Liquid Simulation with Mesh-based Surface Tracking

Siggraph 2011 Submitted Sample Course Notes
Half Day Course

Course organizer:

Chris Wojtan
Institute of Science and Technology Austria

Lecturers:

Matthias Müller-Fischer
NVIDIA

Chris Wojtan
Institute of Science and Technology Austria

Tyson Brochu
University of British Columbia

Course Description

Animating detailed liquid surfaces has continually been a challenge for computer graphics researchers and visual effects artists. Over the past few years, a strong trend has emerged among researchers in this field towards mesh-based surface tracking in order to synthesize extremely detailed liquid surfaces as efficiently as possible. This course will provide attendees with a solid understanding of the steps necessary to create a fluid simulator with a mesh-based liquid surface.

The course will begin with an overview of several existing liquid surface tracking techniques, discussing the pros and cons of each method. We will then provide instructions and a simple demonstration on how to embed a triangle mesh into a finite-difference-based fluid simulator. Once this groundwork has been laid, the next section of the course will stress the importance of surface quality and review techniques for maintaining a high quality triangle mesh. Afterward, we will describe several methods for allowing the liquid surface to merge together or break apart. The final section of this course showcase the benefits and further applications of a mesh-based liquid surface, highlighting state-of-the-art methods for tracking colors and textures, maintaining liquid volume, preserving small surface features, and simulating realistic surface tension waves.

Level of Difficulty: Advanced.

Intended Audience

This course is intended for both researchers and developers in industry who want to implement and have a solid understanding of the state of the art in fluid simulation for computer animation.

Preprequisites

A familiarity with Eulerian fluid simulation techniques for computer animation. The necessary background material can be found in the book Fluid Simulation for Computer Graphics by Robert Bridson (available from A K Peters), or the SIGGRAPH 2007 course notes on Fluid Simulation by Robert Bridson and Matthias Mller-Fischer. In addition, a passing knowledge of basic triangle mesh algorithms like subdivision and edge collapses will be useful.

About the Lecturers

Chris Wojtan
Institute of Science and Technology Austria
wojtan@ist.ac.at
http://pub.ist.ac.at/group_wojtan/

Dr. Chris Wojtan is an assistant professor at the Institute of Science of Technology Austria (IST Austria), where he is establishing a computer graphics lab with a research focus on physically-based animation, geometric modelling, and numerical techniques. His computer graphics contributions include methods for animating detailed viscoplastic materials, several techniques for controlling physics simulations, and an algorithm for efficiently computing topological changes in deforming triangle meshes. His research into mesh-based fluid surface tracking has helped produce extremely detailed liquid surface animations, allowing arbitrarily thin features and detailed crown splashes. Prior to his work at IST Austria, Chris received his Ph.D. in Computer Science from the Georgia Institute of Technology in 2010, and he worked as a visiting scientist at Carnegie Mellon University and ETH Zürich.

Matthias Müller-Fischer
NVIDIA
matthias@mueller-fischer.com
http://www.matthiasmueller.info/

Matthias Müller-Fischer is a principal software engineer at nvidia and head of the PhysX research team. He received his PhD in atomistic simulation of polymers in 1999 from ETH Zürich. During his post-doc with the MIT Computer Graphics Group (1999-2001) he changed fields to macroscopic real-time simulations. Since then, his main research field has been the development of fast and robust physically based simulation methods for computer games. He has published key papers in computer graphics on real-time particle based water simulation and visualization as well as finite element and geometry based soft body, cloth and fracture simulation. In 2002, he co-founded the game middleware company NovodeX (acquired in 2004 by AGEIA Technologies, Inc.), where he was head of research and responsible for the extension of the physics simulation library PhysX by innovative new features. He has been with nvidia since the acquisition of AGEIA in early 2008.

