Surface Representation of Particle Based Fluids

by

Jihun Yu

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2011

Professor Chee K. Yap

To my mother, Insuk Lee, who made me what I am today.

Acknowledgements

I would like to thank my mother, Insuk, and my wife, Mijung, for their constant belief and support, and for creating an environment that encouraged learning and achievement.

I would like to thank to my collaborators, Greg Turk and Chris Wojtan, without whose help this dissertation would not exist.

I am also grateful for the members of the Geometry group at Georgia Tech, especially Karthik Raveendran, Jie Tan, Yuting Ye and Sumit Jain for their willingness to always help me during the intense conference deadline periods.

I would like to thank my good friends, Joonho Baek and Sehoon Ha. They always welcomed and accommodated me during my visit to Atlanta.

I wish to particularly thank Greg Turk, who invited me to the GVU group of Georgia Tech as a visiting member. I feel truly fortunate to collaborate with Greg upon my exploration to the world of computer animation.

Most of all, I would like to thank my advisor, Professor Chee Yap for his remarkable support and patience. Chee always allowed me to pursue my own interest, and guided me with his unique and fresh perspective. The last six years I spent as his apprentice was the most wonderful and fruitful period in my life.

Abstract

In this thesis, we focus on surface representations for particle-based fluid simulators such as Smoothed Particle Hydrodynamics (SPH). We first present a new surface reconstruction algorithm that formulates the implicit function as a sum of anisotropic smoothing kernels. The direction of anisotropy at a particle is determined by performing Weighted Principal Component Analysis (WPCA) over the neighboring particles. In addition, we perform a smoothing step that re-positions the centers of these smoothing kernels. Since these anisotropic smoothing kernels capture the local particle distributions more accurately, our method has advantages over existing methods in representing smooth surfaces, thin streams and sharp features of fluids. This method is fast, easy to implement, and the results demonstrate a significant improvement in the quality of reconstructed surfaces as compared to existing methods. Next, we introduce the idea of using an explicit triangle mesh to track the air/liquid interface in a SPH simulator. Once an initial surface mesh is created, this mesh is carried forward in time using nearby particle velocities to advect the mesh vertices. The mesh connectivity remains mostly unchanged across time-steps; it is only modified locally for topology change events or for the improvement of triangle quality. In order to ensure that the surface mesh does not diverge from the underlying particle simulation, we periodically project the mesh surface onto an implicit surface defined by the physics simulation. The mesh surface presents several advantages over previous SPH surface tracking techniques: Our method for surface tension calculations clearly outperforms the state of the art in SPH surface tension for computer graphics. Our method for tracking detailed surface information (like colors) is less susceptible to numerical diffusion than competing techniques. Finally, a temporally-coherent surface mesh allows us

to simulate high-resolution surface wave dynamics without being limited by the particle resolution of the SPH simulation.

Contents

Dedication iii

Acknowledgements iv

Abstract v

List of Figures x

List of Tables xiii

List of Appendices xiv

1 Introduction **1**

1.1 Surface Reconstruction for Particle Based Fluids Using Anisotropic Kernels 3

1.2 Surface Tracking for Particle Based Fluids With an Explicit Surface Mesh 4

1.3 Contributions 7

2 Previous Work **9**

2.1 Surface Extraction 9

2.2 Surface Tracking 14

2.3 Surface Tension Models 16

2.4 Small Scale Surface Dynamics 17

3 SPH Fluid Simulator **18**

3.1 Eulerian and Lagrangian Viewpoint of the Material Derivative . . . 18

3.2 Formulation and Algorithm of Smoothed Particle Hydrodynamics . 22

4 Surface Extraction of Particle Based Fluids Using Anisotropic Kernels **28**

4.1 Implicit Surface Representation 28

4.2 Determining the Anisotropy 31

4.3 Alleviating Attraction Artifacts 35

4.4 Implementation Overview 37

4.5 Optimization of Performance 39

4.6 Results 40

4.7 Comparison 45

4.8 Limitations 49

4.9 Conclusion and Future Work 49

5 Explicit Mesh Surface for Particle Based Fluids **51**

5.1 Mesh Advection and Topology Changes 51

5.2 Surface Property Advection 58

5.3 Surface Tension Models 59

5.4 Small Scale Surface Dynamics 61

5.5 Implementation Overview 64

5.6 Results 65

5.7 Limitations 73

5.8 Conclusion and Future Work 73

Appendices **75**

Bibliography **83**

List of Figures

1.1 Interactive double dam break (Image courtesy of Simon Green). Left: Visualization of anisotropic particles. Right: Extracted Surface. 4

1.2 Left image is original water crown. Right is water crown that has been augmented with capillary waves on the surface mesh as a post-process. 6

2.1 Comparison between different surface reconstruction approaches on the Double dam break animation. 12

3.1 Illustration of the Lagrangian form of the material derivative. Particle p flows with the background velocity field (diagonal arrows) and a reference frame moves along with the particle. The change of the quantity ϕ is represented by the change of color of the particle. As we sit on the particle, only the change on particle p is observed. 20

3.2 Illustration of the Eulerian form of the material derivative. A reference frame is fixed in space while particles flow with the background velocity field. In contrast to the Lagrangian form, both the advection and the material derivative influence our observation to the change of the quantity. 21

4.1 A comparison between the surface reconstruction using isotropic kernels (a) and our anisotropic kernels (b). Top row: the surface of SPH particles from a single dam break simulation. Bottom row: Illustration of particles at the top right corner. The shape of a particle in (b) represents the anisotropy of the corresponding smoothing kernel. Note that our approach constructs a flat surface with sharp edges and corners from properly stretched particles. 32

4.2 Merging spheres. Top: Not accounting for components. Bottom: Correction using connected components. 36

4.3 Water splash. Top: Anisotropic kernels, Middle: Opaque surface, Bottom: Transparent surface 42

4.4 Melted chocolate falling on a Bunny. 43

4.5 Falling armadillo and bunny. 44

4.6 Double dam break simulation. 46

4.7 Interactive Double Dam Break 47

5.1 Replication of the ring edges. Left:A red colored edge is flagged for the collapse operation. Right: Instead of collapsing the edge, the mesh is separated by replicating the ring edges into the blue colored triangles. 57

5.2 A dumbbell in zero gravity exhibits pinch-off due to surface tension effects (Rayleigh-Plateau instability). Later in this animation the separate components re-join each other. 66
5.3 Three viscous figures with surface colors are dropped on a bar. . . . 68
5.4 Comparison between different surface color tracking approaches on the viscous figures animation. 69
5.5 Two spheres in zero gravity that merge and exhibit surface waves. . 70
5.6 A cubic water drop that oscillates and settles into a sphere. 70
5.7 A drop falling into a shallow pool creates a water crown. 72

List of Tables

4.1 Average per frame timings (in minutes) for different surface reconstruction methods on the falling figures example of Figure 4.5. . . . 43

4.2 Average per frame timings (in minutes) for four surface reconstruction methods on the double dam break simulation. 48

5.1 Mesh resolution relative to the average particle spacing and average per frame timings (in seconds) for our simulation examples. 71

List of Appendices

A Analysis on Volume Shrinkage **75**

B Source Code **77**

Chapter 1

Introduction

It is increasingly popular to create animated liquids using physics-based simulation methods for feature film effects and interactive applications. There exist two broad categories for simulation methods based on their different approaches to spatial discretization: mesh-based (Eulerian) methods and particle-based (Lagrangian) methods. In mesh-based methods, the simulation domain is discretized into mesh grids and the values of physical properties on grid points are determined by solving the governing equations. In particle-based methods, on the other hand, the fluid volume is discretized into sampled particles that carry physical properties and that are advected through space by the governing equations. In recent years, particle-based methods have become a competitive alternative to mesh-based methods due to various advantages such as their inherent mass conservation, the flexibility of simulation in unbounded domains, capability of capturing small scale features, and ease of implementation. Several particle-based methods such as Smoothed Particle Hydrodynamics (SPH), Particle-in-cell (PIC), and Fluid-implicit-particle (FLIP) are popularly employed in computer animation field for

the simulation of various fluid phenomena.

Although particle-based methods have been used to simulate various fluid phenomena, extracting high quality fluid surfaces from particle samples is not straightforward. In Eulerian methods, a surface is well defined as an iso-surface on the grid structure and is easily extracted using methods like Marching Cubes or ray-tracing. In contrast, reconstructing a surface from the particle samples is an entire research area and the shape of the surface is not uniquely defined by the particle samples. Classical surface reconstruction methods have difficulty in producing smooth surfaces due to irregularly placed particles. Few researchers have successfully addressed this issue of reconstructing smooth fluid surfaces from particles.

The aim of this thesis is to explore the potential gains in surface quality and visual fidelity of particle-based fluids when we introduce a new surface representation. This dissertation has two main research contributions: the smooth surface reconstruction of particle-based fluids presented in Chapter 4, and the surface tracking of particle-based fluids using the explicit surface mesh presented in Chapter 5. In particular, we define a novel implicit surface from particle samples using anisotropic kernels. Embedding anisotropic information, we have seen vast gains in surface smoothness and visible details (Chapter 4.6). Furthermore, we introduce an explicit surface tracking method for particle-based fluids. Tracking surfaces using an explicit mesh provides benefits by augmenting the surfaces with visual details such as colors, surface tension and small scale capillary waves (Chapter 5.6).

Chapter 2 will cover previous work in fluids and the background material necessary to understand research about the fluid surfaces. Chapter 3 will cover the concept of the material derivative, and it will cover the formulation and the implementation of SPH simulator for computer graphics. All our running examples

are produced based on SPH simulations.

1.1 Surface Reconstruction for Particle Based Fluids Using Anisotropic Kernels

In Chapter 4, we present a novel surface extraction method that significantly improves the quality of the reconstructed surfaces by using ellipsoidal smoothing kernels. The new method can create smooth surfaces and thin streams while preserving sharp features such as edges and corners (Figure 1.1). The key to the method is to use a stretched, anisotropic smoothing kernel to represent each particle in the simulation. The orientation and scale of the anisotropy is determined by capturing each particle's neighborhood spatial distribution. The neighborhood distribution is obtained in the form of a covariance tensor that is given through Principle Component Analysis (PCA). Then these principal components are used to orient and scale the anisotropic kernel. The centers of these kernels are adjusted using a variant of Laplacian smoothing to counteract the irregular placement of particles. A new density field is then constructed by the weighted mass contribution from the smoothing kernels. Finally, the renderable surface is reconstructed from the iso-surface of the given density field.

The method that we present allows us to extract smooth surface meshes from a newly defined implicit surface. Other published methods for particle-based fluids in the graphics literature simply cannot represent smooth surfaces [Blinn, 1982], or they require to apply an additional smoothing step on the implicit surface or the extracted mesh [Zhu and Bridson, 2005; Williams, 2008; Sin *et al.*, 2009].

Our new method has been well received in the graphics industry. Numerous

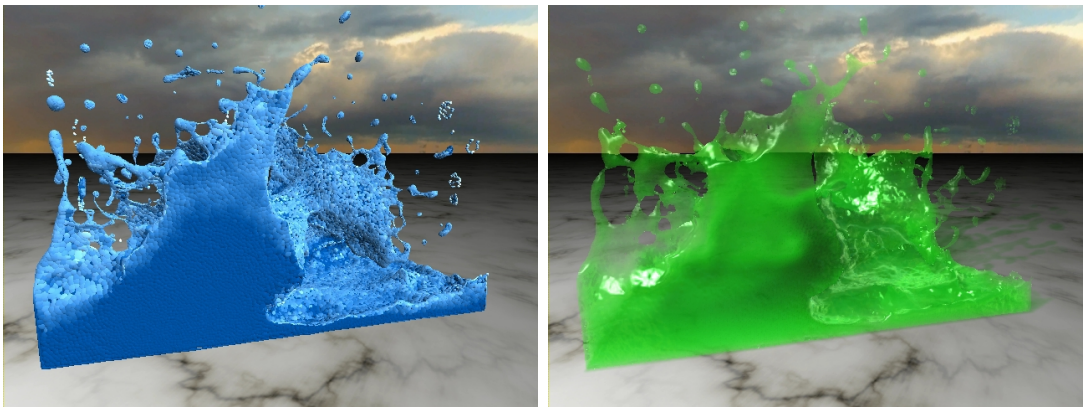


Figure 1.1: Interactive double dam break (Image courtesy of Simon Green). Left: Visualization of anisotropic particles. Right: Extracted Surface.

production studios have asked us to share our implementation code and our method is also available in a commercial software package FROST (©2011 ThinkSoft). We expect our method to be added into many production toolboxes where it will complement other surface extraction methods that are currently available.

1.2 Surface Tracking for Particle Based Fluids With an Explicit Surface Mesh

In Chapter 5, we introduce a new *explicit* method for tracking the surface of a particle-based fluid simulation. "Explicit" means that we maintain surface meshes that are advected by the velocity flows, while implicit methods extract surface meshes from the iso-surface of an implicit function. Our method builds the idea of reconstructing fluid surfaces using anisotropic kernels. All the published methods for extracting surfaces from particle-based fluids, including our anisotropic kernel method, provide implicit representations for the fluid surfaces. However, they do not give us any flow information from one surface to the same surface at a

later time instance. In the computer animation field, surface representation is considered more important than the computational accuracy because the emphasis is primarily on visual fidelity. In this context, surface flow (surface tracking) is a very attractive concept because it allows us to track various quantities such as colors, textures or bump maps on the time evolving surface. Also, embedding a high resolution surface simulation in the lower resolution fluid simulation leads to detailed surface animation like capillary waves (Figure 1.2). By concentrating computation resources only on the 2D surfaces rather than on the entire 3D volume, it becomes possible to augment realistic visual features on the surfaces without imposing a significant computational overhead. In recent years, several surface tracking methods have been introduced into the field. In particular, the explicit mesh tracking methods equipped with topology changes have been successfully applied to the Eulerian fluid simulation [Wojtan and Turk, 2008; Müller, 2009; Wojtan *et al.*, 2009; Brochu *et al.*, 2010; Thürey *et al.*, 2010] and their results demonstrate that the explicit mesh well preserves thin and sharp features and surface wave dynamics can be separately simulated on the mesh.

Our surface tracking approach is the first mesh tracker for a particle-based fluid. We begin by constructing a surface mesh for the fluid at the first time step in the simulation. This initial mesh is an isosurface given by our anisotropic kernel surface representation. For subsequent simulation time steps, the old mesh vertices are advected using the velocity of the fluid particles. Then the new vertex locations are projected back onto the implicit surface in order to alleviate numerical drift, ensuring that accumulating numerical errors do not cause the surface to diverge from the simulation particles. Where the mesh is stretched or compressed, we perform edge splits or collapses in order to improve the triangle shapes, and when

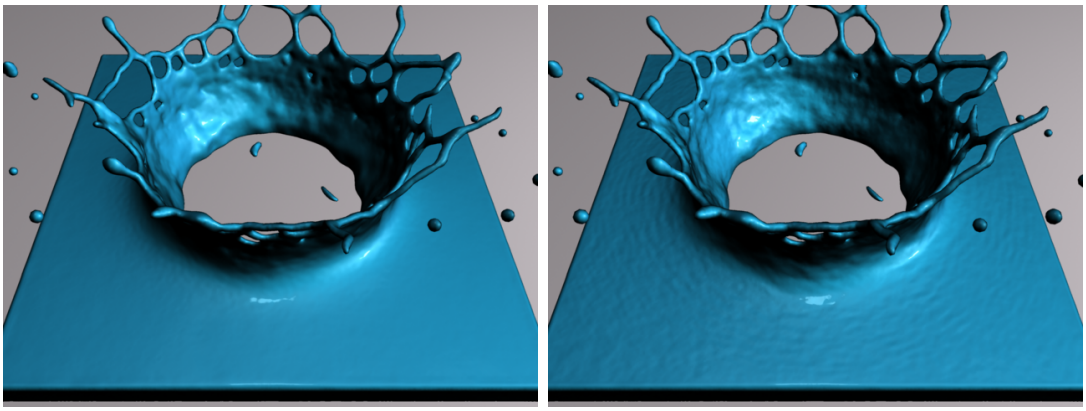


Figure 1.2: Left image is original water crown. Right is water crown that has been augmented with capillary waves on the surface mesh as a post-process.

the fluid surface splits or merges we use a robust technique for changing the mesh topology. Here we follow ideas from [Wojtan *et al.*, 2009].

There are several advantages to carrying a mesh between time steps instead of reconstructing a new mesh for each frame. Our method allows us to calculate surface tension forces in a stable manner from the mesh itself, and propagate these forces back to the particles. We can carry surface properties such as color along with the mesh in a manner that retains a high level of detail. A final advantage that we demonstrate is to simulate high-resolution surface wave dynamics on the mesh using vertical displacements that are carried between time steps.

Resolution mismatch between the surface mesh and the physics simulation often results in surface artifacts such as surface kinks and small voids. In particular, the low resolution physics simulation does not capture high resolution surface features because it cannot see anything below its resolution unit. Other published methods resolve this issue by re-sampling the simulation domain. Then the surface mesh influences back to the physics simulation. In contrast, we project the explicit mesh

onto the implicit surface in order to prevent the mesh drifting away from particle samples. By performing projection, our approach misses out on a major advantage of the explicit mesh tracking that it preserves thin and sharp features. However, the projection step enables us to apply the surface tracking as a post-process for the already simulated animation sequences because our tracking approach does not influence the simulation.

1.3 Contributions

The proposed surface tracking approaches discussed in this dissertation provides several advantages over previous approaches in the computer animation field. The contributions of our thesis are as follows:

- **Smooth surface reconstruction:** We present a novel surface reconstruction method for particle-based fluids by repositioning particle locations and deforming isotropic kernels to anisotropic kernels. Our approach significantly reduces surface noise and produces unprecedented high quality fluid surfaces compared to previous approaches.

- **Explicit mesh tracking:** We introduce the first *mesh-based* surface tracker for a particle-based simulation. The explicit mesh is advected by the background particle simulation and projected onto the iso-surface defined by the particle samples to prevent the mesh diverging from the particle locations. Our surface tracker does not influence the physical simulation and can be easily applied to an already-compute simulation as a post-process.

- **Surface property tracking:** Our mesh-based surface tracker allows for

accurate tracking of surface data like colors and textures. Our method is virtually free of common artifacts such as numerical diffusion or irregular sampling by carrying surface data on the mesh vertices along the simulation.

- **Accurate surface tension:** We present a new surface tension model that clearly outperforms the state of the art in SPH for computer graphics. The explicit mesh representation allows us to directly apply surface tension forces by exploiting the cotangent based mean curvature computation on the mesh.

- **Post-processed capillary waves:** We introduce a new method for adding subtle ripple details to an animation that has *already been simulated* — this is first technique that adds surface tension dynamics as a post-process to an already-computed simulation. Our non-linear capillary waves model produces small scale waves at varying speeds.

Chapter 2

Previous Work

2.1 Surface Extraction

Because the representation of a fluid surface is crucial for realistic animation, methods for reconstructing and tracking fluid surfaces have been a topic of research ever since fluid simulation was first introduced in computer graphics. In mesh-based frameworks, numerous methods for surface representation have been been proposed. The levelset method [Osher and Sethian, 1988] has been successfully applied to track the free surface of a fluid in an Eulerian simulation scheme. They evolve an implicit signed distance field by advection and sample the field on the Eulerian grid. To overcome volume loss and the blurring of features of the basic levelset method, Enright et al. [2002; 2005] introduced the particle levelset method that advects explicit particles along with an implicit signed distance function. Lossaso et al. [2004] proposed an adaptive simulation method that captures fine surface details by using an octree data structure to increase the simulation resolution. Bargteil et al. [2005] introduced the semi-Lagrangian contouring method.

They create a new mesh by advecting a signed distance field to maintain a more accurate surface representation, and their method can track free surfaces of liquid along with surface properties such as colors and texture maps.

