grid based pressure solver is used to make this intermediate face velocity divergence free. This scheme allows one to calculate the intermediate velocity using a simple node based method, while still using a standard MAC grid approach to solve for the pressure. Moreover, it avoids direct averaging of the velocity field significantly reducing numerical dissipation that gives the fluid an unwanted viscous appearance. When solving for the pressure, Neumann boundary conditions are applied at cell faces located within solid objects using the effective velocity of the solid, which is calculated by evolving the solid forward in time by the size of the next time step. Using the effective velocity (as opposed to the instantaneous velocity) greatly reduces mass loss, especially when thin liquid films interact with solid objects.

In [42], they first compute the intermediate fluid velocity $u^*$ using knowledge of the solid’s position at both time $n$ and time $n+1$. Then they solve the variable density Poisson equation $\nabla \cdot (\nabla p/\rho) = \nabla \cdot u^{old}/\Delta t$ for the solid fluid coupling pressure. The density at each face is set to either the fluid’s density or the solid’s density, and $u^{old}$ is set to $u^*$ in the fluid region and to the effective solid velocity inside the object. The resulting pressure is interpolated to the barycenter of each triangle and multiplied by the triangle area and normal to obtain the net force on a face. For rigid bodies the force and torque are calculated by accounting for all the triangles, and for deformable objects one third of the net force is distributed to each node. These forces are used
when the solid is evolved from time \( n + 1 \) to time \( n + 2 \), and that motion is used to compute the effective velocity which is used to project the intermediate fluid velocity \( \mathbf{u}^* \) to be divergence free at time \( n + 1 \) concluding the time step from \( n \) to \( n + 1 \).

The pressures are defined at the cell centers of a MAC grid making interpolation to the barycenter of triangles difficult near the boundaries of the domain. This can be remedied by extrapolating pressure values into a one ring of ghost cells surrounding the domain, but this causes errors in the case of solid walls. We propose a new method that accurately calculates pressure derivatives at solid wall cell faces on the boundary of the domain, and then we use those derivatives to fill in pressure values in the ghost cells assuming a linear pressure profile across each cell face. (Of course, this same method can be used for kinematic interior boundaries as well.) To do this we point out that the pressure calculated from the variable density Poisson equation is typically used to update the fluid velocity via equation 4.3, although we do not do this since this pressure is merely used to calculate the force the fluid applies to the solid. We instead use the appropriate component of this equation to solve for the pressure derivative at a wall using the density of the fluid for \( \rho \) and the final known wall velocity \( u_{\text{new}} \) for the new velocity. For example, the pressure derivative across a face normal to the \( x \)-axis would be \( p_x = \rho (u_{\text{old}} - u_{\text{new}}) / \Delta t \). The old velocity is calculated by updating the nodes on the wall to their \( \mathbf{u}^* \) values and averaging these values to the cell face. Note that substituting this pressure derivative into the variable density Poisson equation leads to the replacement of the old velocity value with the new value, which is exactly equivalent to the standard practice of setting the pressure derivative to zero and using the new solid velocity value on a face. Thus one does not need to calculate \( \mathbf{u}^* \) on the cell face unless the true (nonzero) pressure derivative is desired there.

Besides this improved treatment of the pressure near kinematic boundaries, we also propose improving the overall accuracy of the fluid to solid coupling forces. The main idea is to compute a more accurate representation for \( u_{\text{old}}^{dd} \) in the cells occupied by the object. For example, consider a stationary neutrally buoyant object submerged in a still fluid, where it should remain at rest. In order for the variable density Poisson equation \( \nabla \cdot (\nabla p / \rho) = \nabla \cdot u^{dd} / \Delta t \) to give the correct coupling pressure, \( u^{dd} \) inside the
solid should be identical to the fluid’s $u^*$, which is $\Delta t g$. If $u^{old} = 0$ was used inside the solid region (corresponding to the solid’s instantaneous or effective velocity), an incorrect coupling pressure would arise due to the incompatibility between the solid and fluid velocity causing the solid to incorrectly accelerate. Instead of using the effective velocity of the solid which represents the change in the solids position from time $n$ to time $n + 1$, we subtract out the effects of the fluid to solid pressure coupling that was used to update the solid from time $n$ to time $n + 1$. This velocity is more accurately in line with the intermediate $u^*$ velocity computed for the fluid, which accounts for all the forces except those due to the pressure.

### 4.9 Examples

For all our examples, the additional cost of melting or burning was small compared to the time required to couple solid and fluid together without phase change. The total computational cost averaged between 5 and 15 minutes per frame.