Tyson Brochu
University of British Columbia
tbrochu@cs.ubc.ca
www.cs.ubc.ca/ tbrochu/

Tyson Brochu is a PhD candidate in the department of Computer Science at the University of British Columbia. His research focuses on purely mesh-based representations of surfaces undergoing extreme deformations and changes in topology, with a focus on tracking liquid surfaces. Additionally, he developed a novel approach to constructing fluid simulation elements, which captures fine surface details present in mesh-based surface tracking. His work has been published in scientific computing and computer graphics journals. Collaboration with VFX studios such as Weta Digital and Digital Domain informs his research and provides industry practitioners with access to cutting-edge applied research.

Course Overview

5 minutes: Introduction and welcome
Chris Wojtan

25 minutes: Liquid surface tracking review
Matthias Müller-Fischer
- Review of Eulerian surface tracking methods
- Review of point-based Lagrangian surface tracking methods
- Problems with previous methods that can be overcome with a Lagrangian surface mesh
- Benefits of a mesh-based Lagrangian representation (Examples with fixed topology)

30 minutes: Embedding a surface mesh into an Eulerian fluid simulation
Matthias Müller-Fischer
- Mesh-based surface tracking in a simulation using regular cubic grid cells
- Boundary conditions for solids and the free surface
- Live demonstration
- Problems with differing surface and simulation resolutions
- Adapting Eulerian grid geometry to better fit the surface mesh

15 minute break

15 minutes: Maintaining surface mesh quality
Chris Wojtan
- Why? Visual artifacts, memory limitations, and stable computation
- Quality measures
- Operations: Edge splits, edge flips, edge collapses, and null-space smoothing

50 minutes: Topology changes
Chris Wojtan
- Why do we need topology changes?
- Topological operations on the mesh itself
- Global grid-based re-meshing of the surface
- Local grid-based re-meshing of the surface (marching cubes & convex hulls)

50 minutes: Advantages of a mesh surface
Tyson Brochu
- Color and texture tracking
- Volume preservation (both global and local)
- Preserved surface details
- Thin features
- Surface tension (several methods)
- The future of mesh-based surface tracking

5 minutes: Conclusion
Chris Wojtan

The remaining pages are sample course notes. They have been assembled primarily from the lecturers' personal experiences while performing related research [4, 6, 3, 5, 17, 27, 25, 23, 26, 24].

These notes are meant to represent the level of quality and intellectual depth of the material that we plan to present at SIGGRAPH 2011.

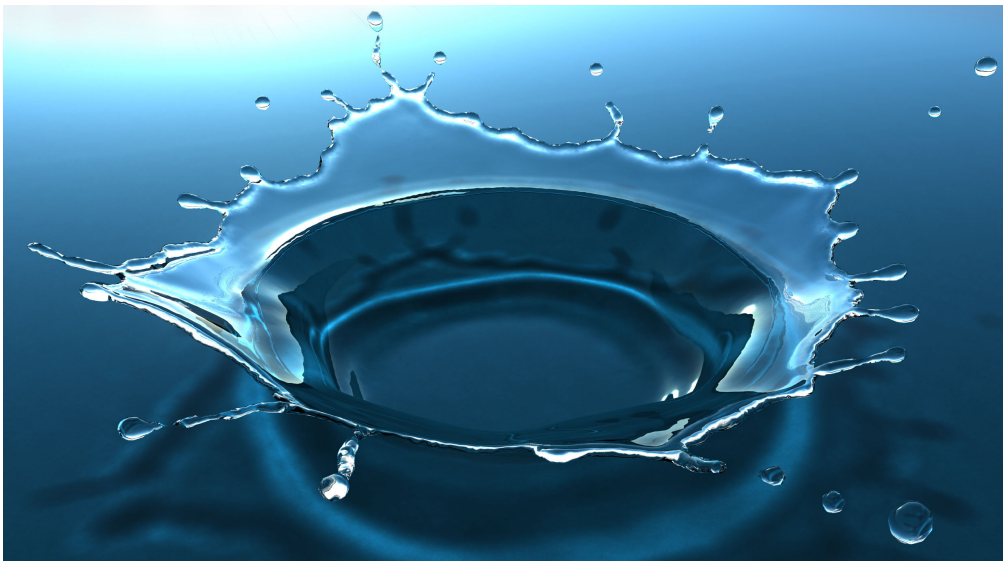

Figure 1: This course will provide all of the imlpementation details necessary to implement a fluid simulator capable of preserving thin liquid sheets and efficiently exhibiting detailed surface tension behavior.