In the particle-based framework, Blinn [1982] introduced the classic blobby spheres approach. In this method, an iso-surface S is extracted from a scalar field $\phi(\mathbf{x})$:

$$S = \{\mathbf{x} | \phi(\mathbf{x}) = \alpha\}, \tag{2.1}$$

$$\phi(\mathbf{x}) = \sum_i w_i(\mathbf{x}), \tag{2.2}$$

that is constructed from a sum of a radial basis kernel P that is placed at each particle center $\mathbf{x}_i$:

$$w_i(\mathbf{x}) = \sum_i P(\|\mathbf{x} - \mathbf{x}_i\|). \tag{2.3}$$

Typical value of α is 0 or a small positive value in order to obtain some smoothing effect.

One of the drawbacks of Blinn's original formulation is that high or low densities of particles will cause bumps or indentations on the surface (top left of Figure 2.1). Noting this problem, Zhu and Bridson [2005] modified this basic algorithm to compensate for local particle density variations. They calculate a scalar field from the particle positions that is much like a radial basis function that is centered at a particle. For arbitrary location $\mathbf{x}$, they calculate a scalar value from a basis function whose center is a weighted sum of nearby particle centers $\mathbf{x}_i$'s, and whose radius is a weighted sum of particle radii r_i's:

$$\phi(\mathbf{x}) = \|\mathbf{x} - \bar{\mathbf{x}}\| - \bar{d}, \tag{2.4}$$

$$\bar{\mathbf{x}} = \frac{\sum_i w_i(\mathbf{x})\mathbf{x}_i}{\sum w_i(\mathbf{x})}, \tag{2.5}$$

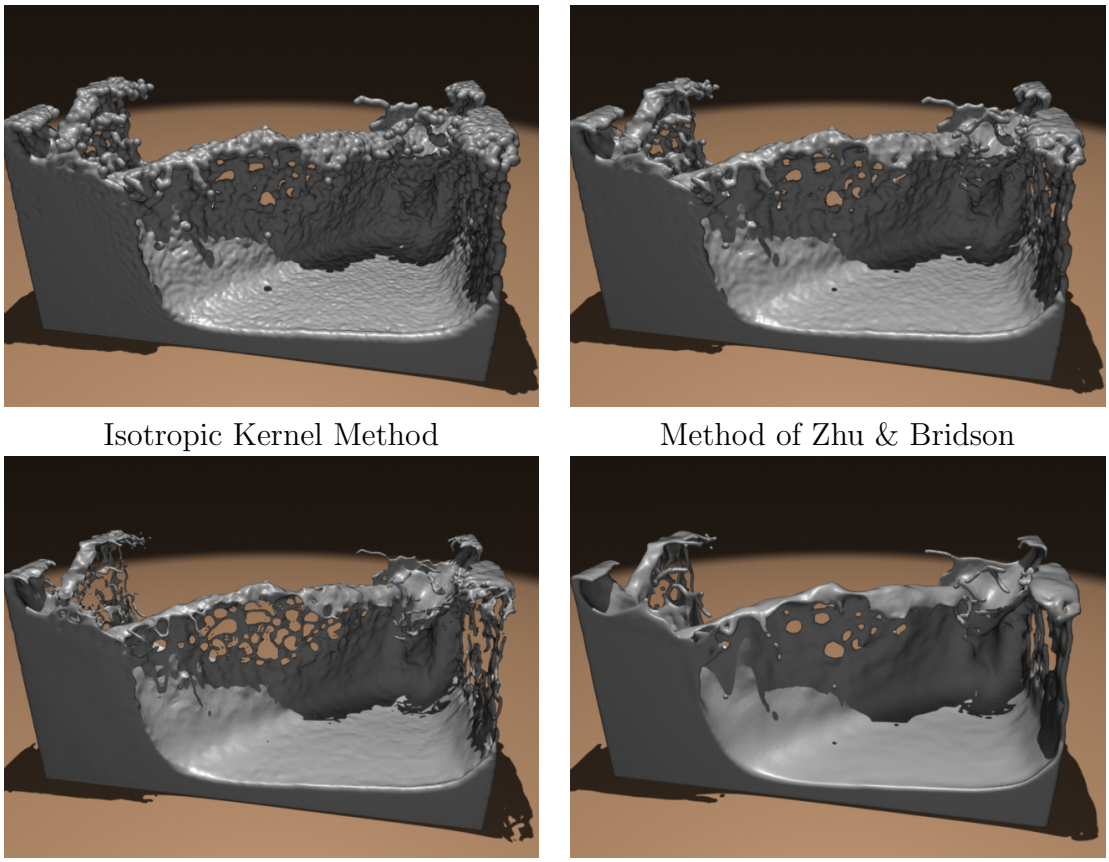
$$\bar{d} = \frac{\sum_i w_i(\mathbf{x})r_i}{\sum w_i(r_i)}. \tag{2.6}$$

They then sample this scalar distance function on a grid, perform a smoothing pass over the grid, and then extract an isosurface mesh from the grid. Their results are considerably smoother than the classic blobby spheres surface (top right of Figure 2.1). Adams et al. [2007] further improved upon the method of Zhu and Bridson by tracking the particle-to-surface distances d_i's over time and using them instead of the static particle radii:

$$\bar{d} = \frac{\sum_i w_i(\mathbf{x})d_i}{\sum w_i(\mathbf{x})}. \tag{2.7}$$

Specifically, they retain a sampled version of a signed distance field at each time step, and they use this to adjust per-particle distances to the surface. They perform particle redistancing by propagating the distance information from surface particles to interior particles using a fast marching scheme. The final surface is from the Zhu and Bridson scalar field, but calculated using these new per-particle distances. This method is successful at generating smooth surfaces both for fixed-radius and adaptively-sized particles (bottom left of Figure 2.1).

One drawback of these aforementioned methods is a typical assumption that the smoothing kernel of each particle is isotropic, and the spherical shape of the kernel makes it difficult to produce flat surfaces and sharp features. In contrast to these methods, our approach uses anisotropic kernels to stretch spheres into



Isotropic Kernel Method | Method of Zhu & Bridson

Method of Adams et al. | Our Anisotropic Kernel Method

Figure 2.1: Comparison between different surface reconstruction approaches on the Double dam break animation.

ellipsoids in order to alleviate those limitations (bottom right of Figure 2.1).

Desbrun and Cani-Gascuel [1998] and Premoze et al. [2003] use a different surface tracking method in which an implicit scalar field is updated by solving the advection equation on a Eulerian grid. Unfortunately this approach is more difficult to use for large scale simulations due to the storage overhead on the large-sized regular grids.

Recently, an alternative method of surface reconstruction was proposed by Williams [2008]. In his method, a nonlinear optimization problem is solved iteratively to achieve global smoothness on surface mesh. The important contribution of the method is that the perfectly flat surfaces can be generated under certain conditions. Williams' method occasionally produces temporally incoherent smoothed meshes because the convergence of smoothing scheme is sensitive to the renewed mesh connectivity between rendering frames. Sin al. [2009] use level-set variants of the original method to alleviate the problem of temporal coherence.

Our anisotropic kernel approach is inspired by the work of Owen et al. [1998] and Liu et al. [2006]. They adapt anisotropic kernels to simulate large deformations of materials in the SPH framework, and their primary interest is in simulation accuracy. In their approach, the axes of their anisotropic kernels evolve in time according to the strain-rate tensor estimates. Our approach is also related to the work of Kalaiah and Varshney [2003] and Dinh et al. [2001]. Kalaiah and Varshney apply PCA to point clouds for point-based modeling. Dinh et al. reconstruct surfaces from voxel carving data by combining anisotropic kernels with variational implicit surfaces.

2.2 Surface Tracking

In this section, we focus on previous methods for tracking surfaces of the fluid simulation. In computer graphics field, **surface tracking** and **surface extraction** are similar terms referring to the methods that captures 2D the surface geometry from the fluid volume. In this thesis, we use **surface tracking** for the methods which track the geometry of time evolving surfaces along with associated surface properties.

A large body of techniques in surface tracking have been developed by researchers in recent years. In Eulerian simulation, an implicit surface is sampled on the discrete simulation grid and is evolved in time. This includes the levelset method [Osher and Sethian, 1988], the particle levelset method [Enright *et al.*, 2002; Enright *et al.*, 2005], the adaptive levelset method [Losasso *et al.*, 2004] and semi-Lagrangian contouring method [Bargteil *et al.*, 2005].

While these implicit surface techniques easily handle changes in surface topology, the size of the features that they can represent is limited by a grid resolution. To alleviate this limitation, researchers have turned to explicit tracking methods that maintain and advect a triangle mesh along with the fluid velocity field. The method presented by Müller [2009] globally re-samples the surface using a marching cubes grid, but then retains previous mesh samples in order to preserve fine surface features. Brochu and Bridson [2009] use a mesh surgery technique that is purely based on geometric intersections. They later combined this with an Eulerian simulation in which pressure samples are placed in order to capture fine surface geometry [Brochu *et al.*, 2010]. To handle changes in topology when using a mesh surface representation, Du et al. [2006] and Wojtan et al. [2009] use local re-meshing techniques in regions where the mesh topology differs from that of an

isosurface representation of the fluid interface. In [Wojtan *et al.*, 2010], a local convex hull procedure is used for local re-meshing to preserve thin fluid features. The mesh-based surface tracking in our research makes use of techniques from [Wojtan *et al.*, 2009] and [Wojtan *et al.*, 2010].

Surface tracking is difficult in particle-based fluid simulation schemes because particles do not retain any connectivity information. As mentioned in previous section, researchers have developed a number of methods for reconstructing surfaces from such particle simulations : The blobby sphere approach [Blinn, 1982], the method of Desbrun and Cani [1998], the method of Zhu and Bridson [2005], and the method of Adams et al. [2007], the mesh smoothing approach [Williams, 2008], the levelset smoothing approach [Sin *et al.*, 2009] and anisotropic surface extraction approach [Yu and Turk, 2010].

One limitation of these aforementioned surface reconstruction methods for particle based fluids is that carrying surface properties on the surface from one frame to a next frame is not straightforward, because a new surface mesh is constructed at each rendering step. In contrast, our approach facilitates the surface tracking by advecting the explicit mesh surface at each frame.

In addition to surface front tracking, special techniques have been developed for tracking surface characteristics such as colors, texture coordinates and physical properties. Texture mapped particles are advected through the fluid velocity field in [Rasmussen *et al.*, 2004], while texture color and local orientation is advected and used as a constraint to synthesize temporally coherent textures in [Kwatra *et al.*, 2007] and [Bargteil *et al.*, 2006]. Bargteil et al. [2006] showed that semi-Lagrangian surface tracking [Bargteil *et al.*, 2005] can be used to track surface properties during a simulation.

2.3 Surface Tension Models

Surface tension forces are responsible for essential details in fluid simulation. For example, the formation and oscillation of a water droplet and ripples on the fluid surface are driven by surface tension forces. Surface tension forces have been applied in a number of Eulerian grid simulation methods. Kang et al. [2000] estimate surface curvature from a levelset function in order to produce surface tension forces. Lossaso et al. [2004] calculated the surface tension forces at free surfaces more accurately by employing an octree structure. Hong and Kim [2005] treat surface tension effects as discontinuous boundary conditions at the interface. Using an explicit surface representation, discrete curvature operators such as that of Desbrun et al. [1999] can be used to compute accurate surface tension forces. Brochu et al. [2010] added a discontinuous pressure jump based on the mean curvature of the surface. The method of Thürey et al. [2010] computes the surface tension forces from a mesh using volume preserving mean curvature flow, and these forces are used as a boundary condition to the grid-based pressure solve.

In particle-based simulations, there have also been several approaches to modeling surface tension. Müller et al. [2003] approximate the surface tension force as the divergence of the surface normal field. Clavet et al. [2005] used a double density relaxation to achieve effects similar to surface tension. Hu and Adams [2006] discussed a surface tension model for a multi-phase SPH simulation, and Becker and Teschner [2007] proposed a molecular cohesive force approach for surface tension effects. Zhang [2010] detects boundary particles and measures the curvature from a local surface representation constructed from moving least squares (MLS).

All previous methods in particle-based simulations use particle samples to generate surface tension effects. Consequently, the particle resolution explicitly limits

the resolution of the surface tension force. Our approach is able to push beyond this limit by combining a standard SPH solver with a detailed explicit surface mesh.

2.4 Small Scale Surface Dynamics

One method of creating finely detailed surfaces at a low computational cost is to couple a high resolution surface representation with a lower resolution solver for the bulk of the fluid. Goktekin et al. [2004] and Kim et al. [2009] couple a high resolution particle levelset with the low resolution fluid solver. Bargtiel et al. [2006] used an octree contouring method that is coupled with the uniform grid fluid solver. Sifakis et al. [2007] embedded high resolution particle samples on the simulation mesh, and a low resolution finite element solver [Bargteil *et al.*, 2007] is coupled with a detailed surface mesh in [Wojtan and Turk, 2008].

Our surface dynamics approach has commonalities with the work of Wang et al. [2007] and Thürey et al. [2010]. Wang et al. solve a general shallow water equation on a surface mesh, while Thürey et al. use a dynamic surface mesh to solve the wave equation. In our approach, capillary waves of variable speed are generated by minimizing thin plate energy. We use a per-vertex displacement along the normal of the surface mesh in order to create surface dynamics.

Chapter 3

SPH Fluid Simulator

3.1 Eulerian and Lagrangian Viewpoint of the Material Derivative

In the SPH simulation scheme, particles are used to represent discretely sampled physical quantities and the Navier stokes equations are numerically solved in Lagrangian way using these samples. In this section, we present the Lagrangian and Eulerian interpretation of the material derivative. Later in Chapter 3.2, the Lagrangian form of the material derivative is used in SPH to solve the Navier stokes equations. A momentum conservation part of the Navier Stokes equations appears as:

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}, \tag{3.1}$$

where ρ is density, $\mathbf{u}$ is velocity, $-\nabla p$ is gradient of pressure, $\nabla \cdot \mathbf{T}$ is divergence of the stress tensor, and $\mathbf{f}$ is an external body force. On the left side of the momentum equation, we encounter the material derivative form $\frac{D(*)}{Dt}$, which describes the time

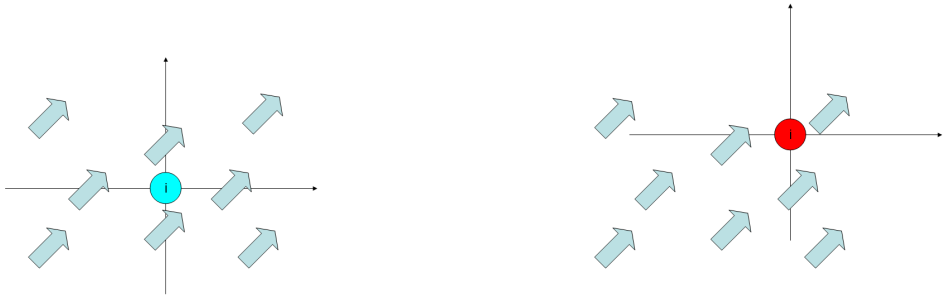
rate of change of some quantity (such as heat or momentum) by following it, while moving with a space and time dependent velocity field. Let's use a particle to represent an infinitesimal fluid element moving with the flow. Suppose that we measure a scalar quantity $\phi(\mathbf{x}, t)$. At time t_1, the quantity of particle i at $\mathbf{x}_1$ is $\phi_1(\mathbf{x}_1, t_1)$. At a later time t_2, particle i has moved to $\mathbf{x}_2$ and has a new value $\phi_2(\mathbf{x}_2, t_2)$. Then, the material derivative measures the instantaneous time rate of change of the quantity, of particle i at time t_1.

$$\left.\frac{D\phi}{Dt}\right|_{t_1} = \lim_{t_2 \to t_1} \frac{\phi_2 - \phi_1}{t_2 - t_1} \tag{3.2}$$

The material derivative can be interpreted using two different viewpoints: Lagrangian and Eulerian. In the Lagrangian viewpoint, the material derivative measures the rate of change ϕ on particle i by using a reference frame whose origin is attached to the particle (Figure 3.1). On this moving reference frame, the particle position is always fixed at the origin ($\mathbf{x}_1 = \mathbf{x}_2 = \mathbf{0}$). Then ϕ of particle i becomes a function of time t and position $\mathbf{x}$ just indicates the position of particle i. Therefore, the rate of change of ϕ is simply a partial derivative of a quantity with respect to time t:

$$\frac{D\phi(\mathbf{x}, t)}{Dt} = \frac{\partial \phi(\mathbf{x}, t)}{\partial t}. \tag{3.3}$$

In the Eulerian viewpoint, on the other hand, we now use a fixed reference frame in space and measure the rate of change of ϕ at position $\mathbf{x}$. As illustrated in Figure 3.2, at time t_1, we have particle i with a quantity $\phi_i(t_1)$ located at $\mathbf{x}$. At a later time t_2, this particle flows away, and there will be a new particle j with a new quantity $\phi_j(t_2)$ at $\mathbf{x}$. Now, the rate of change of ϕ at $\mathbf{x}$ accounts for the two separate factors: One from the particle replacement $i \to j$ due to the flow



at time t_1 at time t_2

Figure 3.1: Illustration of the Lagrangian form of the material derivative. Particle p flows with the background velocity field (diagonal arrows) and a reference frame moves along with the particle. The change of the quantity ϕ is represented by the change of color of the particle. As we sit on the particle, only the change on particle p is observed.

(advection), and the other from the change of the quantity of the replaced particle $\phi_j(t_1) \to \phi_j(t_2)$. By taking infinitesimal time step dt, we can observe that the location of the new particle j at time t is $-\mathbf{u}(\mathbf{x}, t)dt$. Therefore, the rate of change due to the advection factor is

$$\phi(\mathbf{x}, t + dt) = \phi(\mathbf{x} - \mathbf{u}(\mathbf{x}, t)dt, t) = \phi(x, t) - (\mathbf{u}(\mathbf{x}, t)dt) \cdot \nabla\phi(\mathbf{x}, t). \tag{3.4}$$

By a simple rearrangement and substituting dt to ∂t, we obtain

$$\frac{\partial\phi(\mathbf{x}, t)}{\partial t} = -\mathbf{u}(\mathbf{x}, t) \cdot \nabla\phi(\mathbf{x}, t). \tag{3.5}$$

What is the second factor $\phi_j(t) \to \phi_j(t + \delta t)$? By the definition, it is simply the material derivative $\frac{D\phi_j}{Dt}$! We can assume $i = j$ by taking the infinitesimal time step, and by adding this factor to (3.5), we obtain a form for the time rate of change of

at time t_1 at time t_2

Figure 3.2: Illustration of the Eulerian form of the material derivative. A reference frame is fixed in space while particles flow with the background velocity field. In contrast to the Lagrangian form, both the advection and the material derivative influence our observation to the change of the quantity.

ϕ at $\mathbf{x}$:

$$\frac{\partial \phi(\mathbf{x},t)}{\partial t} = -\mathbf{u}(\mathbf{x},t)\cdot\nabla\phi(\mathbf{x},t) + \frac{D\phi(\mathbf{x},t)}{Dt}. \tag{3.6}$$

By a simple rearrangement, the material derivative in the Eulerian viewpoint yields

$$\frac{D\phi(\mathbf{x},t)}{Dt} = \frac{\partial \phi(\mathbf{x},t)}{\partial t} + \mathbf{u}(\mathbf{x},t)\cdot\nabla\phi(\mathbf{x},t). \tag{3.7}$$

Note that (3.7) is augmented with the advection term $\mathbf{u}\cdot\nabla\phi$ compared to (3.3) due to the velocity that we feel by choosing of the fixed reference frame.

The material derivative also can be derived from the total derivative. In fact, (3.2) stands for the total derivative of ϕ with respect to time. The total derivative is expanded through the multivariate chain rule:

$$\frac{D\phi(\mathbf{x},t)}{Dt} = \frac{d\phi(\mathbf{x},t)}{dt} = \frac{\partial \phi(\mathbf{x},t)}{\partial t} + \frac{\partial \mathbf{x}}{\partial t}\cdot\nabla\phi(\mathbf{x},t). \tag{3.8}$$

Let's treat $\mathbf{x}$ as the location of particle i moving with the flow. When we take

the Lagrangian viewpoint by attaching a reference frame to the particle, $\mathbf{x}$ remain constant with respect to time ($\frac{\partial \mathbf{x}}{\partial t} = 0$) and we obtain (3.3). When we take the Eulerian viewpoint, $\mathbf{x}$ does change in time and the rate of change corresponds to the velocity field evaluation at $\mathbf{x}$ that is $\mathbf{u}(\mathbf{x}, t)$. Then we obtain (3.7).