The single torus in Figure 4.7 was melted at a constant rate by uniformly increasing the values of the level set function on the octree grid in material space. For the other melting examples, we set the melting velocity to be a linear function of temperature multiplied by the level set normal on the octree background grid. In the multiple tori example (Figure 4.10), the temperature was defined procedurally to be large in a band near the table. In Figure 4.8, we inject hot water advecting the temperature with the liquid volume (Figure 4.9). Figure 4.13 shows ice cubes melting after being dropped into warm water. For rendering, we place a small ellipsoid around both negative and removed negative particles and blend it with the level set before ray tracing. The aspect ratios of the ellipses are based on the velocity of the associated particle.

To model the material catching on fire in Figures 4.1, 4.12 and 4.14 we store a temperature field on the nodes of the simulation triangle mesh and gradually relax that temperature towards the temperature of the surrounding air, igniting the material when a prescribed ignition temperature is reached. Then we use a bandwidth of 2 grid cells to generate a fuel level set around the solid, and apply divergence sourcing in a band of 2.5 grid cells. The solid level set function is set to $\phi = Y - Y_{max}$, where
$Y$ is the reaction coordinate and $Y_{\text{max}}$ indicates full combustion, causing the surface to erode once the reaction completes.

Since we have complete control over the level set function defining the boundary of the solid, we can readily control the pattern of melting or burning. In Figure 4.12, a region of the material is constrained to be inflammable by performing a CSG union with a fixed shape and sourcing fuel only from the flammable region.

### 4.10 Conclusions and Future Work

We have presented a novel technique for simulating the phase change of solid objects modeled by Lagrangian meshes into fluids defined on Eulerian grids. Examples were presented to demonstrate that this algorithm works well for both the melting of volumetric solids into liquid and the burning of thin sheets of material into gas. Simulation meshes for the solid are constructed by adaptive red-green refinement starting with a BCS or BCC lattice, and the boundary of the object is specified by a level set function defined on the nodes of the simulation mesh which is evolved on a background quadtree or octree grid in material space. The most important benefit of the method is the ability to use state of the art techniques for both the solid and fluid without compromising simulation quality in order to couple them together or convert one into the other.

There are several areas for improvement for both the solid and fluid phases. For solids, significant work is needed to make embedded collision handling as accurate as the nonembedded case. Moreover, while embedded meshes allow for smooth changes in geometry, the mass distribution changes discontinuously when simulation nodes disappear, which can result in small popping artifacts. This could be alleviated by adjusting the masses of boundary points as smoothly as possible while clamping them away from zero for stability. Finally, the use of the BCS-quadtree correspondence for arbitrarily curved shells would require dividing the surface up into rectangular patches. Fortunately, the quadtree correspondence is unnecessary in the burning case (at least for our ad-hoc model) since the level set is defined procedurally and all other steps in the algorithm carry over to an irregular mesh (including red-green
refinement). In the case of melting volumetric objects, the octree correspondence is used frequently but does not restrict the class of objects which can be simulated.

Both melting and burning require the representation of details at close to the resolution of the grid, and some aliasing is visible in the burning simulations. Some of this is due to the use of linear interpolation during rendering, and could be dramatically improved with smoother thin shell-aware interpolation schemes. Further improvement would require better object boundary conditions for the fluid solver.
Figure 4.4: Starting with a uniform body-centered square mesh in material space (upper left), we apply red-green refinement and discard triangles completely outside the object (upper right). The simulation is then performed on the red-green parent mesh (lower left). Level set operations are performed by overlaying a quadtree grid on top of the red-green structure, noting that each mesh vertex is also a node in the quadtree grid (lower right). A major advantage of this approach is that the front evolution is carried out on a quadtree mesh in the two-dimensional material space even as the object deforms in three spatial dimensions.
CHAPTER 4. MELTING AND BURNING SOLIDS INTO LIQUIDS AND GASES

Figure 4.5: A portion of the BCC lattice. The blue and the green connections depict the two interlaced grids, and the eight red connections at each node lace these two grids together.