1 Introduction

Animating detailed liquid surfaces is continually an important challenge for computer graphics researchers and visual effects artists. Until recently, purely Lagrangian techniques (like triangle meshes or particles) for simulating the liquid surface had been limited by several challenging problems. The difficulties associated with tangling surfaces were quite unattractive, and the complicated mesh surgery techniques necessary to split and merge surfaces were often plagued with robustness problems and promptly dismissed. On the other hand, Eulerian techniques (such as the level set method) were practically bulletproof and quite simple to implement. As a result, surface tracking techniques based on dynamic implicit surfaces became extremely popular and helped revolutionize the field of physics-based computer animation.

Over the past few years, however, several of the seemingly insurmountable problems associated with dynamic explicit surfaces have become more tractable — steady research in engineering, applied mathematics, computer vision, computational geometry, and computer graphics has slowly chipped away at the problem. Methods now exist for efficiently manipulating complex meshes while preventing self-collisions, and recent advances have allowed for the fast and robust computation of topological changes on deforming meshes. As a result of this progress, dynamic explicit meshes can now be integrated in a fluid simulation environment.

While explicit meshes may require more care for certain operations than standard implicit surface methods, dynamic meshes have the potential to become the new standard tool for creating beautiful fluid simulations. By default, mesh-based dynamic surfaces preserve an exquisite amount of visual detail, and they naturally provide a straightforward computational

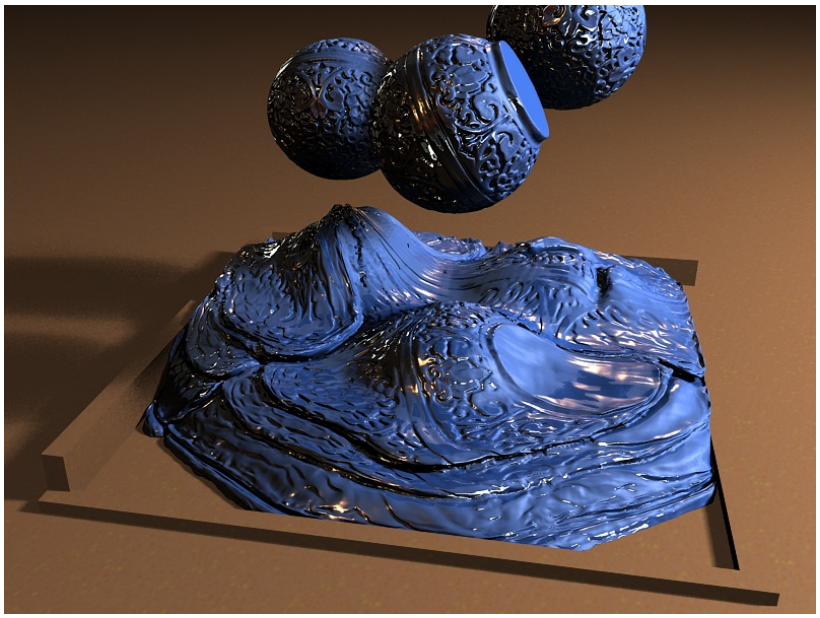

Figure 2: This course will explain how to couple a grid-basd fluid solver with a mesh-based surface in order to simulate materials with extremely detailed surface features.

environment for simulating differential equations on a moving surface. In addition, although care is required to robustly handle topological changes, we now have the ability to *control* the topology of a dynamic surface, allowing for the persistence of thin liquid features and the efficient formation of tiny water droplets.