In Eulerian fluid simulation, the Eulerian form (3.7) is discretized on the simulation mesh to solve the Navier stokes equations. For the advection part, several algorithms such as semi-Lagrangian method [Stam, 1999] are used in computer graphics field, but they generally introduce numerical diffusion because they resample the quantity on the mesh at simulation time steps. In particle-base fluids, on the contrary, numerical diffusion does not occur during the advection because particles carry the quantity in a Lagrangian way.

3.2 Formulation and Algorithm of Smoothed Particle Hydrodynamics

SPH was originally developed by Monahan and Gingold [1977], and Lucy [1977] to solve astronomical problems. Because of its efficiency in computation and its advantage of being a fully Lagrangian method, SPH has been widely used for solving applied mechanics problem. In graphics, SPH is a popular method for interactive fluid simulation. In this section we summarize the basic formulation of SPH and its application to fluid dynamics. For comprehensive introduction to SPH and its application, we refer to [Monaghan, 2005; Liu and Liu, 2003]. For application of SPH to fluid simulation in graphics, we refer to [Adams and Wicke, 2009].

The core of SPH is based on a function approximation method (also called Re-

producing Kernel Method) on a finite discrete sampling of points. The derivation of the method starts from the representation of a scalar function f in space in the integral form

$$f(\mathbf{x}) = \int_{\Omega} f(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')d\mathbf{x}', \tag{3.9}$$

where Ω is a given domain and $\delta(\mathbf{x})$ is a Dirac-Delta function. For a finite set of discrete sampling locations $\{\mathbf{x}_j\}$, the above equation gives

$$f(\mathbf{x}_j) = \int_{\Omega} f(\mathbf{x}')\delta(\mathbf{x}_j - \mathbf{x}')d\mathbf{x}'. \tag{3.10}$$

After this spatial discretization over the domain, we still need to discretize the term $\delta(\mathbf{x} - \mathbf{x}')d\mathbf{x}'$ to approximate function values in regions between samples. A good way is to smooth out the function values at each sampling location over the space by replacing the Dirac-Delta function with a smoothing kernels $W(\mathbf{x}, h)$ and discretizing $d\mathbf{x}'$ by V_j as a volume at $\mathbf{x}_j$. For any scalar or vector function f, denote the spatial discretization of f at $\mathbf{x}$ by $\langle f \rangle(\mathbf{x})$[1].

$$\langle f \rangle(\mathbf{x}) = \sum_j f(\mathbf{x}_j)W(\mathbf{x} - \mathbf{x}_j, h)V_j, \tag{3.11}$$

where W is commonly chosen to be a isotropic smoothing kernel of finite support with the support radius h for computational efficiency. In SPH for physics simulations, physical quantities such as density and pressure are irregularly sampled at discrete spatial particle points and particles carry the prescribed mass with it. Then V_j can be approximated further by $\frac{m_j}{\rho_j}$ where m_j and ρ_j are the mass and the density of the particle j. Denote $f(\mathbf{x}_j)$ by f_j. Now the approximation formula

[1]This notation should not be confused with the usual angle brackets for inner product, since it is applied to a single argument.

for f at any location is given by

$$\langle f \rangle(\mathbf{x}) = \sum_j f_j \frac{m_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j, h). \tag{3.12}$$

Although there is a wide range of options for choosing kernel functions, kernel functions are required to satisfy several requirements to ensure the convergence of (3.12) to (3.9) in limit. In particular,

$$\int_\Omega W(\mathbf{x}, h) d\mathbf{x} = 1, \ \lim_{h \to 0} W(\mathbf{x}, h) = \delta(\mathbf{x}). \tag{3.13}$$

A popular kernel in graphics community is a cubic spline with a support radius $2h$ (M_4 kernel in [Monaghan, 2005])given by

$$W(\mathbf{x}, h) = \frac{\sigma}{h^d} \begin{cases} (2-\eta)^3 - 4(1-\eta)^3 & \text{if } 0 < \eta \leq 1 \\ (2-\eta)^3 & \text{if } 1 < \eta \leq 2 \\ 0 & \text{otherwise} \end{cases}, \tag{3.14}$$

where d denotes dimension of the simulation, and $\eta = \left\| \frac{\mathbf{x}}{h} \right\|$. The derivatives of f such as gradient or Laplacian are also easily approximated by applying differential operators to the kernel function of (3.9)

$$\langle \nabla f \rangle(\mathbf{x}) = \sum_j f_j \frac{m_j}{\rho_j} \nabla W(\mathbf{x} - \mathbf{x}_j, h), \tag{3.15}$$

$$\langle \Delta f \rangle(\mathbf{x}) = \sum_j f_j \frac{m_j}{\rho_j} \Delta W(\mathbf{x} - \mathbf{x}_j, h). \tag{3.16}$$

Alternative approximations are possible for differential operators to conserve phys-

ical quantities such as linear or angular momentum. We refer to [Monaghan, 2005] and [Adams and Wicke, 2009] for different differential operators and their usage in the approximation. Now, we can numerically solve the Navier-Stokes equation with a SPH formulation. The Navier-Stokes equations consist of the momentum equation and the continuity equation. If we assume that fluid is incompressible and Newtonian, which is typical in graphics, The continuity equation is given as

$$\nabla \cdot \mathbf{u} = 0. \tag{3.17}$$

The continuity equation is not needed in the Lagrangian formulation since the mass is carried by particles, and the total mass is preserved unless particles are inserted or removed. The momentum equation is given as

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \mu \Delta \mathbf{u} + \mathbf{f}^{ext}, \tag{3.18}$$

where $\mathbf{u}$, p, μ, $\mathbf{f}^{ext}$ denote velocity, pressure, viscosity constant and external forces, respectively. Note that the shear stress term ∇T of (3.1) is replaced by the Newtonian viscosity term $\Delta \mathbf{u}$ in the equation. The pressure p is typically a function of the density ρ of the fluid such as given by the Tait equation [Monaghan, 1994], which is

$$p = k \left(\left(\frac{\rho}{\rho_0} \right)^{\gamma} - 1 \right), \tag{3.19}$$

where k and γ are stiffness parameters and ρ_0 is the rest density of the fluid. In Weakly Compressible SPH scheme (WCSPH) [Becker and Teschner, 2007], a stiff value of γ is 7 and k is computed to allow the small density fluctuation. As an alternative to the stiff pressure solver that restricts the time step of WCSPH,

Predictive-Corrective Incompressible SPH scheme (PCISPH) [Solenthaler and Pajarola, 2008] is recently proposed. In PCISPH scheme, a new iteration step is introduced that allows much larger time steps. During each iteration, pressure values are predicted and corrected by minimizing the predicted variation of the density from the reference density.

To spatially discretize (3.18), we need to evaluate ρ and $\mathbf{f}^{ext}$ at discrete locations, and discretize differential operators $\frac{D}{Dt}$, ∇ and Δ. Denote $W(x_i - x_j, h)$ by W_h^{ij}. The density of the particle i, ρ_i is sampled by substituting f_j with ρ_j in the approximation equation (3.12)

$$\rho_i = \sum_j m_j W_h^{ij}. \tag{3.20}$$

The material derivative $\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla$ becomes the partial derivative with respect to time $\frac{\partial}{\partial t}$ in the Lagrangian veiwpoint, since the particles move along with the material and the advection part $\mathbf{u} \cdot \nabla$ disappears. The ∇ and Δ operators are derived from a SPH formulation such as (3.15), (3.16) or alternative formulations [Adams and Wicke, 2009]. An alternative to (3.15) [Adams and Wicke, 2009] is used to ensure momentum conservation for the pressure driven force, which yields

$$\frac{\langle \nabla p \rangle(\mathbf{x}_i)}{\rho_i} = \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_h^{ij}, \tag{3.21}$$

where p_i and p_j are the pressure of the particle i and j computed by (3.19). Now the straightforward spatial discretizaion of (3.18) becomes

$$\rho_i \frac{\partial \mathbf{u}_i}{\partial t} = -\langle \nabla p \rangle(\mathbf{x}_i) + \mu \langle \Delta \mathbf{u} \rangle(\mathbf{x}_i) + \mathbf{f}_i^{ext}. \tag{3.22}$$

Discretization in time is also straightforward. The velocity and the position of particles are updated at each time step. Leap-Frog or Euler scheme is commonly used for the integration scheme. The size of time step at each integration is global and usually determined by the Courant-Friedrichs-Lewy (CFL) condition. The basic SPH simulation algorithm is described in algorithm 1.

Algorithm 1 The SPH simulation algorithm

for all i **do**
  \\ find neighborhood indices
  $N_i = \{j : |\mathbf{x}_i - \mathbf{x}_j| < 2h\}$
  \\ compute density
  $\rho_i = \sum_{j \in N_i} m_j W_h^{ij}$
  \\ compute pressure
  $p_i = k \left(\left(\frac{\rho}{\rho_0} \right)^{\gamma} - 1 \right)$
\\ compute net acceleration
for all i **do**
  $\mathbf{a}_i = \frac{1}{\rho_i} \left(-\langle \nabla p \rangle(\mathbf{x}_i) + \mu \langle \Delta \mathbf{v} \rangle(\mathbf{x}_i) + \mathbf{f}_i \right)$
\\ integrate velocity and location
for all i **do**
  $\mathbf{u}_i = \mathbf{u}_i + \Delta t \mathbf{a}_i$
  $\mathbf{x}_i = \mathbf{x}_i + \Delta t \mathbf{v}_i$

The location $\mathbf{x}_i$, the velocity $\mathbf{u}_i$ and the mass m_i of the particle i are maintained along the time integration and used for updating the particle configuration, while density ρ_i, pressure p_i and the net acceleration $\mathbf{a}_i$ are computed at each time step, but not stored. In the first pass of the algorithm, density and pressure for each particle is computed. In the second pass, forces are computed. At last, forces are integrated and particle positions are updated. For the neighboring search, k-d trees and spatial hash grids have been popular data structures in graphics since searching neighbors in a finite range is simple and efficient, and their data structures is independent of the size of the domain.

Chapter 4

Surface Extraction of Particle Based Fluids Using Anisotropic Kernels

4.1 Implicit Surface Representation

Our surface definition is based on the approach proposed in [Müller *et al.*, 2003], where the surface is defined as an isosurface of a scalar field

$$\phi(\mathbf{x}) = \sum_j \frac{m_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j, h_j), \tag{4.1}$$

and W is an isotropic smoothing kernel of the form

$$W(\mathbf{r}, h) = \frac{\sigma}{h^d} P\left(\frac{\|\mathbf{r}\|}{h}\right). \tag{4.2}$$

In the above equation, σ is a scaling factor, h is a smoothing radius, d is the dimension of the simulation, $\|\mathbf{r}\|$ is the l_2 norm of a radial vector $\mathbf{r}$ and P is a symmetric decaying spline with finite support. The scalar field $\phi(\mathbf{x})$ is designed as a normalized density field that smooths out the scalar value of 1 at each particle's position over a continuous domain, and an isosurface from $\phi(\mathbf{x})$ gives a surface representation that coats the particles. The typical isovalue use for the surface is simply zero, or a small positive value to remove small sharp features. However, the resulting surfaces often have bumps, and there are two reasons for this. First, the irregular placement of particles makes it difficult to represent an absolutely flat surface. Although irregular sampling is an essential feature of any Lagrangian scheme, this irregularity of the positions of the boundary particles can make surfaces appear blobby. Second, the spherical shape of the smoothing kernels is not suitable to describe the density distribution near a surface. That is, in order to correctly model surface geometry, it is necessary for the density of the near-surface particles to decrease at different rates in different directions.

To resolve the problem of irregular particle placement, we apply one step of diffusion smoothing to the location of the kernel centers. This process can be interpreted as a 3D variant of Laplacian smoothing as described in [Taubin, 2000], and has the effect of denoising point clouds. The updated kernel centers $\overline{\mathbf{x}}_i$ are calculated by

$$\overline{\mathbf{x}}_i = (1-\lambda)\mathbf{x}_i + \lambda \sum_j w_{ij}\mathbf{x}_j / \sum_j w_{ij}, \tag{4.3}$$

where w is a suitable finite support weighting function and λ is a constant with $0 < \lambda < 1$. See (4.8) below for the of w_{ij}. We use λ between 0.9 and

1 in our examples to maximize the smoothing effect. Note that this smoothing process is used only for surface reconstruction, and the averaged positions are not carried back into the simulation. Typically, Laplacian smoothing results in volume shrinking, and our approach also shrinks the fluid volume slightly by moving the kernels for boundary particles towards the inside. However, in contrast to level-set methods, our approach does not shrink volume continuously as the simulation evolves. Furthermore, the analysis in Appendix 5.8 shows that the maximum distance from our reconstructed surfaces to the original particle positions is within a small constant of the particle radius scale.

To cope with the problem of density distributions near the surface, our new approach is designed to capture the density distribution more accurately by allowing the smoothing kernels to be anisotropic. By replacing h with a $d \times d$ real positive definite matrix $\mathbf{G}$, we can simply redefine W to be an anisotropic kernel

$$W(\mathbf{r}, \mathbf{G}) = \sigma det(\mathbf{G}) P(\|\mathbf{G}\mathbf{r}\|). \tag{4.4}$$

The linear transformation $\mathbf{G}$ rotates and stretches the radial vector $\mathbf{r}$. Therefore $W(\mathbf{r}, \mathbf{G})$ becomes an anisotropic kernel, and isosurfaces of W are ellipsoids instead of spheres. Note that the isotropic kernel can be treated as a special case of the anisotropic kernel by letting $\mathbf{G} = h^{-1}\mathbf{I}$ where $\mathbf{I}$ is an identity matrix. The key idea of our new method is to associate an anisotropy matrix $\mathbf{G}$ with each particle so that for particle j, $\mathbf{G}_j$ describes better the neighborhood density distribution.

Once all $\mathbf{G}_j$'s and $\overline{\mathbf{x}}_j$'s have been computed, we extract an isosurface (cf. (4.1))

from a redefined scalar field

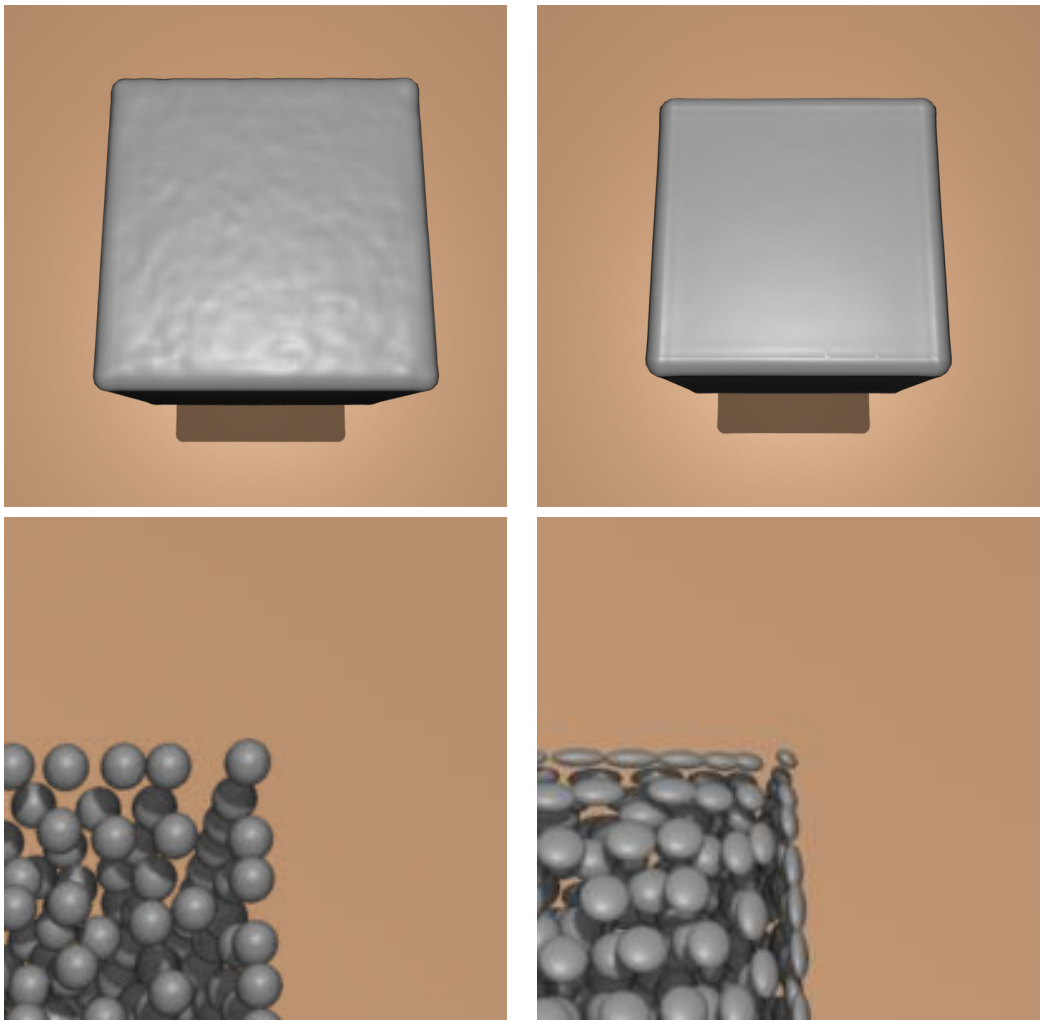
$$\phi_{new}(\mathbf{x}) = \sum_j \frac{m_j}{\rho_j} W(\mathbf{x} - \overline{\mathbf{x}}_j, \mathbf{G}_j). \tag{4.5}$$

It is necessary to point out that the equation of W is depending on the SPH simulation, since different SPH schemes can use different W's for density computation. For our examples, we use the B-cubic spline kernel from [Becker and Teschner, 2007].

4.2 Determining the Anisotropy

As mentioned in the previous subsection, our new approach determines an anisotropy matrix $\mathbf{G}$ for each particle in order to more accurately describe the density distribution around the particle. For example, in the neighborhood of a particle that is inside the fluid volume, the density is likely to be constant in all directions, making the corresponding $\mathbf{G}$ a scalar multiple of an identity matrix to keep the smoothing kernel W isotropic. On the other hand, around a particle that is near a flat surface, the particle density will decay faster along the normal axis than along the tangential axes. Then $\mathbf{G}$ should stretch W along the tangential axes and shrink W along the normal axis. At a sharp feature, the density will decay sharply in several directions, and $\mathbf{G}$ should shrink W in order to capture the sharp feature. See Figure 4.1 for a comparison between isotropic and anisotropic kernels that are near the surface of a region of fluid.

In order to determine $\mathbf{G}$, we apply the weighted version of Principal Component Analysis (WPCA) that is proposed in [Koren and Carmel, 2003] to the neighborhood particle positions. A drawback of the conventional PCA is its sensitivity to



(a) (b)

Figure 4.1: A comparison between the surface reconstruction using isotropic kernels (a) and our anisotropic kernels (b). Top row: the surface of SPH particles from a single dam break simulation. Bottom row: Illustration of particles at the top right corner. The shape of a particle in (b) represents the anisotropy of the corresponding smoothing kernel. Note that our approach constructs a flat surface with sharp edges and corners from properly stretched particles.

outliers, and it often produces inaccurate information when the number of samples is small and the sample positions are noisy, which commonly happens in particle-based fluids. In contrast, WPCA achieves significant robustness against outliers and noisy data by assigning appropriate weights to the data points. Specifically, WPCA begins by computing a weighted mean of the data points. Next, WPCA constructs a weighted covariance matrix $\mathbf{C}$ with a zero empirical mean and performs an eigendecomposition on $\mathbf{C}$. The resulting eigenvectors give the principal axes, and the eigenvalues indicates the variance of points along the corresponding eigenvalues. We then construct an anisotropy matrix $\mathbf{G}$ to match the smoothing kernel W with the output of WPCA.