Figure 4.6: Red refinement produces eight children that reside on a BCC lattice of half the size (left). Three types of green refinement are allowed in order to remove T-junctions (right).
Figure 4.7: An elastic torus bounces and rolls on the ground as it melts, generating thin sheets of liquid (242x121x121 grid, 22K tetrahedra).
Figure 4.8: Ten rigid body simulated ice cubes are melted by a stream of hot water (100^3 grid, 600K total surface triangles).
Figure 4.9: The melting speed of the ice cubes is derived from a temperature field advected with the liquid velocity.
Figure 4.10: Six elastic tori fall onto a hot surface causing them to quickly melt into liquid (181x61x181 grid, 160K total tetrahedra).
CHAPTER 4. MELTING AND BURNING SOLIDS INTO LIQUIDS AND GASES

Figure 4.11: A melting solid partially immersed in liquid. Particles are seeded throughout the solid with the same distribution and properties as the particle level set particles (left). As the solid melts, particles leave the solid and are converted to removed liquid particles. Some of these particles are close enough to grid nodes to convert them to liquid (middle). More liquid is generated as the solid continues melting (right).
Figure 4.12: A predefined level set is used to prescore part of a thin material sheet as inflammable (121x193x85 grid, 37K triangles).
Figure 4.13: Rigid ice cubes floating and melting in water with full two-way force coupling (100³ grid, 600K total surface triangles).
Figure 4.14: Burning cloth draped over an armadillo figurine (89x122x78 grid, 18K triangles).
Chapter 5

Coupled SPH and Particle Level Set Simulation

5.1 Introduction

The physics-based simulation of water has become prevalent in modern feature films, especially for scenes that include realistic secondary effects such as spray, foam, or bubbles. Such simulations have been used extensively in both photorealistic [35, 53] and animated [2, 8, 123] features. We differentiate between dense and diffuse water volumes noting that the incompressible Navier-Stokes equations are not appropriate for modeling diffuse regions such as spray/air mixtures which are clearly compressible. While dense volumes are adequately modeled with the incompressible Navier-Stokes equations as in [21], SPH methods are more appropriate for spray and foam because they can more accurately reflect the physical characteristics of the diffuse or aerated material. Thus, we propose a novel SPH method that is suitable for both dense and diffuse regions as well as the interactions between them. Moreover, since state-of-the-art grid-based solvers yield high fidelity results for dense regions (and especially for smooth surfaces), we also show how to two-way couple our SPH solver to the particle level set method.

Particle systems were first shown to be useful for computer graphics applications in [99]. Early works on using particles to model liquids include [122, 77, 124].
[38, 71] introduced the notion of smoothed particle hydrodynamics in which spatially
smoothed particle representations can be used to solve the Navier-Stokes equations.
[114] leveraged the smoothed particle representation to model gaseous phenomena,
and [18] later introduced the full SPH methodology to the graphics community. [18]
used an equation of state (EOS) to model pressure, thus imposing a severe time
step restriction to resolve the sound waves present in compressible flow (similar to
[132]). Notably, the computational cost of their method increases as the desired
compressibility decreases, becoming most expensive in the incompressible flow limit.
Contemporaneously, [30, 31] introduced another method for simulating particles with
the Navier-Stokes equations based on [45]. Instead of integrating the Navier-Stokes
equations on a smoothed particle basis, they carried out all calculations on a back-
ground grid, thus allowing for an efficient, fully incompressible simulation where the
implicit handling of the acoustic waves removes the related time step restrictions (see
also [32]). Later, [29] used level sets to represent the liquid interface downgrading
particles from physical representations of fluid mass to auxiliary markers for interface
tracking. Although this trend was continued by [21] adding interface trackers on the
air side of the interface, [29] and subsequent papers [42, 59, 68, 40] used these marker
particles to represent spray and bubbles when they crossed over the interface.

Although various authors mixed grid-based solvers with particle methods for spray
and foam using explicit rules for particle behavior [91, 121], one attractive aspect of
the particle level set method is that it automatically produces particles in under-
resolved regions. This is similar in spirit to the particle finite element method [90]
where a standard finite element mesh is created for particles dense enough to form
a continuum, and stray particles are simulated with methods more appropriate for
spray. As researchers explored the strengths and weaknesses of different techniques,
the distinction between grid-based methods and particle-based methods has blurred.
For example, many vortex particle algorithms make use of some sort of background
grid in order to decrease computational cost and algorithmic complexity (see e.g.
[106]), and [136] use a background grid for all non-advection terms in their particle-
based fluid solver. [59] attempted to use SPH for the removed particles in a standard
particle level set implementation, but had difficulties using EOS-type methods to
adequately enforce incompressibility. Thus, they handled the removed particles with a method similar to [136]. Our method is most similar in spirit to these works.