As a result of these compelling advantages of dynamic explicit surfaces, a trend has emerged among computer graphics researchers towards simulating liquids with mesh-based surface tracking. In the interest of keeping researchers and special effects developers up to date with the latest technology, this course aims to provide attendees with a solid understanding of the steps necessary to create their own fluid simulator with a mesh-based liquid surface.

The course will begin with an overview of several existing liquid surface tracking techniques, discussing the pros and cons of each method. We will then provide instructions and a simple demonstration on how to embed a triangle mesh into a finite-difference-based fluid simulator. Once this groundwork has been laid, the next section of the course will stress the importance of surface quality and review techniques for maintaining a high quality triangle mesh. Afterward, we will describe several methods for allowing the liquid surface to merge together or break apart. The final section of this course showcase the benefits and further applications of a mesh-based liquid surface, highlighting state-of-the-art methods for tracking colors and textures, maintaining liquid volume, preserving small surface features, and simulating realistic surface tension waves.

2 Embedding a surface mesh into an Eulerian fluid simulation

This section lays the groundwork for a liquid simulation with mesh-based surface tracking. We will begin with a standard fluid simulation framework used frequently throughout computer animation [22, 9, 10, 2], and then we will explain how to embed a triangle mesh inside of this fluid simulation. The velocity of the fluid simulation is used to advect the vertices of the triangle mesh, and the way that the mesh partitions space into "inside" and "outside" regions will be used to define the domain of the fluid simulation. The result is a mesh that splashes and sloshes around under the influence of fluid forces.

2.1 Physical Model

We start by building upon a common technique for simulating fluid dynamics in computer graphics: We aim to simulate the partial differential equations which govern subsonic fluid flow, known as the incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

where $\mathbf{u}$ is the velocity of the fluid, t is time, ρ is the density of the fluid, p is the fluid's pressure, ν is the fluid's viscosity, and $\mathbf{g}$ is the acceleration due to gravity. We store these variables on a regular hexahedral grid, with the pressure values stored at the centers of each grid cell and the velocity components stored on the faces of each grid cell. The details of storing these variables are discussed further in [2].

We can think of the Equation 1 as a collection of competing accelerations that push around the fluid. This equation governs the momentum of the fluid, and we will refer to it as the *momentum equation*. Equation 2 is a constraint on the momentum equation that forces the fluid's velocity field to have a divergence of zero ("divergence-free"). When the fluid's velocity has no divergence, it is considered *incompressible*, and there are no sources or sinks of momentum in this system. Additionally, as long as the density of the fluid is constant, this divergence-free constraint implies that the fluid must conserve mass and volume.

To simulate the fluid, we can break up the momentum equation into a series of terms using a technique called *operator-splitting*. This effectively means that each term on the right hand side of Equation 1 can be applied independently, one after the other. To account for the gravity term, we simply add a constant downward acceleration to our velocity field. To account for the viscosity term, we solve the diffusion equation $\frac{\partial \mathbf{u}}{\partial t} = \nu \Delta \mathbf{u}$. The advection term $-(\mathbf{u} \cdot \nabla)\mathbf{u}$, has been studied extensively in the applied mathematics and computer graphics literature, and we can use any of several techniques ([22, 9, 14, 16]). We used a higher-order variant on semi-Lagrangian advection known as unconditionally stable MacCormack advection [20] for most of our simulations.

Because the pressure field p is unknown at the beginning of each time step, we can use it as a Lagrange multiplier to satisfy the the divergence-free constraint (Equation 2). We solve the remaining system of equations using the Conjugate Gradient method, giving us the final velocity field $\mathbf{u}$. See [2] for details.