In our approach, the weighted mean $\mathbf{x}_i^w$ and the covariance matrix $\mathbf{C}_i$ of particle i are formulated as follows:

$$\mathbf{x}_i^w = \sum_j w_{ij}\mathbf{x}_j / \sum_j w_{ij}. \tag{4.6}$$

$$\mathbf{C}_i = \sum_j w_{ij}(\mathbf{x}_j - \mathbf{x}_i^w)(\mathbf{x}_j - \mathbf{x}_i^w)^T / \sum_j w_{ij}, \tag{4.7}$$

The function w_{ij} is an isotropic weighting function with respect to particle i and j with support r_i.

$$w_{ij} = \begin{cases} 1 - (\|\mathbf{x}_i - \mathbf{x}_j\|)/r_i)^3 & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| < r_i, \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

With the finite support of w_{ij}, the computation is confined to the neighborhood particles within the radius r_i. In our examples, we choose r_i to be $2h_i$ in order to include enough neighborhood particles and obtain reasonable anisotropy informa-

tion. This kernel is also used to compute the averaged position of the particles in Eq. 4.3.

With each particle, the singular value decomposition (SVD) of the associated $\mathbf{C}$ gives the directions of stretch or compression for deforming the smoothing kernel W in terms of eigenvectors and eigenvalues. The SVD yields

$$\mathbf{C} = \mathbf{R}\Sigma\mathbf{R}^T, \tag{4.9}$$

$$\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_d). \tag{4.10}$$

where $\mathbf{R}$ is a rotation matrix with principal axes as column vectors, and Σ is a diagonal matrix with eigenvalues $\sigma_1 \geq \ldots \geq \sigma_d$. In order to deal with singular matrices and prevent extreme deformations, we check whether $\sigma_1 \geq k_r\sigma_d$ with a suitable positive constant $k_r > 1$. This condition is true when the largest variance in one principal axis is much bigger than the smallest variance in another axis. In this case, we modify $\mathbf{C}$ so that the ratio between any two eigenvalues are within k_r. Also, when the number of particles in the neighborhood is small, we reset W to a spherical shape by setting $\mathbf{G} = k_n\mathbf{I}$ in order to prevent poor particle deformations for nearly isolated particles. In addition, we multiply $\mathbf{C}$ by scaling factor k_s such that $\|k_s\mathbf{C}\| \approx 1$ for the associated particle inside the fluid volume, in order to keep the volume of W constant for particles with the full neighborhood. The aforementioned processes are formulated as follows to obtain a modified covariance matrix $\widetilde{\mathbf{C}}$.

$$\widetilde{\mathbf{C}} = \mathbf{R}\widetilde{\Sigma}\mathbf{R}^T \tag{4.11}$$

$$\widetilde{\Sigma} = \begin{cases} k_s \text{diag}(\sigma_1, \widetilde{\sigma}_2, \ldots, \widetilde{\sigma}_d) & \text{if } N > N_\epsilon, \\ k_n \mathbf{I} & \text{otherwise} \end{cases} \tag{4.12}$$

where $\widetilde{\sigma}_k = \max(\sigma_k, \sigma_1/k_r)$, N is the number of neighboring particles, and N_ϵ is a threshold constant. In our examples, we use $k_r = 4$, $k_s = 1400$, $k_n = 0.5$ and $N_\epsilon = 25$.

In order to make the kernel W of particle i deform according to $\widetilde{\mathbf{C}}_i$, $\mathbf{G}_i$ must be an inversion of $\widetilde{\mathbf{C}}_i$ and scaled by $1/h_i$ to reflect the original radius of particle i. Then our approach produces $\mathbf{G}_i$ as a symmetric matrix of the form:

$$\mathbf{G}_i = \frac{1}{h_i} \mathbf{R} \widetilde{\Sigma}^{-1} \mathbf{R}^T. \tag{4.13}$$

4.3 Alleviating Attraction Artifacts

Our particle relocation approach improves the qualify of fluid surfaces by denoising particle samples. However, as a side effect, the particle relocation may introduce unphysical attraction effects between fluid components. These artifacts are mainly noticeable in the case where two separate fluid components are approaching one another. When two separate fluid components become closer than the radial support r_i of (4.8), particles in one component begin to classify particles in the other component as neighbors, and thus are moved closer to the other component by the relocation step. This artifact occurs because we use a large support $r_i = 2h_i$, where h_i is approximately twice the average particle spacing r_a. Therefore, the relocation step pulls particles together even when they are further apart than r_a, as long as they are within the range of $4r_a$.

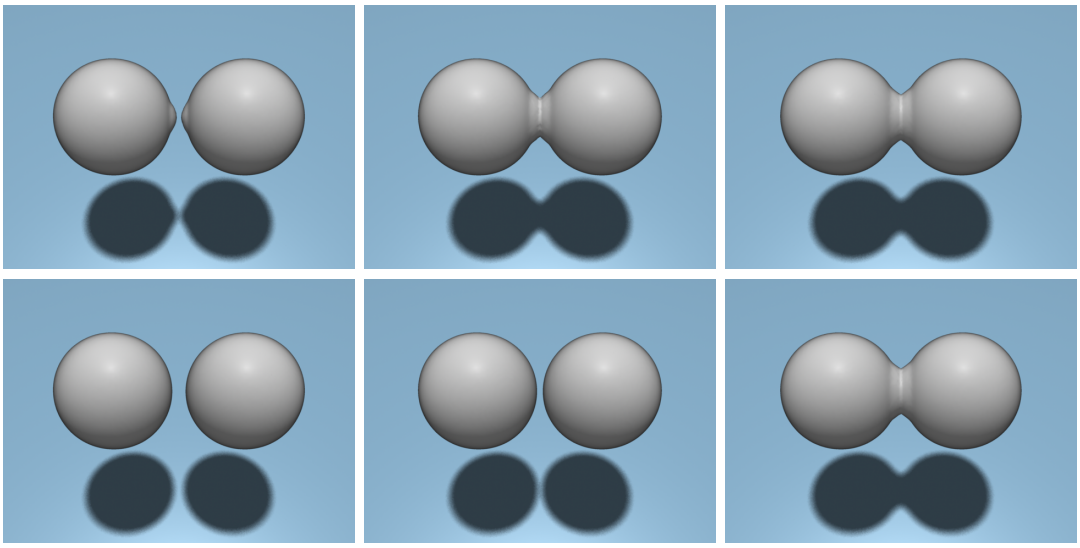


Figure 4.2: Merging spheres. Top: Not accounting for components. Bottom: Correction using connected components.

In order to alleviate this problem, we use a connected-component (CC) algorithm to mark simulation particles. We define two particles i and j as being connected if $\|\mathbf{x}_i - \mathbf{x}_j\| \leq r_a$. For a given particle, other connected particles are those neighboring particles within r_a. These nearby particles are easily identified by a neighborhood search. Starting with a seed particle, we compute its CC by a depth-first search graph traversal, and we label these connected particles with the index of the seed particle. To find all the CC's, we iterate through all the particles, and start a new traversal whenever we find a particle that has not yet been labeled. Once all of the particles have been labeled, we use a modified version of the weight w_{ij} of Eq. 4.8 that uses the CC information. Let the label of a particle i be denoted by c_i. The definition of w_{ij} is then modified to be

$$w_{ij} = \begin{cases} 1 - (\|\mathbf{x}_i - \mathbf{x}_j\|)/r_i)^3 & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| < r_i \text{ and } c_i = c_j, \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

With modified w_{ij} , the covariance matrix $\mathbf{C}_i$ and the anisotropic kernel $W(\mathbf{x} - \mathbf{x}_i, \mathbf{G}_i)$ is no longer affected by the neighbors that belong to different components. As a comparison between the original approach and the modified CC approach, Figure 4.2 shows two spheres approaching each other. As shown in the top row, the two spheres begin attracting each other before the collision when not using the CC labels. In contrast, the bottom row shows that the CC approach avoids this artifact and delays the merge due to the collision until the two spheres are much closer. This simulation used 23k particles. The overhead imposed by the CC computation is rather small. On average, the CC computation required 0.17 seconds per frame, while the surface reconstruction took 1.69 seconds per frame.

4.4 Implementation Overview

We use the Marching Cubes algorithm [Lorensen and Cline, 1987] to create a mesh that represents the fluid surface from the scalar field of Equation 4.5. We represent walls and geometric obstacles as signed distance fields for collision detection with particles. We refers readers to Appendix 5.8 for the C++ source codes used for determining anisotropic kernels.

4.4.1 Singular Value Decomposition

In order to capture the anisotropy for each particle, a robust and efficient SVD algorithm needs to be implemented. In our 3D examples, we use Cardano's method [Smith, 1961], [Kopp, 2008] to determine three singular values of the symmetric matrix $\mathbf{C}$ of (4.9). Although iterative methods such as two-sided Jacobi or QL are numerically more accurate, Cardano's method is efficient because it

analytically determines singular values, and it is robust enough for our approach because $\mathbf{C}$ is not ill-conditioned in most cases. From Cardano's method, we obtain three singular values $\sigma_1, \sigma_2, \sigma_3$ ((4.10)). Then we use a cofactor matrix of $\mathbf{C} - \sigma_i \mathbf{I}$ to determine a corresponding normalized singular vector $\mathbf{v}_i$ [Carchidi, 1986].

Since the cofactor method works best for singular values of multiplicity one, we identify different cases based upon the multiplicity of the singular values. Let us denote the machine precision by ϵ, and define a relation $a \approx b$ to be true when $|a - b| < \epsilon a$. When $\sigma_1 \approx \sigma_3$ we assume that the three singular values are nearly identical, and we set the singular vectors to be the column vectors of an identity matrix. When $\sigma_1 \not\approx \sigma_2$ and $\sigma_2 \approx \sigma_3$, we compute a singular vector $\mathbf{v}_1$, and choose $\mathbf{v}_2$ and $\mathbf{v}_3$ as two arbitrary orthonormal vectors in the plane normal to $\mathbf{v}_1$. The other multiplicity two case is handled by exchanging σ_1 and σ_3 in the previous case. If the singular values do not match any of the previous cases, we assume that they all have multiplicity one. We first compute $\mathbf{v}_1$ and $\mathbf{v}_3$. In order to correct the numerical error, we make sure that $\mathbf{v}_1$ are $\mathbf{v}_3$ orthogonal by computing $\mathbf{v}_3 - \mathbf{v}_1(\mathbf{v}_1 \cdot \mathbf{v}_3)$ and using this as an updated version of $\mathbf{v}_3$. The remaining singular vector $\mathbf{v}_2$ is computed by the cross product $\mathbf{v}_1 \times \mathbf{v}_3$.

4.4.2 Neighborhood Search

For neighborhood searches, we use a variation of the hash grid described in [Adams and Wicke, 2009] that handles the ellipsoidal support of our smoothing kernels. At every reconstruction step, we first compute an axis aligned bounding box (AABB) for the ellipsoid that is associated with each particle. Then we select the uniform grid cells that overlap with the AABB and retrieve the hash grid cells corresponding to these selected cells. We then store an index of the particle in

these retrieved cells. In order to find the neighboring particles at a given point, we determine the hash grid cell that contains the point, examine the particles whose indices are stored in the cell, and tag as neighbors the ones whose ellipsoids contain the point.

4.5 Optimization of Performance

One bottleneck of our approach is that we perform SVD on all particles in order to stretch them. A simple performance optimization is to only create stretched particles near the surface of the fluid, and not in the fluid bulk, in order to accelerate the process of surface reconstruction. We use a simple criterion to isolate the near-surface particles: For a particle i, we count the number N of neighborhood particles within the smoothing radius r_i of (4.8) and compute the center of mass $\mathbf{m}$ of these particles. A particle i is tagged as near-surface particles if $N < 0.9N_s$ or $\|\mathbf{m} - \mathbf{x}_i\| > 0.1r_i$, where N_s is the number of neighborhood particles for a inner-volume particle with rest density ρ_0. Once all of the particles have been examined, we perform SVD only on the tagged particles, while untagged particles remain unstretched using $\mathbf{G} = \mathbf{I}$.

Another bottleneck of our approach comes from the evaluation of the scalar field ϕ at the Marching Cubes grid points. In contrast to the isotropic kernel, the anisotropic kernel involves an extra matrix-vector multiplication $\|\mathbf{G}(\mathbf{x} - \mathbf{x}_i)\|$, and the scalar field value at one grid point is usually contributed to by multiple neighborhood kernels. In order to reduce this extra computational cost, we first introduce a fast exclusion test to check whether a grid point is out of the smoothing kernel support, formulated as $\|\mathbf{G}(\mathbf{x} - \mathbf{x}_i)\| > h_i$. Because $\mathbf{G}$ transforms a sphere

into an ellipsoid, the smallest singular value σ_{min} of $\mathbf{G}$ determines a radius of the inner sphere bounded by an ellipsoid that is transformed from the unit sphere. Therefore $\sigma_{min}\|(\mathbf{x}-\mathbf{x}_i)\| > h_i$ is a sufficient condition for $\|\mathbf{G}(\mathbf{x}-\mathbf{x}_i)\| > h_i$, and a matrix-vector multiplication can be avoided when this condition holds. In addition to the exclusion test, we further reduce the computation cost by observing that the value of ϕ is close to zero near the surface and increases to one towards the inside of the fluid volume. Therefore, we can assume that a grid point at $\mathbf{x}$ is inside the volume and far away from the surface if $\phi(\mathbf{x}) > t$, where t is a positive threshold less than one. At each grid point, we start adding contributions from the nearby particles to the grid point's scalar value. Once the scalar value reaches t, we stop adding contributions for the remaining particles and use t as a scalar field value for the point. For our simulation examples, the value $t = 0.2$ is used. This technique eliminates many of the smoothing kernel evaluations for a given simulation. We refer readers Section 4.6 for the timings on the performance improvement.

4.6 Results

In this section, we describe four simulations that were used to evaluate our surface reconstruction method: a water crown, flow on the bunny, falling figures, and a double dam break. All of our simulations and surface reconstruction algorithms were run on a 2.4 GHz Intel Core2 Duo CPU with 1.72GB of memory. We use NVIDIA Gelato for rendering the resulting animations. For the double dam break example, we present an interactive animation created by Simon Green at NVIDIA, who parallelized our method on the GPU. All of our results were simulated using the Weakly Compressible SPH (WCSPH) approach of Becker et al. [Becker and

Teschner, 2007]. We used a fixed time step of 0.0002s for running our simulations.

Figure 4.3 is an animation of a small drop of water that splashes into a larger body of water, causing a water crown. Only 24k particles were used to create this simulation. Note that even at this low particle count, the particle-based nature of the simulation is difficult to discern from the images. Our anisotropic kernel reconstruction of the surface creates a water crown that is smooth and unbroken near its base, and produces plausible pinch-off at the top. When the fluid rebounds in the center, a thin spike of water is maintained due to the stretching of the smoothing kernels along the spike's axis. When the water settles, the surface is smooth.

Figure 4.4 shows a viscous fluid that is poured over the Stanford Bunny. The fluid sheet that runs off the bunny is thin, usually just one particle thick, and yet the sheet is flat and smooth. In this sheet, the kernels are stretched in the two dimensions that run parallel to the sheet, and are compressed perpendicular to the sheet.

In the animation of Figure 4.5, we demonstrates the performance of our approach on a large scale simulation with one million particles. Two fluid versions of the bunny and the armadillo fall down an inclined plane into a pool. In this example, our optimization techniques from Section 4.5 are used. Table 4.1 shows the improvement on the timings when these optimization techniques are applied. For this example, the surface reconstruction method equipped with the optimization techniques improves timings three times as compared to the non-optimized reconstruction method. For anisotropic kernel computation (second column of Table 4.1), the speed gain is not as significant as expected, and we believe that this is due to an overhead of the neighborhood search for each particle that is unavoid-

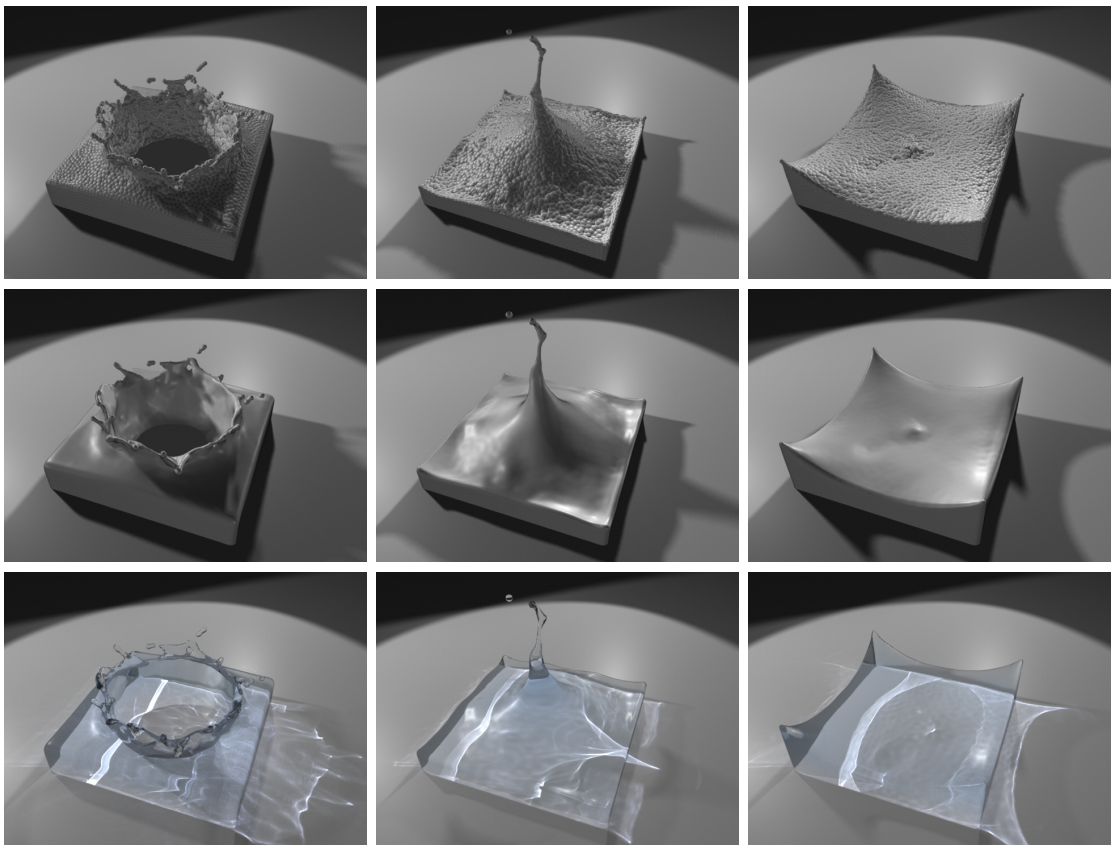


Figure 4.3: Water splash. Top: Anisotropic kernels, Middle: Opaque surface, Bottom: Transparent surface

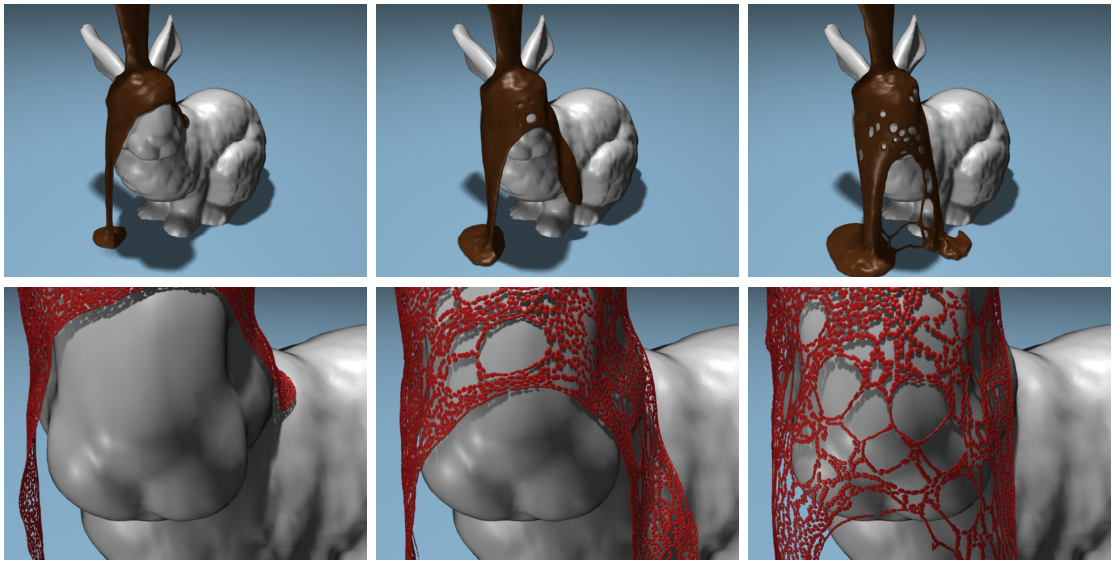


Figure 4.4: Melted chocolate falling on a Bunny.

able unless the search is implemented on GPU's or multi-core CPU's. For triangle mesh construction using Marching Cubes (third column of Table 4.1), We gain the substantial speed improvement of 5.2× due to the fast evaluation of the scalar field $\phi_{new}(\mathbf{x})$.