5.2 Previous Work

Building on the initial work of [18], [115] used EOS-based SPH for lava flows. This EOS SPH framework was also used in a series of papers to simulate water [82], melting solids [83, 130, 111], solid fluid coupling [84, 58], and multiphase flows [85]. [94] pointed out that the typical SPH EOS methods for simulating incompressible fluids lead to very stiff systems, making incompressible flow difficult to simulate. In fact, [96] states that SPH methods can only solve compressible fluid flows and proposes an SPH variant which does not use an EOS relationship for the pressure, but instead solves a global Poisson equation similar to grid-based methods. They obtain some rather impressive simulations of liquids.
Figure 5.2: SPH liquid flowing into a $120 \times 240$ box. The target particle number density is set to be low everywhere except in a region delineated by outlines of the SIGGRAPH logo.

5.3 Particle Level Set Method

Our fluid solver is predicated on previous grid-based Navier-Stokes implementations such as [26], which ignore viscous effects and use the inviscid form of the Navier-Stokes equations

$$\vec{u}_t + (\vec{u} \cdot \nabla)\vec{u} + \nabla p/\rho = \vec{f}$$

$$\nabla \cdot \vec{u} = 0$$

where $\vec{u} = (u, v, w)$ is the velocity, $\rho$ is the density, and $\vec{f}$ accounts for body forces such as gravity and vorticity confinement. First, an intermediate velocity field $\vec{u}^*$ is computed

$$\frac{\vec{u}^* - \vec{u}^k}{\Delta t} + (\vec{u}^k \cdot \nabla)\vec{u}^k = \vec{f}$$

using a second-order unconditionally stable MacCormack method [105]. Since the modified MacCormack method can create spurious oscillations when sampling extrapolated velocities, we revert to the standard first-order accurate semi-Lagrangian
CHAPTER 5. COUPLED SPH AND PARTICLE LEVEL SET SIMULATION

Figure 5.3: Two-way coupling between SPH and the particle level set method to pour water into a glass.

method [112] near the liquid/air interface and object boundaries. Then, we compute a scaled pressure \( \hat{p} = p \Delta t \) via

\[
\nabla \cdot (\nabla \hat{p} / \rho) = \nabla \cdot \vec{u}^* \tag{5.1}
\]

and use it to make the velocity field divergence free

\[
(\vec{u}^{k+1} - \vec{u}^*) + \nabla \hat{p} / \rho = 0. \tag{5.2}
\]

We use the standard particle level set method to model the interface with particles on both sides as in [21]. The removed particles generated by the particle level set method are used to simulate secondary effects like spray and foam via our new SPH solver.

[18] proposed an SPH method predicated on an EOS of the form \( p = k(\rho - \rho_0) \) where \( \rho_0 \) represents the target density of the fluid. However, [81] noted that equations of the form \( p = k \rho \) are generally better behaved since attractive forces between particles are known to cause instabilities in SPH simulations. While this latter formulation is more stable, it no longer provides any mechanism for density targeting. Density targeting is important for adaptivity: being able to control the spatial density essentially allows one to create multiresolution SPH simulations in the
same way that octrees and RLE grids allow for multiresolution simulations.

To avoid the time step restrictions induced by acoustic waves in EOS-based SPH models, one can instead solve a global Poisson equation for the pressure similar to the standard grid-based methods. [96] took this approach solving a Poisson equation that targeted the desired number density of particles based on the work of [61, 60]. These same authors ([133]) later realized that it is desirable to have an incompressible flow field regardless of the current particle density and proposed solving a Poisson equation for the pressure, targeting the removal of any divergence in the intermediate velocity field exactly as in grid-based methods. A similar projection procedure which uses pressure to make the velocity field divergence free was proposed for SPH simulation in [16] (see also [15]).

Later, [65] realized that it is desirable to both have a divergence free flow field and provide a mechanism for number density targeting. They first solve a Poisson equation for pressure to obtain a divergence free velocity field. Then, in order to target the desired particle number density, they solve a second Poisson equation to artificially alter the particle positions. However, since the velocities derived from this second solve are discarded, the method yields non-physical solutions. Consider, for example, a stationary flow field containing a subregion with particle density lower than the target density. The second Poisson solve will force particles from the higher density region to the lower density region as desired, but simply changing particle positions and ignoring the resulting velocities will leave the initially static velocity field unaltered. This is contrary to the true physical behavior where the subregion lower in density should have induced a flow of material into it.