2.2 Embedded Surface Mesh

In order to simulate a liquid, we also need to simulate the motion of the the visible surface at the fluid boundary. This is where our mesh-based surface tracking comes in —- we choose to use an explicit triangle mesh for this fluid surface. The mesh is simply a collection of vertices connected by triangles. This surface mesh partitions space into two regions: a portion of space that is "inside" the simulated liquid, and a vast region of space that is "outside" of the liquid. Because of this clean partition, we use this surface mesh to mark the simulation domain in our fluid solver. In other words, the surface mesh tells the fluid simulation which cells should be simulated as fluid and which should not, depending on whether the cells lie inside or outside of the mesh surface. As such, we insist that this surface is manifold and closed, in order to have a clear definition of what regions are "inside" of the simulated fluid, and which regions are "outside."

2.2.1 Updating the Surface Mesh

Once we have solved for the velocity field $\mathbf{u}$ at the end of each time step, use it to move the fluid surface. Each surface vertex is implicitly associated with a velocity, because it has a unique position $\mathbf{x}_i$ and the velocity field is a function of space, $\mathbf{u}(\mathbf{x})$. In practice, $\mathbf{u}$ is only defined at discrete regular intervals in our fluid grid, but we can use simple interpolation techniques to fit a smoothly-varying function to this data. In practice, we use tri-linear interpolation to find a velocity for a given position in space.

At this point, we assume that no topological changes occur during the motion of the surface, so we do not need to update any triangle connectivity; we simply need to solve the ordinary differential equation

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{u}_i \tag{3}$$

for each surface vertex i with position $\mathbf{x}_i$ and velocity $\mathbf{u}_i$. We can rearrange this equation to get the integral

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i^{\text{old}} + \int \mathbf{u}_i dt \tag{4}$$

The simplest technique for performing this numerical integration for each surface vertex is to use the forward Euler method:

$$\mathbf{x}_i^{\text{new}} = \mathbf{x}_i^{\text{old}} + \mathbf{u}_i \Delta t \tag{5}$$

The forward Euler method works well enough for most purposes, however we choose to use a fourth order Runge Kutta method (RK4) [19] because it performed best in analytical tests (Figure 3, for example) and it is relatively inexpensive to compute.