Table 4.1: Average per frame timings (in minutes) for different surface reconstruction methods on the falling figures example of Figure 4.5.

	Anisotropy	Marching Cubes	Total
Unoptimized method	1.07	3.09	4.16
Optimized method	0.83	0.59	1.42
Speed gain	1.3x	5.2x	2.9x

Our last simulation is a double dam break (Figure 4.6), in which two blocks of water at opposite sides of a tank are suddenly released. The two parcels of water rush towards each other, collide in the center of the tank, and this throws up a

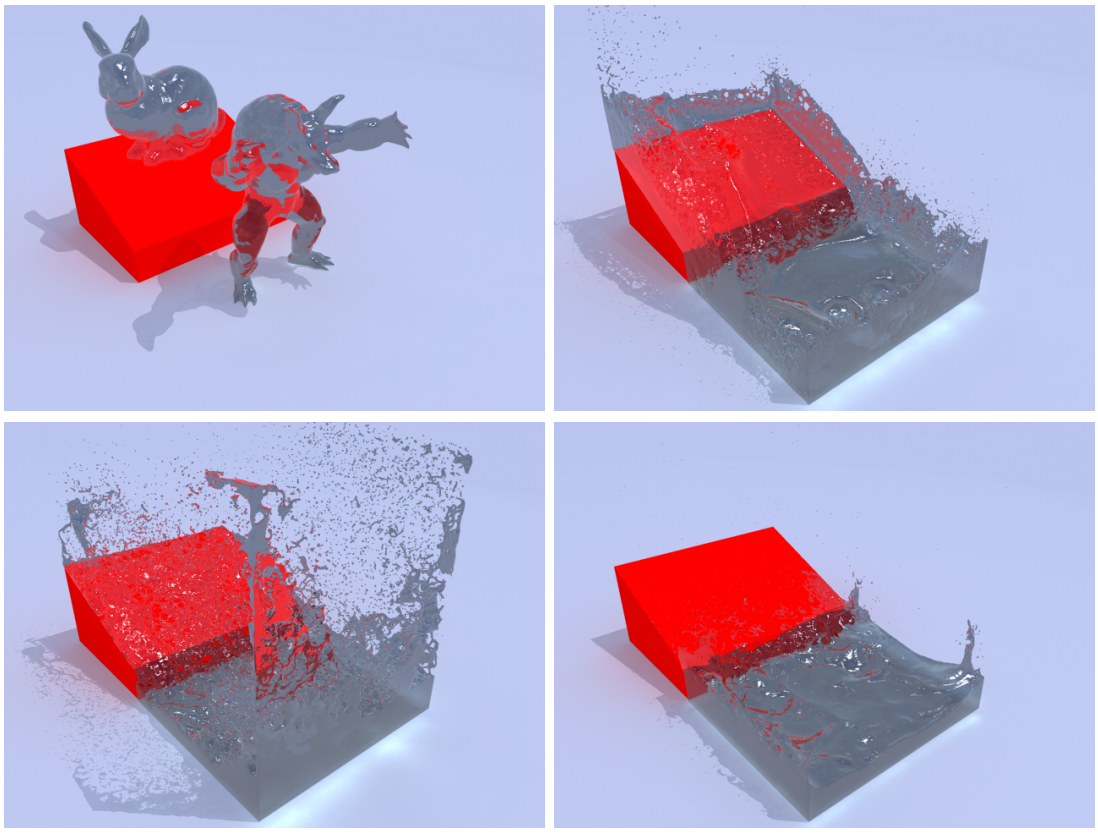


Figure 4.5: Falling armadillo and bunny.

thin sheet of water that runs diagonally across the tank. Similar to the bunny simulation, this thin sheet is often just one particle thick.

Figure 4.7 is an interactive animation of the double dam break, using NVIDIA's CUDA as our parallel programming architecture. For the surface reconstruction, a CUDA kernel with a thread per particle is used for particle relocation and anisotropic kernel computation. The fluid surfaces are rendered using ray-casting in a pixel shader instead of marching cubes. For each particle, the pixel shader calculates the intersection between the view ray and the ellipsoid (defined by the matrix), and also calculates the depth and surface normal. Then the screen oriented quads are extracted on the ellipsoids as point sprites. 128k particles were used for this example. The simulation, surface reconstruction, and rendering were simultaneously run on a GeForce GTX 460 video card and the interactive frame rate of 25fps was achieved. This example successfully demonstrates that parallelizing the surface creation process is simple and effective, since the approach only depends on local information about particle information.

4.7 Comparison

Figure 2.1 shows a comparison between an isotropic surface reconstruction approach [Müller *et al.*, 2003], Zhu and Bridson's approach [Zhu and Bridson, 2005], the method of Adams et al. [Adams *et al.*, 2007] and our anisotropic kernel approach. The simulation that is used for comparison is the double dam break simulation with 140K particles. As the figure shows, the isotropic reconstruction method produces unacceptably bumpy surfaces. Zhu and Bridson's approach creates noticeably smoother surfaces, but some surface bumps are still apparent. In

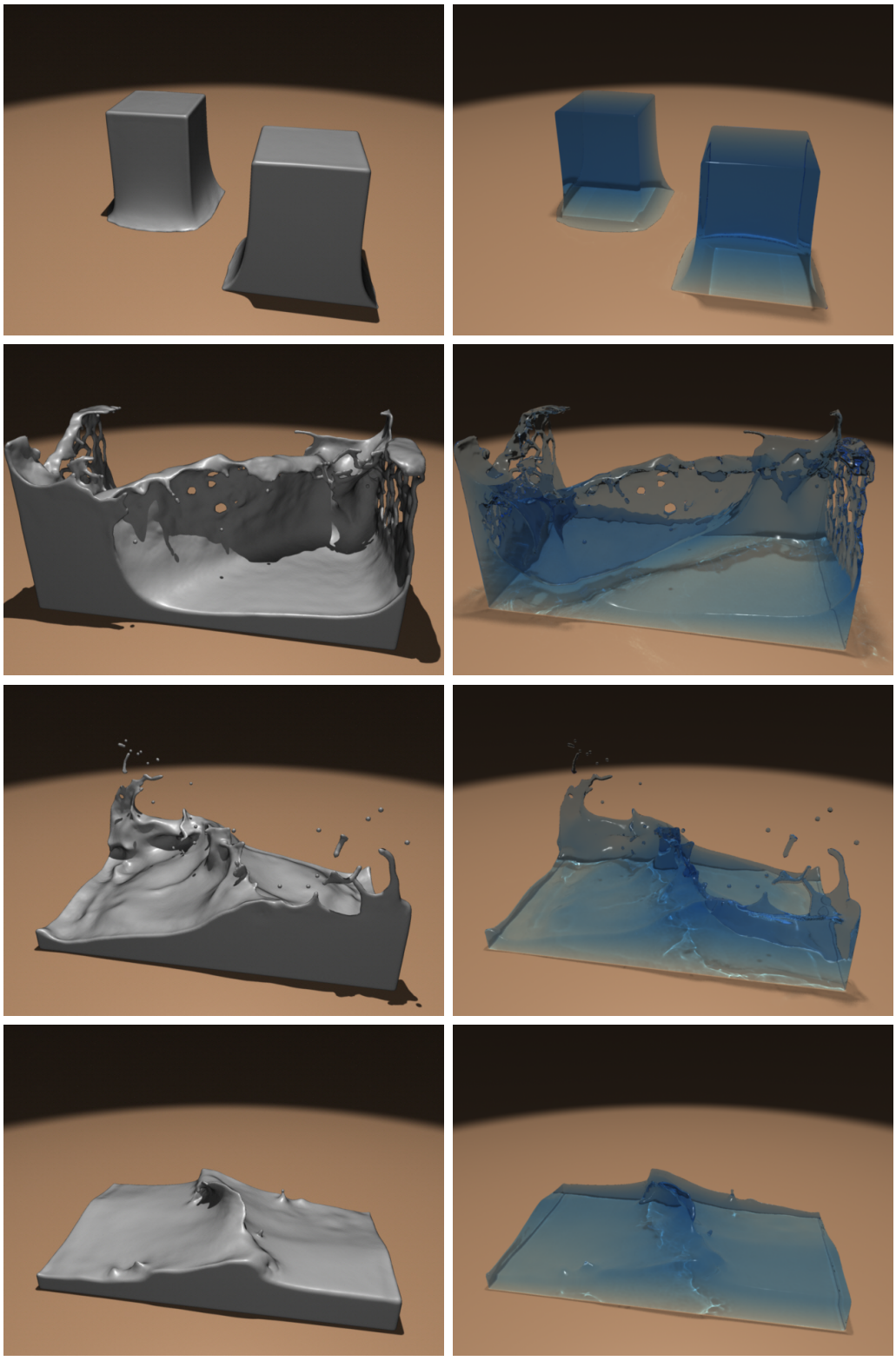


Figure 4.6: Double dam break simulation.

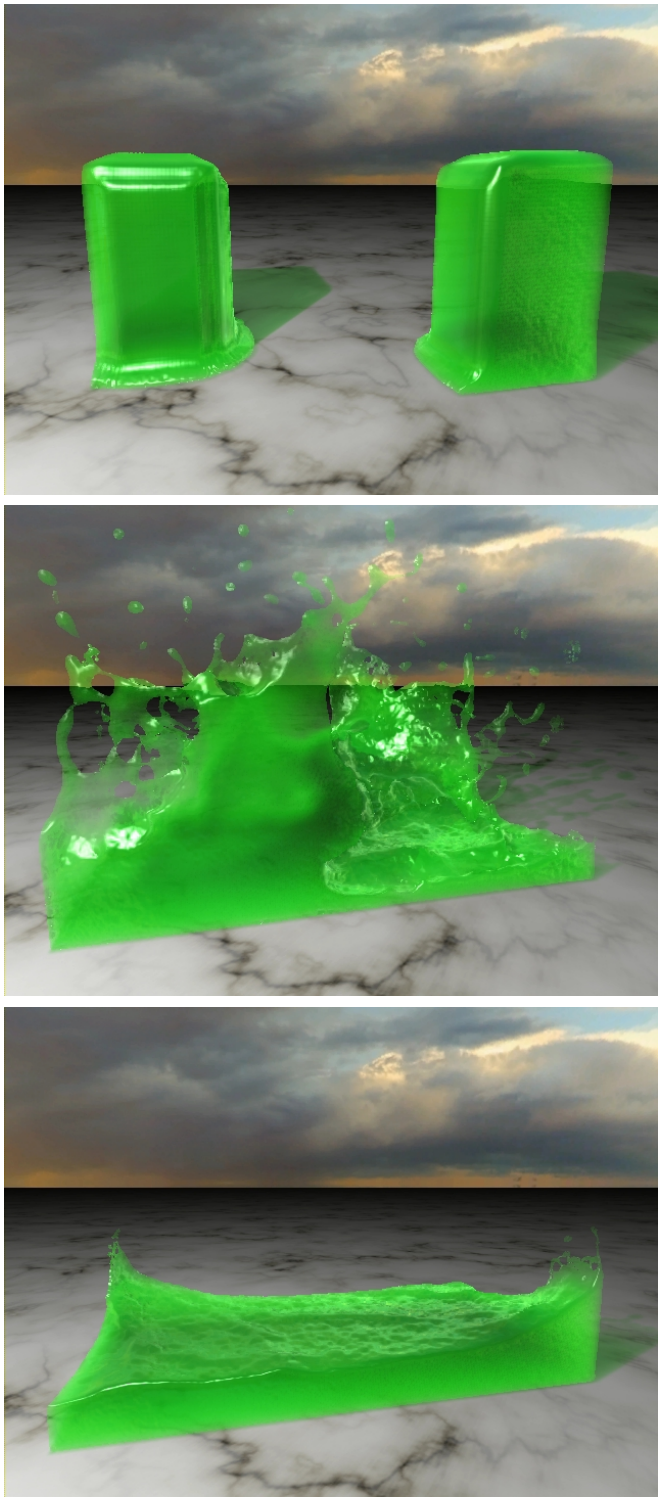


Figure 4.7: Interactive Double Dam Break

fairness to their method, Zhu and Bridson also perform a small amount of additional grid-based smoothing that we have omitted. The method of Adams et al. produces a still smoother surface, and this method creates the highest quality surfaces from among the prior methods that we have tested. Our anisotropic kernel method produces surfaces that are even smoother than the method of Adams et al. Moreover, our method creates a thin sheet of water in the center of the image that is largely unbroken, where the method of Adams et al. creates a sheet with many holes in a lace-like pattern.

Table 4.2 shows timings for the double dam break example. The dimensions of the marching cubes grid for these results is $230 \times 190 \times 350$. The timings are per-frame averages (in minutes) across all of the frames of the animation. Our surface reconstruction approach is roughly twice as expensive as the isotropic kernel method and Zhu and Bridson's approach. The method of Adams et al. is the most time consuming of the four methods, due to the need to re-calculate the signed distance field at a rate of 300 frames-per-second. When we dropped this re-calculation to a lower rate, the surface results from the Adams method became noticeably lower in quality.

Table 4.2: Average per frame timings (in minutes) for four surface reconstruction methods on the double dam break simulation.

Method	Surface reconstruction	Simulation	Opaque rendering
Isotropic	0.39	2.19	0.64
Zhu and Bridson	0.50		
Adams	1.76		
Anisotropic	0.96		

4.8 Limitations

There are several limitations to our approach for surface reconstruction. Perhaps the most important caveat is that the surfaces that are created using this method contain less volume than prior approaches. This is due to the averaging of particle centers. Appendix 5.8 gives an analysis of this volume difference, and demonstrates that a particle near a flat surface will move a fraction of the smoothing kernel radius h_i. Unlike mesh-based smoothing approaches, however, our method does not shrink the surface near thin sheets of fluids. Also note that this smaller volume is only a side-effect of the surface reconstruction process, and it is not carried into the physics of the simulation.

Even though our method produces surfaces that have less noise than the other methods that we tested, it is still possible to see small bumps when the surface is magnified. These slight variations in the surface can be seen in the pattern of the caustics of the water crown animation when the water settles. We think that these slight ripples could easily be smoothed away using mesh-based smoothing, but we left our meshes un-altered in order to clarify what can be achieved using anisotropic kernels alone.

4.9 Conclusion and Future Work

We have presented a new method of reconstructing surfaces from particle-based fluid simulations. This method relies on repositioning and stretching the kernels for each particle according to the local distribution of particles in the surrounding area. Our method preserves thin fluid sheets, maintains sharp features, and produces smooth surfaces when the simulated fluid settles. This method is also competitive

in speed compared to other recent techniques for SPH surface reconstruction.

There are several avenues for future work using this method. One possibility is to smooth away the slight ripples on the reconstructed surface by using the smooth particle skinning method [Williams, 2008] or its level-set improvement [Sin *et al.*, 2009] on the ellipsoidal kernel representation. In addition, this approach to surface creation should be applicable to particle-based simulations other than SPH, and in particular, this approach can be tried for PIC/FLIP simulation [Zhu and Bridson, 2005] that is popular in the visual effects industry. Finally, it would be interesting to investigate whether there is a way to carry texture information along with the surface, as is possible using Semi-Lagrangian contouring [Bargteil *et al.*, 2006]. Our contribution described in Chapter 5 is inspired by this insight: We adopt the explicit mesh tracking method of [Wojtan *et al.*, 2009] to carry surface properties like color or texture with the surface.

Chapter 5

Explicit Mesh Surface for Particle Based Fluids

5.1 Mesh Advection and Topology Changes

Our new approach to surface tracking maintains an explicit mesh that is consistent with an isosurface defined by the position of the SPH particles. This isosurface $\phi(x)$ is used in two ways: 1) to create the initial surface mesh, and 2) as a surface to project mesh vertices onto during subsequent SPH simulation time steps. We use the anisotropic kernel method of Yu and Turk [2010] to define our isosurface based on the particle positions. This method is similar to defining an isosurface based on the sum of per-particle radial basis functions, but differs in that each basis function may be stretched according to the particle distribution in a local neighborhood. The anisotropic kernel method yields an isosurface that is quite close to the center of SPH particles that are on the boundary, and this helps us to interpolate boundary particle velocities onto the mesh vertices. Specifically, we

define the iso-surface as a zero-levelset of the scalar function $\phi(x)$. The function $\phi(x)$ is defined as

$$\phi(x) = \phi_{new}(x) - \alpha, \tag{5.1}$$

where $\phi_{new}(x)$ is the scalar field of (4.5) in Section 4.1 and α is an user-defined isovalue and the value of 0.05 is used for running our examples.

Our method begins by constructing an initial surface mesh by applying Marching Cubes to the isosurface $\phi(x)$. Once this initial mesh is created, our surface tracking method consists of repeating the following four tasks for each time step:

1. Advect the mesh vertices according to the SPH particle velocities.
2. Improve triangle shapes using edge split and collapse operations.
3. Project the mesh vertices onto the implicit surface.
4. Perform topological changes when the fluid bulks merges or splits.

Task 1: To prepare for the advection of a vertex at a given time step, we retrieve all of the simulation particles within the SPH smoothing radius of the vertex using a spatial hash grid. We calculate normalized weights for each of these selected particles by evaluating SPH's smoothing kernel of their distances to the vertex. The vertex velocity is given by summing the weighted velocity of these neighborhood particles. Therefore, the velocity of vertex i is given by

$$\mathbf{v}_i = \sum_j W_{ij}\mathbf{v}_j / \sum_j W_{ij}, \tag{5.2}$$

where W_{ij} is the SPH smoothing kernel evaluation using (4.2) on a distance between vertex i and particle j, and $\mathbf{v}_j$ is the velocity of particle j.

Then we advect both the surface mesh and the SPH particles. We chose simpletic Euler as our numerical advection scheme because it is stable, energy preserving, and simple to implement.

Task 2: We perform edge collapses and edge splits at the end of each advection step to maintain good mesh quality. Maintaining good mesh quality has several advantage: the surface mesh resolution remains regular for property advection and we can robustly evaluate curvatures and bilaplacians on the mesh. In this task, we ignore topological inconsistencies at this stage except a sub-grid scale splitting event that we will describe later in Section 5.1.1.

Task 3: Although the advection step keeps the surface vertices near to the isosurface, accumulated numerical errors may cause inconsistency between the explicit mesh and the isosurface. In addition, topological changes to the fluid such as merging and splitting will produce surface vertices that are too far off the isosurface. Our projection step is designed to keep the surface mesh closely matched to the isosurface. In addition, it is during this projection step that we detect the regions where topological changes occur (to be discussed below). For an updated vertex v_i at x_i after the advection step, we project the vertex onto the isosurface of $\phi(x)$ by performing a binary search along the ray segment from x_i to $x_i + \epsilon n_i$ where n_i is the vertex normal obtained from the discrete mean curvature computation. The value of ϵ is $sign(\phi(x_i)) \cdot r_a$ where $\phi(x_i)$ is the evaluation of the isovalue at v_i and r_a is a global constant representing the average particle spacing. The value of r_a is used 0.04 in our examples. When $\phi(x_i) \cdot \phi(x_i + \epsilon n_i) \leq 0$, we refine the interval until the new vertex is located where the isovalue is close enough to zero or until the maximum number of iterations has been performed. For all examples in this chapter, we use four iterations and the value of 0.001 for the isovalue threshold.