As we show below, the Navier-Stokes equations allow for both the enforcement of incompressibility and the targeting of particle number density within a single Poisson solve, yielding the physically correct solution. This is clearly less expensive than a method that requires two Poisson solves. The equation for conservation of mass is

\[
\frac{1}{\rho} \frac{D\rho}{Dt} + \nabla \cdot \vec{u} = 0
\]

where \( D/Dt = \partial/\partial t + \vec{u} \cdot \nabla \) is the material derivative, and \( \rho \) is the density which can
Figure 5.4: (Top) A thin level set source pouring onto an upturned bowl on a \(150 \times 200 \times 200\) grid. The grid cannot resolve the thin film at the point of impact and catastrophically loses mass. (Middle) The simulation with the removed negative particles visualized. The particles have no notion of volume and thus end up compressed on the bottom right edge of the domain. (Bottom) Our SPH solver treats the removed negative particles as an integral part of the liquid representation, and convincingly conserves volume.

be written in terms of the number of particles per region \(n_r\), the mass per particle \(m_p\), and the volume per region \(V_r\) as \(\rho = n_r m_p / V_r\). This yields

\[
\frac{1}{n_r} \frac{Dn_r}{Dt} + \nabla \cdot \vec{u} = 0. \tag{5.3}
\]

Taking the divergence of both sides of equation (5.2) gives

\[
\nabla \cdot (\nabla \hat{p} / \rho) = \nabla \cdot \vec{u}^* - \nabla \cdot \vec{u}^{k+1}. \]

Substituting the value of \(\nabla \cdot \vec{u}^{k+1}\) from equation (5.3) yields

\[
\nabla \cdot (\nabla \hat{p} / \rho) = \nabla \cdot \vec{u}^* + \frac{1}{n_r} \frac{Dn_r}{Dt}. \tag{5.4}
\]
CHAPTER 5. COUPLED SPH AND PARTICLE LEVEL SET SIMULATION

When the goal is to obtain a divergence free flow field as in equations (5.1) and (5.2), the divergence of $\vec{u}^{k+1}$ is set identically to zero. However, [28] noted that for certain physical phenomena (such as expansion caused by explosions) the ability to target a nonzero divergence can be useful. The above derivation illustrates that the same concept can be used to target a particle number density by replacing $Dn_r/Dt$ in equation (5.4) with the desired change in particle number density. Since $Dn_r/Dt$ is a material derivative it necessarily includes an advection term of the form $\vec{u} \cdot \nabla$, however our SPH-based solver is in the Lagrangian (rather than Eulerian) frame and therefore implicitly accounts for this term. Thus, we approximate

$$\frac{1}{n_r} \frac{Dn_r}{Dt} = \frac{1}{n_r^{k+1}} \left( \frac{n_r^{k+1} - n_r^k}{\Delta t} \right)$$

to obtain

$$\nabla \cdot (\hat{\nabla} \hat{\rho}/\rho) = \nabla \cdot \vec{u}^* + \frac{1}{n_r^{k+1}} \left( \frac{n_r^{k+1} - n_r^k}{\Delta t} \right) \quad (5.5)$$

Examples of our targeting can be seen in figures 5.1 and 5.2.

If the target number density is too far from the current density, our targeting scheme can introduce large velocities that result in substantial oscillations. In our implementation, we compensate by averaging the target divergence over a small time interval $\Delta \tau$. If we represent the last term in equation (5.5) as $T = \frac{1}{n_r} \frac{\Delta n_r}{\Delta \tau}$, we obtain the integral expression

$$\int_{n_r^k}^{n_{\text{target}}} \frac{dn_r}{n_r} = \int_{t_k}^{t_k + \Delta \tau} T \, dt$$

which can be solved to obtain $T = \frac{1}{\Delta \tau} \ln \left( \frac{2^n_{\text{target}}}{n_r^k} \right)$ replacing the last term in equation (5.5).
5.4 Diffuse SPH

The formulation described above is valid for dense fluid regions, but fails to adequately allow for diffuse behavior such as in spray/air mixtures. In diffuse fluid volumes, it is incorrect to assign a target density at each point in space, since diffuse regions are highly compressible and the distribution of particles is governed primarily by ballistic motion. Thus we modify our divergence formulation to better account for diffuse fluid by clamping the target divergence to be non-negative, which will force overly-dense regions to expand without causing diffuse regions to non-physically contract. However, this solution is not totally satisfactory since it still enforces incompressibility in regions below the target density threshold which is inappropriate for ballistic fluid features. The problem with equation (5.5) for diffuse regions is that one cannot ascertain the desired target density a priori without accounting for the ballistic motion of all the surrounding regions of flow.

To correct this problem, we observe that particles in dense regions should be incompressible, and thus move according to the velocities generated by the Poisson solve. On the other hand, solitary particles that are far from any other fluid feature should follow ballistic trajectories independent of incompressibility. Therefore, our SPH method introduces the notion of particle slip in which the smoothed number density at each particle position determines the degree to which the particle is affected by the Poisson solve. This slipping is highly desirable for scenes with both dense and diffuse particle regions as it allows particles to smoothly transition between incompressible and ballistic behavior in a physical manner.

As in most SPH algorithms, each particle represents a smoothed, radially-symmetric attribute field that distributes its associated quantities in a local neighborhood of influence. For a single particle $p$ with radius $r_p$ and position $x_p$, we define the influence at a point $x$ to be

$$\omega_p(x) = \begin{cases} 
    c(1 - \|x - x_p\|^2/r_p^2) & \text{when } \|x - x_p\|^2 \leq r_p^2 \\
    0 & \text{otherwise}
\end{cases}$$

(5.6)

where $c$ is a normalization constant. Since our method enforces incompressibility in
CHAPTER 5. COUPLED SPH AND PARTICLE LEVEL SET SIMULATION

an efficient fashion using a background MAC grid, we compute the influence of each particle at both cell centers and faces. A different normalization constant is used for each, and cumulative number densities are also calculated separately for cells and faces.

Each time step of our algorithm proceeds as follows. First, we apply gravity (and other body forces) to the particles. Next, we compute cell and face weights (i.e. particle number densities) using the blending kernel given in equation (5.6). Then we rasterize weighted particle velocities onto the faces of the grid, and store a copy of this velocity field for later use. Next we calculate the target divergence for each cell, solve the variable-density Poisson equation (5.5), and update the grid-based velocity field using equation (5.2). We calculate the change in the grid-based velocities and use the FLIP method [136] to compute a candidate change in velocity $\Delta \mathbf{v}$ for each particle by taking a linear combination of nearby face velocity differences weighted by the particle’s influence at each face center. One disadvantage of FLIP is that a cell may contain particles with widely varying velocities and merely mapping $\Delta \mathbf{v}$ back to each particle adds no viscous behavior. If the particle velocities in a cell have high variance, we introduce a weighted average between FLIP and PIC on a per particle basis, since the PIC method will substantially damp outlying particle velocities by forcing them to more closely conform to those computed on the background grid.

Before mapping this change in velocity to the particles we introduce our notion of particle slip. To determine how much of the calculated velocity change to apply for a given particle, we compute a particle slip ratio $s$ as the particle number density at the particle’s position divided by the global incompressibility target density. Then, we update each particle’s velocity $\mathbf{v}_p^{k+1} = \mathbf{v}_p^k + s \Delta \mathbf{v}$ and subsequently update the particle position using this velocity.

One of the more attractive features of this particle slip method is that we can control the degree to which the SPH solver influences our particles simply by adjusting the slip coefficient $s$. Also, we typically maintain a ballistic particle threshold of around 15% of the incompressible target density and remove cells with weights below this level from the Poisson solve altogether. Because of this level of control, our method can be trivially adapted to interact with arbitrary particle systems simply by
Figure 5.5: Two-way coupled SPH and particle level set fluid simulation. The level set is depicted in green. The far left image shows a few negative removed particles generated from the level set that are subsequently simulated with SPH. In the second figure, we have added a source of further SPH particles. After turning off the source in the third figure, we turn on reincorporation of particles in dense regions so that they transition to a level set representation when possible providing for a smoother interface representation.

disabling or scaling back the influence of our solver in regions that are subject to other dominant forces. In this manner, we can take into account spring-based attractive forces, elasticity, or even completely non-physical particle rules.

5.5 Two-way Coupling

In the standard particle level set algorithm, passive marker particles are seeded on both sides of the fluid interface and advected along the fluid flow. In areas where the grid is unable to fully resolve the level set’s behavior, these marker particles will pass from one side of the interface to the other indicating error in the level set representation and prompting a local rebuilding of the level set function with the characteristic information present in these particles. When a particle strays too far
across the interface, it can be removed from the set of interface tracking markers and instead used to represent spray or bubbles depending on whether it is a removed water or air particle, respectively. We use the removed negative particles that were originally on the interior of the fluid volume to seed our SPH algorithm (although other particles can be introduced as well).

For simulations with sparse particles or those in which the scale of the negative removed particles is small, it suffices for the fluid to exert force on the particles without the particles affecting the behavior of the fluid. In these cases, we one-way couple
the grid-based solver to the SPH solver by first carrying out all the steps of a normal
grid-based solve, and then using the result to generate boundary conditions for the
SPH solver. In particular, each face that lies along the level set interface is set to a
Neumann condition with the velocity provided by the grid-based solver. This coupling
strategy is particularly convenient for adding detail to existing simulations, since it
can be run entirely as a post-process. When employed simultaneously with the fluid
solver, however, we can improve the visual quality of the simulation by reincorporating
particles that penetrate the level set surface and applying a local momentum
conservation force to slightly alter the level set velocities in reincorporation regions.
Figure 5.4 shows a thin level set source impacting a tilted bowl. At the point of
contact, the grid resolution is insufficient to resolve the thin liquid sheet resulting in
complete mass loss (top). In the middle row, we see the same simulation visualizing
the removed negative particles generated by the level set following their initial bal-
listic trajectories. These particles provide better visual cues, but fail to exhibit any
fluid-like behavior. In the bottom row, we use the removed negative particles in a
one-way coupled SPH simulation with improved results.