Task 4: This projection step will fail if $\phi(x_i) \cdot \phi(x_i + \epsilon n_i) \geq 0$, because we cannot locate the zero isovalue position along the ray. This condition signals a topological change — the isosurface of $\phi(x)$ has either merged or split, so we need to merge or split the surface mesh as well in order for it to remain consistent with the simulation. Topological merges may produce vertices inside the fluid volume with a positive isovalue, while topological splits may produce vertices out outside the fluid volume with a negative isovalue. For both cases, we first move v_i to $x_i + \epsilon n_i$ in case $|\phi(x_i + \epsilon n_i)| < |\phi(x_i)|$. We also set a topology change flag that indicates that we need to identify and fix such local changes. We then use the method of Wojtan et al. [2009] described in Section 5.1.1 to repair the topology of the mesh.

It is worth mentioning that by projecting the surface mesh onto the SPH isosurface, we sacrifice one benefit of using an explicit mesh — that it can easily represent thin features. We can easily allow these thin sheets if we advect the surface mesh without projection. Unfortunately, ignoring the projection would lead to drift in the simulation (either the surface drifting too far away from the SPH particles, or the SPH particles drifting far outside of the surface mesh). One way to correct this drift is to re-sample the SPH particles inside the surface mesh. However, continual re-sampling negates the Lagrangian benefits of SPH, ultimately leading to an inefficient meshless approximation of an Eulerian method. Instead, we chose to project the surface mesh onto the SPH isosurface in order to remain faithful to the Lagrangian simulation.

5.1.1 Repairing the Topology of the Mesh

The method of Wojtan et al. performs a local re-meshing of the surface mesh in order to handle the topology changes. We refer the reader to [Wojtan *et al.*, 2009] for details of resolving topology changes of the mesh, but we briefly outline the steps here. After the projection step, we calculate a signed distance field $\mathbf{D}$ on a regular grid for the surface mesh. In our examples, the value of the grid is 0.002. Since topological changes only occur at the grid cells that intersect with the mesh, we calculate signed distances only for those cells and we mark them as "surface cells". Specifically, we calculate the distances to the nearby triangles from each grid point of the surface cell and take the minimum as the exact distance for that point. Then we flag grid points as inside of the mesh or outside of the mesh by the voxelization technique described in [Müller, 2009]. We assign positive signed distance to the points inside and assign negative signed distance to the points outside.

We now have three different surfaces: the levelset of $\phi(x)$, one is defined as a surface mesh and the other defined by a zero level-set L of $\mathbf{D}$. This signed distance field $\mathbf{D}$ (and not the implicit function $\phi(x)$) will be used to recognize grid cells that require changing the mesh. We first mark each cell as a "complex-cell" in which the explicit mesh surface is connected significantly differently than the level-set, i.e. the eight corner samples of the distance field disagrees with the mesh in terms of topology. Specifically, a cell is marked as a complex cell if any of its edge intersects with the surface mesh more than once, or any of its face intersects the surface mesh in the shape of a closed loop, or it has the same sign of the signed distance function at its all eight corners while also having explicit geometry of the mesh inside of it. Complex cells contrast the inconsistent regions between the explicit mesh and

the levelset. However, we do not want to re-sample the explicit mesh on all of the complex cells to preserve the surface details such as sharp corners. We further mark some of complex cells as "deep cells" and perform topological changes only to those deep cells. We ignore any complex cells that are close within one cell to both surface representations. We only mark a complex cell as a deep cell that is at least one cell away from the level-set.

Next, we alter the topology of the surface mesh in the deep cells. The surface mesh is clipped to these cells, and the polygons inside such cells are removed. New triangles are created using Marching Cubes inside these deep cells, and these triangles are sewn together with the mesh at the cell faces. This approach handles most of the topology change mesh events, including surface merging, surface splitting and other self-intersection events at the resolution of the grid.

The topological event that the above procedure does not handle is a sub-grid scale splitting event, as occurs with a droplet pinch-off. We treat this case as in [Wojtan *et al.*, 2010], during edge collapse operations of Task 2. As illustrated in Figure 5.1, when a short edge is identified for collapse, we first check to see whether the edge is part of a thin tunnel surrounded by a ring of three edges that would be flattened if the edge were to be collapsed. Such a case indicates a topological split, and we do not perform the edge collapse. Instead, we replicate the vertices along the ring of edges, separate the mesh into two parts at this ring, and cap off these rings with one triangle each.

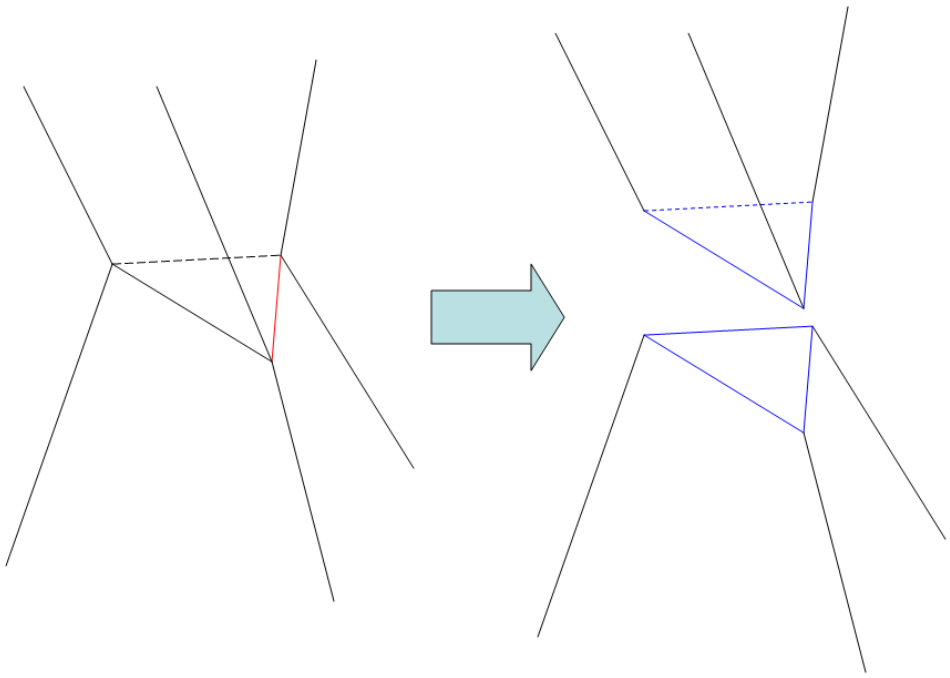


Figure 5.1: Replication of the ring edges. Left:A red colored edge is flagged for the collapse operation. Right: Instead of collapsing the edge, the mesh is separated by replicating the ring edges into the blue colored triangles.

5.2 Surface Property Advection

By maintaining the explicit surface mesh representation, we are able to easily track surface quantities such as colors, textures or physical properties. We choose to carry surface quantities on the mesh vertices because the interpolation and extrapolation schemes at vertices are straightforward. As the surface mesh is advected, a local re-meshing happens when we perform edge operations or topological changes.

When a new vertex is created by the edge split operation, the properties in one-ring neighbor vertices are linearly averaged according to the edge length weight. When two vertices are merged by the edge collapse operation, two properties are averaged on a new vertex. In the event of topological changes, we remove local invalid mesh patches and replace them with newly created correct topology triangle patches. To assign surface properties to the newly created vertices, all vertices connected to an original mesh vertex (one that was not replaced by the topological operation) are assigned extrapolated properties from the original vertices and pushed into a queue. Then we propagate the properties of the original vertices to the interior of the patch by iteratively popping the vertex at the front of the queue, extrapolating its properties onto the unvisited one-ring neighbor vertices, and then pushing the neighbors into the queue. We could use more sophisticated methods such as fast marching, but we found that our flood-fill approach is simple and accurate enough, especially because the size of newly created patches is usually fairly small.

The main benefit of tracking surface quantities using the explicit mesh is that we introduce little dissipation of the quantities over time. Most mesh vertices are carried across many time steps undisturbed, and thus the properties at these

vertices remain untouched. This is in contrast to mesh-based surface tracking methods that perform global re-meshing at each time step.

Another benefit of keeping an explicit representation is that we can delay the quantity tracking and perform it as a post-process. Instead of transferring surface quantities during the simulation, we can make a separate sequence by storing each quantity transferring process per-timestep. That is, for each newly created vertex we make a list of the influencing vertices and their weights. Once we build this transferring sequence, we can track different quantities on the identical simulation. The most interesting application of this post-processing approach to tracking surface properties is the generation of surface waves on a mesh sequence. We refer readers Section 5.6 for the details of this method.

5.3 Surface Tension Models

We now want to apply surface tension to the SPH particles as a body force. The curvature normal at each mesh vertex is computed by applying the discrete mean curvature operator on the surface mesh. We then create a per-vertex surface tension force and transfer these forces to the SPH particles. To do this, we first insert mesh vertices into a hash table. Then for each particle, we locate the nearby mesh vertices within the SPH smoothing radius. For a near boundary particle p_i that has mesh vertices nearby, the surface tension force $\mathbf{f}_i$ is given by:

$$\mathbf{f}_i = \gamma \sum_j W_{ij} A_j \boldsymbol{\kappa}_j / \sum_j W_{ij} A_j, \tag{5.3}$$

where γ is the surface tension coefficient, W_{ij} is the SPH smoothing kernel evaluation on a distance between particle i and vertex j, and $\boldsymbol{\kappa}_j$ is the curvature

normal of vertex j. The value A_j is the area of the mesh closest to vertex j, that is, A_j is one-third of the sum of the area of the triangles adjacent to j. We use the shape operator of Meyer et al. [2002] to calculate the curvature $\boldsymbol{\kappa}_j$ from the one-ring of triangles around vertex j. We use area-based weights to generate the surface tension forces of Equation 5.3 since small area triangles are often obtuse and Meyer's approach is unstable for such poorly-shaped triangles. By weighting these curvature estimates using triangle areas, we are able to attenuate the noise caused by these cases.

The double density relaxation method [Clavet *et al.*, 2005] and the molecular cohesive force approach [Becker and Teschner, 2007] also result in the effect of surface tension. In these formulations, the surface tension is rather an emergent feature from the methods and the tension force is not induced from the mean curvature of the fluid surface. In contrast, our surface tension model captures the detailed curvature from the explicit surface representation and the tension force is explicitly induced from the fluid surface.

The color field approaches to surface tension [Müller *et al.*, 2003], [Hu and Adams, 2006] suffer from numerical errors caused by irregular particle samples. In contrast, our approach is unaffected by the particle distribution because the curvature information on the surface is computed from the high resolution mesh representation. We refer readers Section 5.6 for the comparison of our approach with one of the previous SPH tension approaches.

5.4 Small Scale Surface Dynamics

Our mesh-based surface tension induces forces on the SPH particles, and causes smoothing of the fluid that is characteristic of small-scale flows. There is, however, another phenomena due to surface tension that would be computationally prohibitive to simulate using forces on particles, namely capillary waves. Capillary waves are nonlinear waves traveling on the surface of a fluid, whose dynamics are dominated by the effects of surface tension. We take a different approach to simulating this phenomena, and in particular we simulate this form of wave propagation directly on the surface mesh. In order to do this, we treat these waves as displacements from a base mesh in the normal direction. These displacements are properties that we store directly on the mesh vertices, much like color values.

We perform our surface wave physics on a dynamic surface mesh $\mathbf{D}$ that is constructed from a base surface mesh $\mathbf{B}$ that represents the fluid surface. Initially, $\mathbf{D}$ is just a duplicate of the base mesh $\mathbf{B}$. At each simulation step, we perform surface wave dynamics on $\mathbf{D}$ and the updated displacements of $\mathbf{D}$ are stored back on $\mathbf{B}$ and carried to the next step by advecting $\mathbf{B}$. Similar to the wave equation model used in [Wang *et al.*, 2007] and [Thürey *et al.*, 2010], we allow the vertices of $\mathbf{D}$ to move along the normal rays of the corresponding vertices of $\mathbf{B}$. We store the height scalar h and a scalar representing the normal component of the velocity v from $\mathbf{D}$ on the corresponding vertices in $\mathbf{B}$. By maintaining the configuration of $\mathbf{D}$ as scalar quantities on the vertices of $\mathbf{B}$, tracking $\mathbf{D}$ is simple under topological events or edge operations. That is, we treat h and v as surface quantities (similar to color) and assign influence values on the newly created vertices using the method described in Chapter 5.2.

We use a thin-plate bending energy model that is similar to [Williams, 2008]

and [Bergou *et al.*, 2006] for our surface wave simulation. The method of Thüery et al. [2010] computes a surface tension model at the highest resolution by solving a wave equation on the surface. As mentioned in their paper, this linearized approximation is limited by an unphysical, user-specified constant wave speed, while natural capillary waves experience a wave speed dependent on the wave number. By directly modeling bending energy instead of the linearized wave equation, we successfully generate small waves at varying speeds, which appear visually less disturbing. To overcome numerical drift and volume loss of $\mathbf{D}$ under bending energy minimization, we add a shaping energy force by linking linear springs from the vertices of $\mathbf{D}$ to the corresponding vertices of $\mathbf{B}$.

We model bending energy as a weighted surface bilaplacian as described in [Williams, 2008]. Given a dynamic mesh with n vertices, let W be a $3n \times 3n$ matrix representing angle cotangent weights, let A be a $3n \times 3n$ diagonal matrix representing scaled vertex areas, and let X be a $3n \times 1$ vector representing vertex locations, so that the discretized curvature normal is expressed as $A^{-1}WX$. We approximate bending energy as:

$$E_b = k_b X^T (W^T A^{-1} W) X, \tag{5.4}$$

where k_b is a bending stiffness coefficient. By assigning lumped mass on the surface, the discrete shape energy is formulated as:

$$E_s = \frac{k_s}{2}(X - X_b)^T A (X - X_b), \tag{5.5}$$

where X_b is a vector representing vertex locations of the base mesh $\mathbf{B}$ and k_s is a spring stiffness coefficient. X_b remains unchanged during the surface dynamics

integration to provide a reference mesh location. The total kinetic energy of the surface is given to be $T = \frac{1}{2}\dot{X}^T A\dot{X}$. From the Lagrangian $L = T - (E_b + E_s)$, the Euler-Lagrange equation is given as:

$$\ddot{X} = 2k_b(A^{-1}W)^2X + k_s(X - X_b). \tag{5.6}$$

To ensure that our capillary waves propagate over long distances and do not dissipate too early, we apply a variational integrator [Stern and Desbrun, 2006] to the Lagrangian and obtain a symplectic Euler method for our time integration rule. For each SPH simulation step, we perform $5 \sim 10$ iterations of wave dynamics. The wave frequency is proportional to k_s and the wave height is proportional to k_b. W and A are recomputed at every iteration as $\mathbf{D}$ is updated. Instead of explicitly constructing W and A matrices, we exploit the mesh connectivity to compute $(A^{-1}W)^2X$: First, $A^{-1}WX$ is computed by gathering from the one-ring neighbors of each vertex and storing the computed value back on each vertex. In a similar fashion, $A^{-1}W(A^{-1}WX)$ is computed by gathering the stored values of the one-ring neighbor of each vertex.

We found that the bending force becomes noisy when $\mathbf{D}$ is displaced more than the average edge length from $\mathbf{B}$ due to poorly-shaped triangles. To maintain a higher mesh quality, we project the location of vertices in $\mathbf{D}$ onto the vertex normal ray of the corresponding vertices in $\mathbf{B}$. As long as these displacements are small enough, this is justifiable since the volume preserving curvature flow runs along the normal direction. This adds a small but acceptable amount of viscosity. We can manually introduce energy dissipation by adding velocity damping. Taming the wave height can be done by clamping the height above a fixed limit. We can also turn off surface waves for small features and high curvature regions based on

the curvature information of the base mesh $\mathbf{B}$.

5.5 Implementation Overview

Throughout the Chapter 5, we introduce three different time scales for the proposed surface tracking method: the SPH simulation time step, the rendering frame rate and the capillary waves simulation time step. For SPH simulations, we use a fixed SPH simulation time step 0.0002s (5000 fps) to evolve the particle system. For the high quality rendering of the surfaces, we use various frame rates of 30, 60 or 120fps, i.e. one simulation step out of every few hundred steps is chosen for the high quality rendering. Capillary waves simulation are decoupled from the SPH simulation and performed on the dynamics mesh $\mathbf{D}$ between simulation steps. A fixed time step of 0.001s is used for running our capillary wave examples.

In order to exploit the benefits of the explicit mesh tracking, it is crucial to employ a simple and efficient data structure for the mesh connectivity representation and the mesh traversal. For our explicit mesh representations, we decided to use the corner table (CT) structure [Safonova and Szymczak, 2003] to perform edge operations and traverse one-ring neighbors of a vertex. Regarding the edge operations, an edge $\mathbf{e}$ is split if $l(\mathbf{e}) > \alpha r_a$ and is collapsed, if $l(\mathbf{e}) < \beta r_a$, where $l(\mathbf{e})$ is the length of $\mathbf{e}$ and r_a is the average particle spacing. To prevent oscillatory splits and collapses, β should be smaller than half of α. We use $\alpha = 2$ and $\beta = 0.5$ for all our simulations.

In our tracking method, the neighborhood search is a globally employed routine used in various situations. It is used for the velocity interpolation (5.2) from SPH particles to mesh vertices, and the mean curvature force interpolation from mesh

vertices to SPH particles (5.3). SPH simulation itself heavily uses the neighborhood search to compute density and force of particles. Our surface reconstruction method [Yu and Turk, 2010] also determines the anisotropic kernels according to the neighboring particle distributions. Furthermore, during the projection step, we use the neighborhood search for evaluating the iso-value several times in order to couple the explicit mesh with the implicit iso-surface. We use a single-threaded CPU implementation of the spatial hash grid to perform the neighborhood search. Although the hash grid is well suited for unbounded simulations, it is the major computational bottleneck for our approach. In the future, we plan to implement a parallel version of the hash grid structure on GPU to gain significant performance improvements over the surface tracking, reconstruction and the SPH simulation, compared to the current timings shown in Table 5.1.

5.6 Results

We have used our mesh-based surface tracker to produce several SPH animations, and we describe them in this section.

Figure 5.2 shows an example of a Raleigh-Plateau instability. The initial condition for this example is a dumbbell in zero gravity. Due to the surface tension forces (induced by the surface mesh), portions of the bar become thin and eventually pinch off. This illustrates that our mesh surfaces can undergo topological splits. As can be seen in our video, the separation of these components also induce ripples on the surface of the different drops. In this example, these ripples are purely based on the positions of the SPH particles, and no mesh-based dynamics is used. This simulation used 7,279 particles.

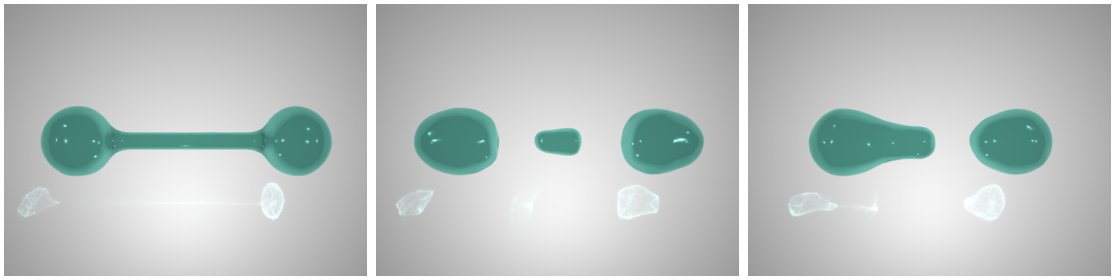


Figure 5.2: A dumbbell in zero gravity exhibits pinch-off due to surface tension effects (Rayleigh-Plateau instability). Later in this animation the separate components re-join each other.