For simulations with dense particle regions, full two-way coupling is desirable.
Thus, we use a single Poisson solve to compute the divergence free flow field simul-
taneously for the grid-based level set fluid volume and the SPH-governed regions.
First we carry out all the non-projection steps of the grid-based particle level set
solver. Then for faces near the interface that have both valid SPH particle and valid
grid-based level set velocities, we discard the particle velocity and instead use only
the grid-based velocity for the Poisson solve. Cells inside the level set volume have
their divergence set to zero, whereas SPH cells outside the fluid volume have their
divergence set as described above (section 5.3). After projection and the subsequent
updating of the velocity field according to equation (5.2), the velocity is mapped back
to the particles incorporating any desired slip. As in [21] the last step in the particle
level set algorithm is to extrapolate velocities from the valid fluid volume to a band of
surrounding air cells. However if particles occupy part of this region, their velocities
should not be overwritten. Therefore we mark the cells in the grid-based air volumes
where the particles provide an adequate external velocity field and only extrapolate to
cells that do not contain a sufficient number of particles. Figure 5.5 shows a two-way coupled simulation of a particle-only SPH source interacting with a particle level set source. In this simulation, the particles are seeded at the target density and thus are given roughly equal weight to the fluid.

For added efficiency and surface smoothness, one can optionally convert SPH particles back to a grid-based level set representation in areas with sufficient particle density. This is accomplished by defining a level set around each particle in exactly the same manner as is done for the marker particles in the particle level set method. This computes new values for the level set function whereas new velocity values are defined directly from the smooth particle kernel (see rightmost image in figure 5.5).

5.6 Examples

Our three-dimensional examples were carried out on a number of 4 processor Opteron machines and averaged between 30 seconds and 3 minutes per frame. In figure 5.6, we apply our SPH method to an ocean scene with crashing waves on a $560 \times 120 \times 320$ grid. A number of authors have considered using the three-dimensional Navier-Stokes equations to simulate large ocean views, but it has proven quite challenging to convincingly convey the appropriate sense of scale without resorting to non-physical post-processes. We use full two-way coupling with the negative removed particles generating convincing SPH-simulated spray. We also run a secondary one-way coupled simulation of the air to generate fine-detail mist and foam that is sourced from the spray particles along the lines of [69]. Positive removed particles are also passively advected and used to represent bubbles. To generate good initial conditions for our waves, we use the same wave formulation as [76]. We rendered this scene in Pixar’s RenderMan with a deep-water texture applied to the surface. We also consider the simulated pouring of water into a glass as in [21]. Figure 5.3 shows the results obtained using our two-way coupled variable-density SPH solver with particle to fluid conversion enabled on a $120 \times 240 \times 120$ grid. The particles provide convincing splashes and add turbulence to the fluid surface resulting in a much more realistic simulation.
5.7 Conclusions

We proposed a novel SPH solver which allows us to enforce incompressibility in an efficient fashion similar to standard grid-based methods as well as target arbitrary particle number densities with a single Poisson solve. We introduced the notion of particle slip in order to extend this SPH solver to simulate diffuse phenomena such as mixtures of spray and air. Finally we showed how to two-way couple our new SPH solver with a standard particle level set method and illustrated the efficacy of our approach with a number of examples including the crashing of ocean waves against a lighthouse and beach.
Chapter 6

Conclusions and Future Work

This thesis has presented a number of novel algorithms to improve the simulation of fluids for computer graphics. The algorithms presented have allowed us to simulate new phenomena such as melting and burning of solids into liquids and gasses, multiple discrete fluids interacting in a fully coupled manner and smoothed particle hydrodynamics based fluids coupled with the particle level set method. We have also shown how to simulate both single phase fluids and free surface flows on adaptive grids allowing simulations to run faster in a smaller memory footprint.