Figures 5.3 and 5.4 show various methods for carrying properties such as color on the surface of an object that is represented using particles. In the sequence of Figure 5.3, three highly viscous figures (a bunny, a cube, and an armadillo) are dropped. The bunny and the armadillo spill over both sides of a bar, and they all pool onto the floor. We use the vertices of our surface mesh to carry color information for each of these three objects. We use an average edge length that is one quarter of the average inter-particle distance, and this allows us to carry a more fine resolution of color information than per-particle colors. Our method blends color values when a mesh triangle becomes overly stretched, at which time we subdivide the triangle and use interpolation to create a color value at the new vertices. We also blend colors during topological changes to the mesh. The surface of this material in this example is heavily stretched, yet the colors do not blend together at their common borders. This simulation used 29,172 particles. Figure 5.4 shows how our method compares to the logical alternatives. Part (a) of this figure shows a color-per-particle view, which is the information used for parts (b) and (c). Part (b) uses a distance-weighted average of per-particle colors.

Part (c) demonstrates coloring the surface mesh based on the color of the nearest neighbor particle. Part (d) shows our mesh-based method of carrying colors.

The simulation of Figure 5.5 demonstrates two spheres of water merging in zero gravity. This animation was produced using 12,532 particles. When the drops merge, this causes ripples on the water's surface. Along with the simulation, we added surface dynamics to these drops based on the surface bilaplacian. Please watch the accompanying video to observe the differences between these animations.

Figure 5.6 shows the effect of surface tension on a cube of water in zero gravity, inspired by the example of [Brochu *et al.*, 2010]. This was our lowest resolution simulation, and used 4,096 particles. Our mesh-based surface tension rounds off the corners of the cube and causes the bulk of the fluid to be drawn to a more spherical shape. Due to the momentum of the fluid, the sphere shape is overshot and the fluid moves to a nearly octahedral shape before being drawn back towards the sphere. As can be seen in the accompanying video, this oscillation between the cube and octahedron occurs several times before the fluid settles into a sphere. In the video, we compare our results to the method of Becker and Teschner [2007]. Their particle-based surface tension approach also draws the cube towards a sphere, but this motion is considerably more damped than the motion from using our mesh-based surface tension forces.

Our final example is that of a water crown that was created by a falling drop of water hitting the surface of a still pool (Figure 5.7). Note that this example demonstrates numerous topological changes to the surface, and these are handled by the local topology repair method. In Figure 1.2, we show opaque renderings from two versions of this animation, one without surface dynamics and one that adds surface ripples using our surface dynamics post-process. Note that in the

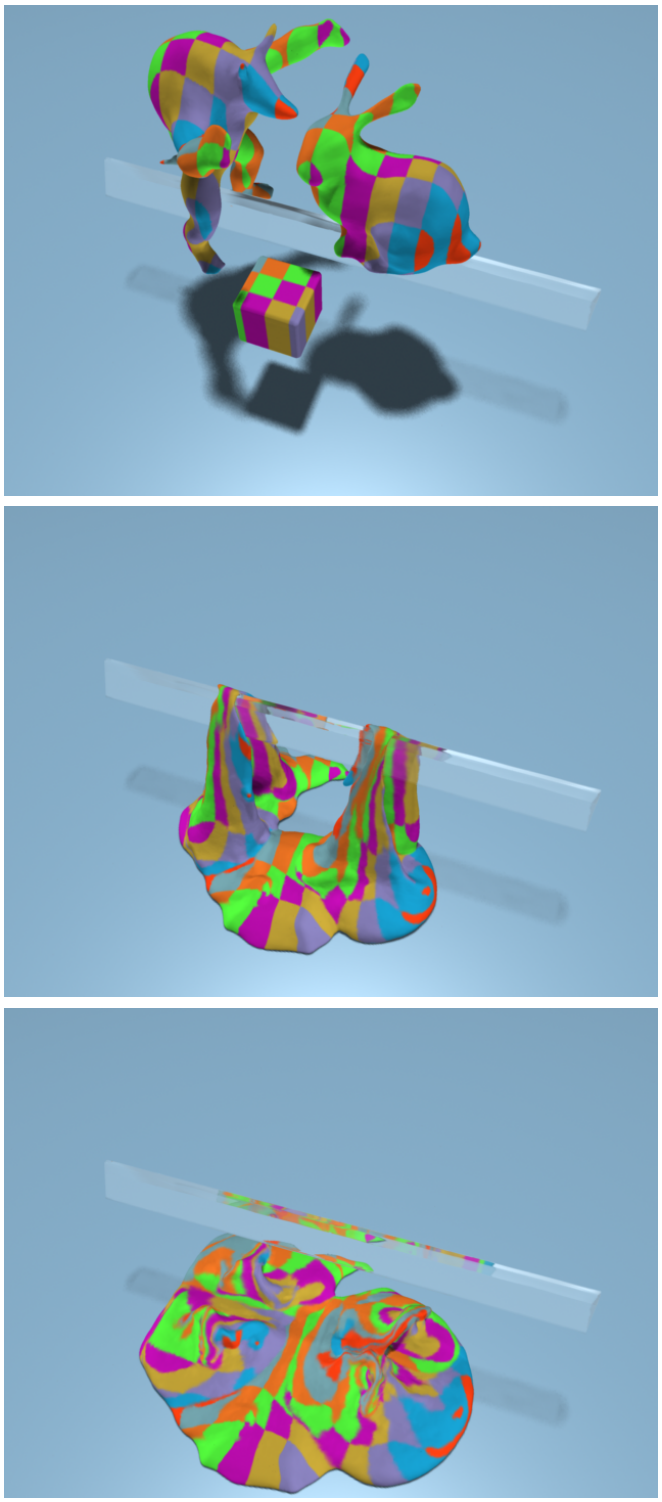
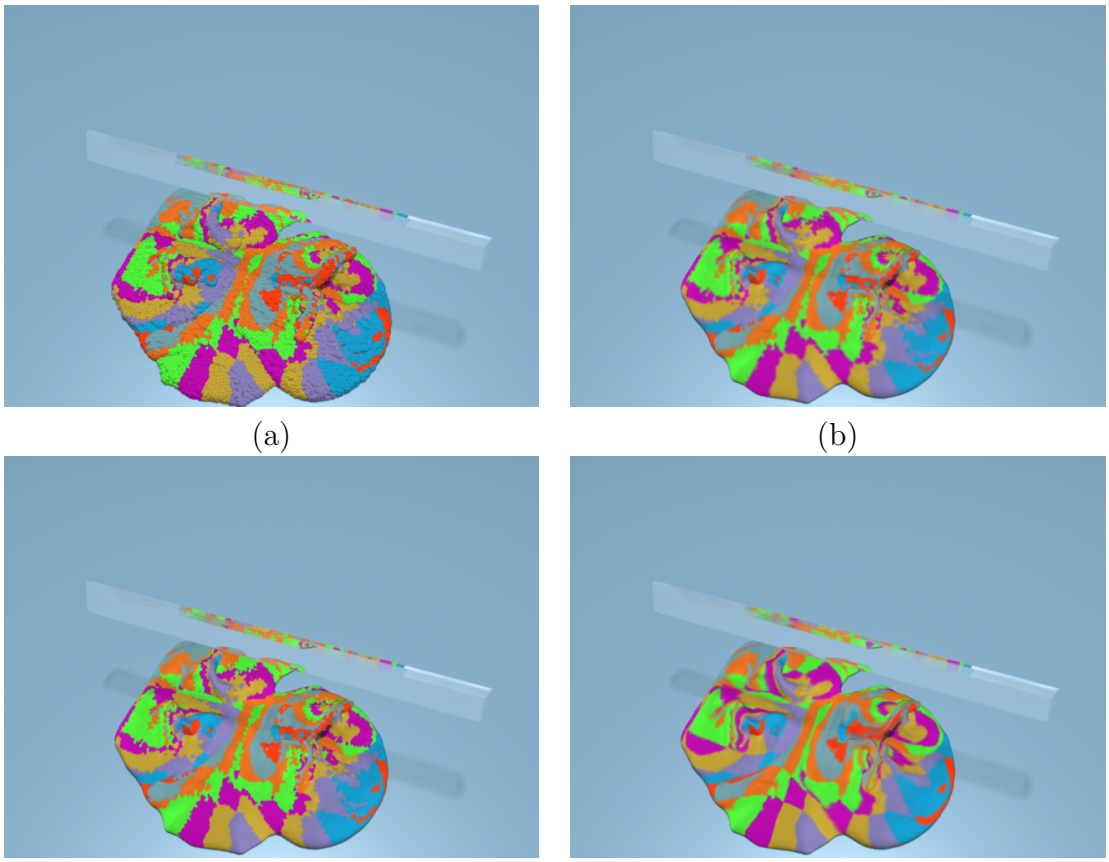


Figure 5.3: Three viscous figures with surface colors are dropped on a bar.



(a) (b) (c) (d)

Figure 5.4: Comparison between different surface color tracking approaches on the viscous figures animation.

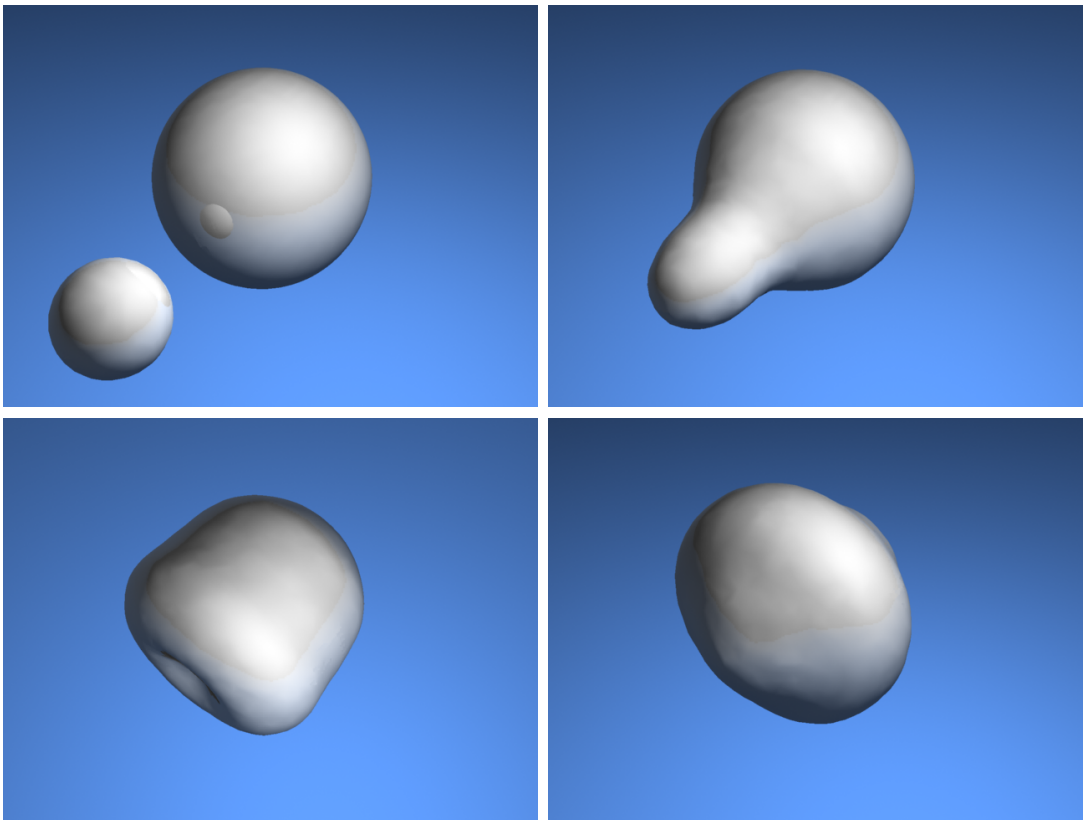


Figure 5.5: Two spheres in zero gravity that merge and exhibit surface waves.

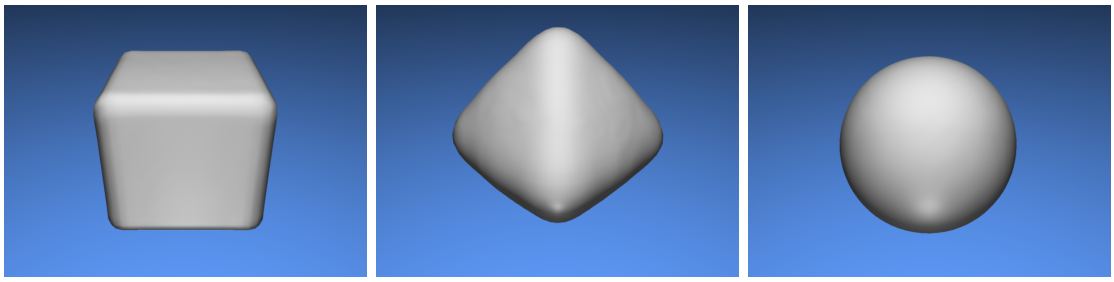


Figure 5.6: A cubic water drop that oscillates and settles into a sphere.

Table 5.1: Mesh resolution relative to the average particle spacing and average per frame timings (in seconds) for our simulation examples.

Example	Mesh	Tracking	Reconstruction	Simulation	Wave	Total
Cube (Figure 5.6)	0.5	0.54	0.24	1.15	N/A	1.93
Dumbbell (Figure 5.2)	0.5	0.54	0.20	2.67	N/A	3.42
Two Sphere (Figure 5.5)	0.5	1.767	5.08	3.69	11.41	21.70
Viscous Figures (Figure 5.3)	0.25	41.08	24.37	22.49	N/A	87.94
Water Crown (Figure 5.7)	0.37	46.87	38.54	63.32	24.57	173.31

version without the capillary ripples the surface around the water crown looks unnaturally flat. The images in Figure 5.7 include the surface dynamics for added realism. This was our largest simulation, and it required 200,000 particles to simulate.

Although our method incurs a per-timestep cost of the mesh vertex projection, it turns out that this projection does not adversely affect our running time. In a typical timestep, most of our projections require just one implicit surface evaluation. In contrast, a high-quality Marching Cubes surface extraction requires a root-finding step to locate the zero-value point along each edge, and this typically requires around ten implicit surface evaluations (see [Bloomenthal, 1994] for details). Although the Marching Cubes surface extraction is needed only per-frame and not per-timestep, the higher cost of the Marching Cubes is roughly equal to our per-timestep projection cost for the examples in this section. In the zero-gravity cube example of Figure 5.6, our surface tracking requires 0.78 seconds per frame, while the Marching cubes requires 0.67 seconds per frame.

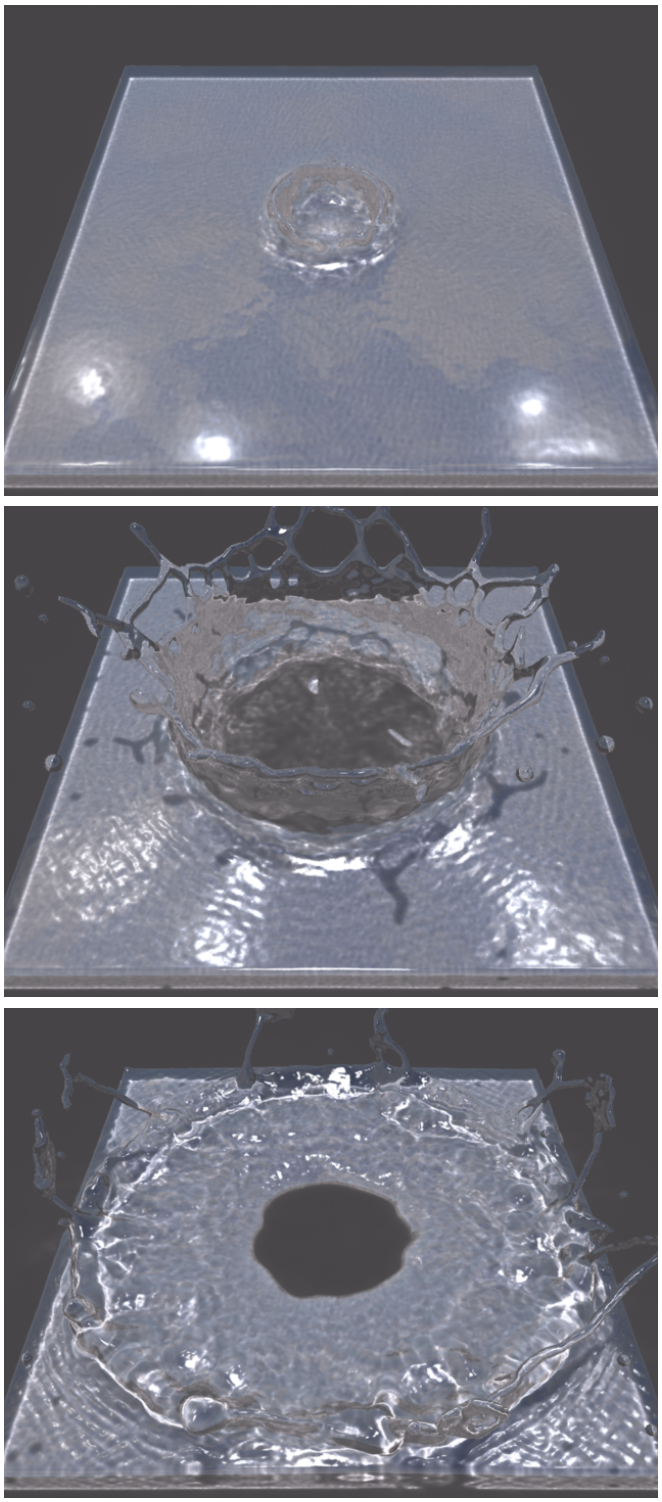


Figure 5.7: A drop falling into a shallow pool creates a water crown.

5.7 Limitations

There are a few limitations of our mesh-based surface tracking method. When we use our surface wave dynamics as a post-process, we have to prohibit the creation of new waves by highly curved regions. If we do not take this step, then many tiny waves are initiated and this quickly fills the entire surface with a noisy wave field. Although the projection step matches the explicit surface with the boundary of the particle volume, a small number of boundary particles with high velocity occasionally escape the mesh volume, and the mesh tracker loses them until they collide with the mesh again. We think that this can be corrected by evaluating the sign distance of each particle with respect to the explicit mesh, and skinning particles with a sphere mesh that are outside of the mesh. Another limitation of our method is that our surface dynamics do not exactly preserve the volume of the surface. This is not noticeable for large fluid volumes, but very small drops can be seen to slightly oscillate in volume. We think that this can be corrected by calculating the volume of each connected component and then re-scaling, but we have not yet implemented this.

5.8 Conclusion and Future Work

We have introduced a new way of tracking SPH fluids that carries an explicit surface mesh from one time step to the next. This approach allows us to create surface tension forces, carry properties on the surface such as colors, and add capillary waves to these dynamic surfaces.

There are several avenues for future work. One possibility is to re-sample SPH particles nearby the high resolution surface mesh to resolve the resolution mis-

match, instead of projecting the surface mesh onto the isosurface. We expect that re-sampling process can preserve thin and sharp features of the explicit mesh, while the mesh still produces smoother SPH surfaces as well as support surface tension and capillary waves. Our capillary wave model is orthogonal to the SPH application as long as the explicit mesh is used for the surface tracking. It will be interesting to apply capillary waves to other particle-based simulation or even Eulerian simulations as a post-process. In addition, we want to validate the capillary wave dynamics beyond visualization. In particular, we plan to investigate whether our capillary wave formulation is stable or not. Another challenge is to use our surface mesh to carry foam on the fluid surface in order to increase the visual realism of SPH fluids. It is also possible is to perform dynamic texture synthesis on SPH fluid surfaces, similar to the work of [Bargteil *et al.*, 2006] and [Kwatra *et al.*, 2007]. Finally, there are other particle-based methods used for animation besides SPH, and it is likely that our mesh tracking approach can be useful for these other simulators.