Although fluid dynamics in computer graphics have advanced significantly in the last few years, it is also obvious that there is much work left to be done. The two fundamental problems that computer graphics researchers interested fluid dynamics will always be able to work on are the two that this thesis has focused on: Efficiency and fidelity. With today’s computer technology, we are still many orders of magnitude away from being able to simulate interesting phenomena at a resolution that allows the features that we are interested in to be resolved. In my experience, I estimate that our largest simulations would need another order of magnitude resolution in each dimension to truly capture the detail that we are interested in, and complete in one tenth of the time to allow artists to iterate efficiently. The amount of time that it will take for computers to become fast enough to allow this will take a long time, which suggests that fundamental algorithmic improvements rather than pure computational power will save the day.
6.1 Simulation of Fluids on Octree Grids

An implementation of a fluid solver requires a number of different ways of fetching data from its data representation. Although there are many different implementations of octree data structures, the cache behavior is always significantly worse than uniform grids for some of these data access patterns. Although we have tried several different techniques, we have had to rely on heavy coarsening to reduce the number of computational cells in the domain to be competitive with uniform grids. A new octree implementation that supports the data access patterns of a fluid solver at a comparable speed to uniform grids would allow us to use the algorithms in this thesis at much higher resolutions.

In order to make an adaptive fluid solver efficient, we have had to rely on good refinement criteria. We have been very aggressive when it comes to coarsening the grid, and the criteria or heuristics for doing so have been very ad-hoc. More research into the types of refinement criteria that are best for computer graphics is needed to get the most out of current adaptive techniques.

6.2 Multiple Interacting Liquids

The ability to simulate multiple interacting fluids allows us to simulate phenomena that are significantly more complex than previously possible. An example is the mushrooming of one liquid into another when the two liquids are of similar density (this can be seen in Figure 3.5 which results in very thin persistent filaments. Even though our multiple particle level set method exhibits the same mass loss as the regular particle level set method, the simulations that are now possible suffer from significantly more mass loss than previous examples.

The multiphase framework supports one fluid turning into another in the same fashion as [89], and we have shown that combining two fluids to create a third allows for really interesting simulations (Figure 3.10). The reaction between the two liquids to create a third was done in a very ad hoc manner: The system detects the proximity of the two liquids, and allowed them to be turned into the third fluid. Although this
was meant model the region where two reactants mix and turn into the product, it was not designed to more accurate than required to create pretty images. The multiple interacting liquids framework should be suitable for modeling and visualizing a large number of chemical reactions on a large scale.

6.3 Melting and Burning

The particles in the particle level set method are not meant to represent mass. They are mass-less marker particles that perform best when there they overlap as much as possible. Typical velocity fields used with the particle level set method are divergence free, which means that as the particles are evolved in time, they should not gain or lose mass. In the case of particles being used in the melting framework that we have proposed, there is no such velocity field when particles escape from the solid and turn into fluid (since they may be outside of the water). A method that better conserves mass for escaped particles such as the two way coupled method in Chapter 5 would alleviate this problem.

The burning paper in Figures 4.1, 4.12 and 4.14 used several heuristics to determine browning and combustion of the solid. A physically accurate method to determine these would undoubtedly lead to even better results.

6.4 Large Scale Simulation

Large scale water scenes (such as oceans) are extremely common in the visual effects industry. These types of scenes present very different challenges from the typical small scale examples seen in most research papers. The transition between dense liquid, spray, mist and foam is a challenging problem both for rendering and simulation. The coupled approach in Chapter 5 attempts to handle the transition between dense liquid and spray from a simulation perspective, but the other effects mentioned and the rendering remains an open and challenging problem.
Bibliography


[72] A. McNamara, A. Treuille, Z. Popović, and J. Stam. Fluid control using the


[74] Z. Melek and J. Keyser. Multi-representation interaction for physically based

[75] B. Merriman, J. Bence, and S. Osher. Motion of multiple junctions: A level set

[76] V. Mihalef, D. Metaxas, and M. Sussman. Animation and control of breaking

[77] G. Miller and A. Pearce. Globular dynamics: A connected particle system for

on computational fluid dynamics. Proc. Eurographics 2002 Short Presentation,

mesh topology during simulation. ACM Trans. Graph. (SIGGRAPH Proc.),

[80] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green
strategy for meshing highly deformable objects with tetrahedra. In 12th Int.


[82] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for
interactive applications. In Proc. of the 2003 ACM SIGGRAPH/Eurographics


simulation of fluids. In Comp. Graph. Forum (Eurographics Proc.), volume 22, 

[97] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, 

large scale phenomena. ACM Trans. Graph. (SIGGRAPH Proc.), 22:703–707, 
2003.

[99] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy 
376, 1983.

[100] O. Roussel, K. Schneider, A. Tsigulin, and H. Bockhorn. A conservative 


New York, NY.