Appendix A

Analysis on Volume Shrinkage

Our new approach applies a volume smoothing to sampled particles, and pulls boundary particles inside. Typically, this results in the volume shrinkage when the fluid surface is reconstructed. In this appendix, we estimate the maximum distance between the original position and the smoothed position of a particle on a flat surface. Suppose that a particle i is located at the origin of an Eulerian coordinate system and the neighboring particles are continuously located in a hemisphere of radius r_i above the xy plane. Due to the spatial symmetry, the weighted mean is on the positive z-axis. We compute the length of the weighted mean $\|\mathbf{x}_i\|$ from (4.6) (4.8) using spherical coordinates. The numerator and the denominator of $\|\mathbf{x}_i^w\|$ are formulated as

$$\|\sum_j w_{ij}\mathbf{x}_j\| = 2\pi \int_0^{r_i} \int_0^{\frac{\pi}{2}} \sin 2\phi (r^3 - r^6 r_i^{-3}) dr d\phi, \tag{7}$$

and

$$\sum_j w_{ij} = 2\pi \int_0^{r_i} \int_0^{\frac{\pi}{2}} \sin \phi (r^2 - r^5 r_i^{-3}) dr d\phi. \tag{8}$$

A simple algebraic manipulation yields

$$\|\mathbf{x}_i^w\| = 9r_i/28. \tag{9}$$

Using (4.3), we estimate a bound between the particle position $\mathbf{x}_i$ and the updated position $\overline{\mathbf{x}}_i$ after the volume smoothing by

$$\|\overline{\mathbf{x}}_i - \mathbf{x}_i\| = \|\lambda(\mathbf{x}_i^w - \mathbf{x}_i)\| \leq 9\lambda r_i/28. \tag{10}$$

With $\lambda = 0.9$, $r_i = 2h_i$ used in our examples, we obtain $\|\overline{\mathbf{x}}_i - \mathbf{x}_i\| \leq 0.58h_i$. Since the extracted isosurface encloses $\overline{\mathbf{x}}_i$'s, the maximal distance from the reconstructed surface to the simulation particles is less than $0.58h_i$. This analysis shows that the volume shrinkage effect does not cause significant visual artifacts, since h_i is small compared to the size of simulation domain. In fact, the volume shrinkage of our method results in *surface erosion* which often generates thin features. The surface erosion method proposed in [Williams, 2008] has a risk of creating non-manifold surfaces such as self-intersecting surfaces. In contrast, our new approach produces an eroded surface as a manifold, since we erode particles instead of a polygonal mesh.

Appendix B

Source Code

This appendix contains the verbatim source code used to compute anisotropic kernels for the particles samples. First, we have `StretchParticles`, the top level function for determining anisotropic kernels of each particle.

```
void StretchParticles() {
  enum {kTensorScale=2, kMatrixScale=1600};

  static const TS radius_tensor =
    interaction_radius_*kTensorScale;
  static const int neighbor_size_max =
    4*kPi*cube(radius_tensor)*rest_density_/(3*particle_mass_);
  static const int reserved_cell_size =
    3*cube(radius_tensor)*rest_density_/particle_mass_;

  static SpatialPartitionUniGrid<TV>
    uni_grid(bounding_box_,radius_tensor,reserved_cell_size);
  static SpatialPartitionUniGrid<TV>::PairListType pair_list;

  uni_grid.AssignPointList(particles_.x_);

  particles_aniso_.Resize(particles_.Size());
  particles_aniso_.r_ = particles_.r_;

  for(size_t i=0; i<particles_.Size(); ++i) {
    TV center;
```

```
uni_grid.QueryPairCenter(particles_.x_[i], particles_.x_,
   pair_list, center);

TV center_of_mass(TV::Zero());
TS total_mass(0);
vector<TS> neighbor_weight;
neighbor_weight.reserve(neighbor_size_max);
int small_neighbor(0);

// check inner volume particles
// check density and the neighbor size.
static const size_t neighbor_size_e =
  neighbor_size_max*0.9;
if(OPT && pair_list.size() > neighbor_size_e) {
  particles_aniso_.x_[i] = particles_.x_[i];
  particles_aniso_.scale_[i].setIdentity();
  particles_aniso_.rot_[i].setIdentity();
  particles_aniso_.G_[i].setIdentity();
  particles_aniso_.scale_min_sqr_[i] = 1;

  continue;
}

for(size_t k=0; k<pair_list.size(); ++k) {
  const size_t id = pair_list[k].first;

  TS weight;
  // check connected component info
  if(CC && particles_.comp_[i] != particles_.comp_[id])
    weight = 0;
  else
    weight = ComputePCAKernel(pair_list[k].second*0.5);

  center_of_mass += particles_.x_[id]*weight;
  total_mass     += weight;
  neighbor_weight.push_back(weight);

  if(weight < 0.5) small_neighbor++;
}
center_of_mass /= total_mass;
particles_aniso_.x_[i] =
  particles_.x_[i]*0.1+center_of_mass*0.9;

TM D;
```

```
    // compute a covariance tensor D
    for(size_t k=0; k<pair_list.size(); ++k) {
      const size_t id = pair_list[k].first;
      const TV d = particles_.x_[id] - center_of_mass;
      D += neighbor_weight[k]*Outer_Product(d);
    }
    D /= total_mass;

    TV& S = particles_aniso_.scale_[i];
    TM& V = particles_aniso_.rot_[i];

    // do the SVD analysis
    EigenSolve(D, S, V);

    // prevent too much deformation
    static const TS maxRatio = 4;
    if(S(0)> maxRatio*S(2) ) {
      S(2) = S(0)/maxRatio;
      S(1) = max(S(2), S(1));
    }
    S *= kMatrixScale;

    if(small_neighbor < 4)
      S = TV(0.5,0.5,0.5);

    particles_aniso_.G_[i] =
      V*TV(1/S(0),1/S(1),1/S(2)).asDiagonal()*V.transpose();
  }
}
```

EigenSolve and its sub-level functions implement a matrix decomposition algorithm that is specialized for symmetric 3 real matrices.

```
void EigenSolve(const TM& D, TV& S, TM& V) {
  // D is symmetric
  // S is a vector whose elements are eigenvalues
  // V is a matrix whose columns are eigenvectors
  S = EigenValues(D);

  TV V0,V1,V2;
```

```
  if(S(0) - S(1) > S(1) - S(2)) {
    V0 = EigenVector(D,S(0));

    if(S(1) - S(2) < numeric_limits<TS>::epsilon()) {
      V2 = Orthogonal(V0);
    } else {
      V2 = EigenVector(D,S(2));
      V2 -= V0*V0.dot(V2);
      V2.normalize();
    }

    V1 = V2.cross(V0);
  } else {
    V2 = EigenVector(D,S(2));

    if(S(0) - S(1) < numeric_limits<TS>::epsilon()) {
      V1 = Orthogonal(V2);
    } else {
      V1 = EigenVector(D,S(1));
      V1 -= V2*V2.dot(V1);
      V1.normalize();
    }

    V0 = V1.cross(V2);
  }

  V << V0(0), V1(0), V2(0),
       V0(1), V1(1), V2(1),
       V0(2), V1(2), V2(2);
}

// D is symmetric, S is an eigen value
TV EigenVector(const TM& D, const TS S) {
  // Compute a cofactor matrix of D - sI.
  TM F = D;
  F(0,0) -= S; F(1,1) -= S; F(2,2) -= S;

  // Use an upper triangle
  TM C;
  C(0,0) = F(1,1)*F(2,2)-sqr(F(1,2));
  C(0,1) = F(1,2)*F(0,2)-F(0,1)*F(2,2);
  C(0,2) = F(0,1)*F(1,2)-F(1,1)*F(0,2);
  C(1,1) = F(0,0)*F(2,2)-sqr(F(0,2));
  C(1,2) = F(0,1)*F(0,2)-F(0,0)*F(1,2);
```

```
  C(2,2) = F(0,0)*F(1,1)-sqr(F(0,1));

  // Get a column vector with a largest norm (non-zero).
  TS norm[3];
  norm[0] = sqr(C(0,0))+sqr(C(0,1))+sqr(C(0,2));
  norm[1] = sqr(C(0,1))+sqr(C(1,1))+sqr(C(1,2));
  norm[2] = sqr(C(0,2))+sqr(C(1,2))+sqr(C(2,2));

  const int index =
    IndexOfLargestValue(norm[0],norm[1],norm[2]);

  TV V;

  // special case
  if(norm[index] == 0) {
    V(0) = 1; V(1) = 0; V(2) = 0; return V;
  } else if(index == 0) {
    V(0) = C(0,0); V(1) = C(0,1); V(2) = C(0,2);
  } else if(index == 1) {
    V(0) = C(0,1); V(1) = C(1,1); V(2) = C(1,2);
  } else {
    V(0) = C(0,2); V(1) = C(1,2); V(2) = C(2,2);
  }

  return V.normalized();
}

// D is symmetric
TV EigenValues(const TM& D) {
  static const TS one_third(1/(TS)3.0);
  static const TS one_sixth(1/(TS)6.0);
  static const TS three_sqrt(sqrt((TS)3.0));

  const TS m = one_third*D.trace();

  // K is D - I*diag(S)
  const TS K00 = D(0,0)-m;
  const TS K11 = D(1,1)-m;
  const TS K22 = D(2,2)-m;

  const TS K01s = sqr(D(0,1));
  const TS K02s = sqr(D(0,2));
  const TS K12s = sqr(D(1,2));
```

```
  const TS q =
    0.5*(K00*(K11*K22-K12s)-K22*K01s-K11*K02s)+
    D(0,1)*D(1,2)*D(2,0);
  const TS p =
    one_sixth*(sqr(K00)+sqr(K11)+sqr(K22)+2*(K01s+K02s+K12s));

  const TS p_sqrt = sqrt(p);

  const TS tmp = cube(p) - sqr(q);
  const TS phi = one_third*atan2(sqrt(max((TS)0,tmp)),q);

  const TS phi_c = cos(phi);
  const TS phi_s = sin(phi);

  const TS sqrt_p_c_phi = p_sqrt*phi_c;
  const TS sqrt_p_3_s_phi = p_sqrt*three_sqrt*phi_s;

  TV S;

  S(0) = m+2*sqrt_p_c_phi;
  S(1) = m-sqrt_p_c_phi-sqrt_p_3_s_phi;
  S(2) = m-sqrt_p_c_phi+sqrt_p_3_s_phi;

  Sort(S(0),S(1),S(2));

  return S;
}
```

Bibliography

[Adams and Wicke, 2009] Bart Adams and Martin Wicke. Meshless approximation methods and applications in physics based modeling and animation. In *Eurographics 2009 Tutorials*, pages 213–239, 2009.

[Adams *et al.*, 2007] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3):48, 2007.

[Bargteil *et al.*, 2005] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. In *SIGGRAPH '05: Proceedings of the ACM SIGGRAPH 05 electronic art and animation catalog*, pages 238–238, New York, NY, USA, 2005. ACM Press.

[Bargteil *et al.*, 2006] Adam W. Bargteil, Funshing Sin, Jonathan E. Michaels, Tolga G. Goktekin, and James F. O'Brien. A texture synthesis method for liquid animations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Sept 2006.

[Bargteil *et al.*, 2007] A.W. Bargteil, C. Wojtan, J.K. Hodgins, and G. Turk. A finite element method for animating large viscoplastic flow. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 26(3):16–1, 2007.

[Becker and Teschner, 2007] M. Becker and M. Teschner. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217. Eurographics Association, 2007.

[Bergou *et al.*, 2006] M. Bergou, M. Wardetzky, D. Harmon, D. Zorin, and E. Grinspun. A quadratic bending model for inextensible surfaces. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 227–230. Eurographics Association, 2006.

[Blinn, 1982] J.F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982.

[Bloomenthal, 1994] J. Bloomenthal. An implicit surface polygonizer. *Graphics gems IV*, 1:324–349, 1994.

[Brochu and Bridson, 2009] Tyson Brochu and Robert Bridson. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493, 2009.

[Brochu *et al.*, 2010] Tyson Brochu, Christopher Batty, and Robert Bridson. Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph.*, 29(4):1–9, 2010.

[Carchidi, 1986] M Carchidi. A method for finding the eigenvectors of an n x n matrix corresponding to eigenvalues of multiplicity one. *Am. Math. Monthly*, 93:647–649, October 1986.

[Clavet *et al.*, 2005] S. Clavet, P. Beaudoin, and P. Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228. ACM, 2005.

[Desbrun and Cani, 1998] Mathieu Desbrun and Marie-Paule Cani. Active implicit surface for animation. In Wayne A. Davis, Kellogg S. Booth, and Alain Fournier, editors, *Graphics Interface 1998, June, 1998*, pages 143–150, Vancouver, BC, Canada, June 1998. Canadian Human-Computer Communications Society. Published under the name Marie-Paule Cani-Gascuel.

[Desbrun *et al.*, 1999] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Dinh *et al.*, 2001] H.Q. Dinh, G. Turk, and G. Slabaugh. Reconstructing surfaces using anisotropic basis functions. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 606–613. IEEE, 2001.

[Du *et al.*, 2006] J. Du, B. Fix, J. Glimm, X. Jia, X. Li, Y. Li, and L. Wu. A simple package for front tracking. *Journal of Computational Physics*, 213(2):613–628, 2006.

[Enright *et al.*, 2002] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[Enright *et al.*, 2005] D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-Lagrangian particle level set method. *Computers & Structures*, 83(6-7):479–490, 2005.

[Goktekin *et al.*, 2004] Tolga G. Goktekin, Adam W. Bargteil, and James F. O'Brien. A method for animating viscoelastic fluids. *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2004)*, 23(3):463–468, 2004.

[Hong and Kim, 2005] Jeong-Mo Hong and Chang-Hun Kim. Discontinuous fluids. *ACM Trans. Graph.*, 24:915–920, July 2005.

[Hu and Adams, 2006] XY Hu and NA Adams. A multi-phase SPH method for macroscopic and mesoscopic flows. *Journal of Computational Physics*, 213(2):844–861, 2006.

[Kalaiah and Varshney, 2003] A. Kalaiah and A. Varshney. Statistical point geometry. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, page 115. Eurographics Association, 2003.

[Kang *et al.*, 2000] M. Kang, R.P. Fedkiw, and X.D. Liu. A boundary condition capturing method for multiphase incompressible flow. *Journal of Scientific Computing*, 15(3):323–360, 2000.

[Kim *et al.*, 2009] Doyub Kim, Oh-Young Song, and Hyeong-Seok Ko. Stretching and wiggling liquids. *ACM Transactions on Graphics*, 28(5):120, 2009.

[Kopp, 2008] Joachim Kopp. Efficient numerical digonalization of hermitian 3 x 3 matrices. *International Journal of Modern Physics C*, 19(03):523–548, 2008.

[Koren and Carmel, 2003] Yehuda Koren and Liran Carmel. Visualization of labeled data using linear transformations. In *Proceedings of the Ninth annual IEEE conference on Information visualization*, INFOVIS'03, pages 121–128, Washington, DC, USA, 2003. IEEE Computer Society.

[Kwatra *et al.*, 2007] V. Kwatra, D. Adalsteinsson, T. Kim, N. Kwatra, M. Carlson, and M. Lin. Texturing fluids. *IEEE transactions on visualization and computer graphics*, pages 939–952, 2007.

[Liu and Liu, 2003] G. R. Liu and M. B. Liu. *Smoothed particle hydrodynamics: a meshfree particle method*. World Scientific Publishing, 2003.

[Liu *et al.*, 2006] M. B. Liu, G. R. Liu, and K. Y. Lam. Adaptive smoothed particle hydrodynamics for high strain hydrodynamics with material strength. *Shock Waves*, 15:21–29, March 2006.

[Lorensen and Cline, 1987] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM.

[Losasso *et al.*, 2004] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (TOG)*, 23(3):457–462, 2004.

[Lucy, 1977] L Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical J.*, 82:1013–1024, 1977.

[Meyer *et al.*, 2002] M. Meyer, M. Desbrun, P. Schröder, and A.H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and mathematics*, 3(7):34–57, 2002.

[Monaghan and Gingold, 1977] J Monaghan and R.A. Gingold. Smoothed particle hydrodynamics-theory and application to nonspherical starts. *Mon. Not. R. Astron. Soc.*, 181:357, 1977.

[Monaghan, 1994] J. Monaghan. Simulating free surface flows with sph. *J. Comput. Phys.*, 110(2):399–406, 1994.

[Monaghan, 2005] J Monaghan. Smoothed particle hydrodynamics. *Rep. Prog. Phys.*, 68:1703–1759, 2005.

[Müller *et al.*, 2003] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Müller, 2009] M. Müller. Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 237–245. ACM, 2009.

[Osher and Sethian, 1988] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[Owen *et al.*, 1998] J.M. Owen, J.V. Villumsen, P.R. Shapiro, and H. Martel. Adaptive smoothed particle hydrodynamics: Methodology. ii. *The Astrophysical Journal Supplement Series*, 116(2):155–209, 1998.

[Premoze *et al.*, 2003] Simon Premoze, Tolga Tasdizen, James Bigler, Aaron Lefohn, and Ross T. Whitaker. Particle-based simulation of fluids. In *Proceedings of Eurographics 2003*, pages 401–410, 2003.

[Rasmussen *et al.*, 2004] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 193–202. Eurographics Association, 2004.

[Safonova and Szymczak, 2003] J.R.A. Safonova and A. Szymczak. Edgebreaker on a corner table: A simple technique for representing and compressing triangulated surfaces. *Hierarchical and geometrical methods in scientific visualization*, page 41, 2003.

[Sifakis *et al.*, 2007] E. Sifakis, T. Shinar, G. Irving, and R. Fedkiw. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 81–90. Eurographics Association, 2007.

[Sin *et al.*, 2009] F.S. Sin, A.W. Bargteil, and J.K. Hodgins. A point-based method for animating incompressible flow. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2009.

[Smith, 1961] O.K. Smith. Eigenvalues of a symmetric 3× 3 matrix. *Communications of the ACM*, 4(4):168, 1961.

[Solenthaler and Pajarola, 2008] Barbara Solenthaler and Renato Pajarola. Density contrast sph interfaces. In *Proceedings of ACM SIGGRAPH / EG Symposium on Computer Animation*, pages 211–218, 2008.

[Stam, 1999] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Stern and Desbrun, 2006] A. Stern and M. Desbrun. Discrete geometric mechanics for variational time integrators. In *SIGGRAPH '06: ACM SIGGRAPH 2006 courses*, pages 75–80, New York, NY, USA, 2006. ACM.

[Taubin, 2000] G. Taubin. Geometric signal processing on polygonal meshes. *Eurographics State of the Art Reports*, 4(3), 2000.

[Thürey *et al.*, 2010] Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. In *SIGGRAPH '10: ACM SIGGRAPH 2010 papers*, pages 1–10, New York, NY, USA, 2010. ACM.

[Wang *et al.*, 2007] H. Wang, G. Miller, and G. Turk. Solving general shallow wave equations on surfaces. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–238. Eurographics Association, 2007.

[Williams, 2008] Brent Warren Williams. Fluid surface reconstruction from particles. Master's thesis, The University of British Columbia, Canada, February 2008.

[Wojtan and Turk, 2008] Chris Wojtan and Greg Turk. Fast viscoelastic behavior with thin features. *ACM Trans. Graph.*, 27(3):1–8, 2008.

[Wojtan *et al.*, 2009] C. Wojtan, N. Thürey, M. Gross, and G. Turk. Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 papers*, page 76. ACM, 2009.

[Wojtan *et al.*, 2010] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph.*, 29(4):1–8, 2010.

[Yu and Turk, 2010] Jihun Yu and Greg Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 217–225, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[Zhang, 2010] M. Zhang. Simulation of surface tension in 2D and 3D with smoothed particle hydrodynamics method. *Journal of Computational Physics*, 229:7238–7259, 2010.

[Zhu and Bridson, 2005] Y. Zhu and R. Bridson. Animating sand as a fluid. In *ACM SIGGRAPH 2005 Papers*, page 972. ACM, 2005.