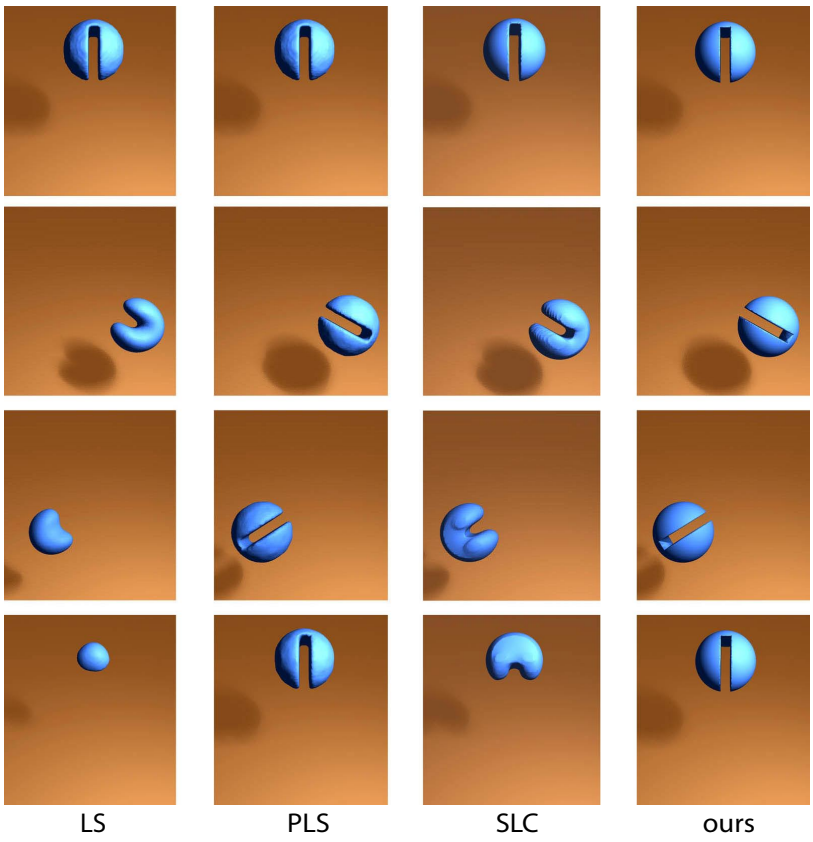

Figure 3: How does our mesh-based surface tracking scheme compare with other common surface trackers? This figure compares our mesh-based surface (far right column) with three other schemes (from left to right: level set method, particle level set method, and semi-Lagrangian contouring). The top row shows the initial conditions of the Zalesak sphere test. As time goes on (subsequent lower rows), the notched sphere rotates. The analytical solution to this problem is that the final condition after one rotation should be idential to its initial shape and position. The mesh-based surface tracker described in this section behaves significantly better than other methods.

Because we make extensive use of velocity interpolation when integrating the surface vertices, we need to be able to access the velocity field $\mathbf{u}$ in many regions just outside of the boundary of the fluid simulation. To make sure that our velocity field $\mathbf{u}$ adequately samples space, we extrapolate $\mathbf{u}$ outward using a fast marching method [2].

Lastly, when we update the positions of the surface vertices, some vertices may lie within solid obstacles. One way to address this problem is to resolve the collision by projecting the vertices onto the surface of the obstacle. This works well in most cases, though it can cause problems when an entire thin sheet of liquid lies inside an obstacle. In such cases, simply projecting vertices onto the surface of the obstacle tends to create infinitely thin sheets of liquid. These degenerate cases can cause problems later (during volume or convex hull calculations, for example). In practice, we used free-slip boundary conditions at these solid obstacles and ignored any resulting collisions. While this approach worked surprisingly

well for us, a more principled approach to collision handling may work better.

2.2.2 Updating the Finite Difference Grid

As mentioned in the previous section, the fluid simulation uses the surface mesh to determine which cells will be active fluid cells, and which cells will be inactive "air" cells. This is essentially the same problem as *voxelizing* a surface mesh, or returning a set of grid cells that overlap the volume contained within a surface mesh. However, we can take this method a step further and actually compute a signed distance function in order to give ourselves some more information about the surface geometry. The scalar values of this signed distance function (negative values inside, and positive values outside) are collocated with the cell-centered pressure values in the fluid grid.

We employ a voxelization-style method [17] to compute the signed distance function. We first voxelize the triangle mesh onto a grid by computing intersections with a rays in the x, y, and z directions. For each ray, we keep track of its inside/outside status with a counter. At the start of the ray (which is guaranteed to be outside of the mesh), the counter has a value of zero. For each intersection with the triangle mesh, we increment the counter if it intersects a triangle whose normal is facing the opposite direction of the ray, and we decrement it if it intersects a triangle with a normal facing the same direction of the ray. This way, all regions outside of the mesh will have a counter value of zero, regions inside the mesh will have a value of one, regions where multiple surfaces overlap each other will have a value greater than one, and regions that are inside-out will be negative. We store this counter value at each grid point to assign an inside/outside status, and then we compute the distance from the point to the nearest surface triangle in order to complete the signed distance calculation. Because our algorithms only need an accurate signed distance in the thin band surrounding the surface, we save time by only performing this distance calculation for grid cells that overlap surface geometry.

At this point, we can use this signed distance function to determine which fluid cells are inside of the surface (treating all "inside" cells as active fluid cells in the next time step). However, some regions of the surface may have been too thin to be adequately sampled by the fluid grid, so these thin regions would not be surrounded by any active fluid cells with this method.

We use an explicit geometric technique to ensure that all thin features will be represented by negative values on the signed distance function grid. First, we find the set of all fluid cells that intersect or completely envelop any surface triangles — this can be done efficiently by looking at triangle vertex locations and seeing which grid cells they lie within. We then take the union of that set of cells and the original set of cells associated with negative values in the signed distance function. This final set of cells will adequately sample all thin features.

2.3 Free-surface boundary conditions

Like any Eulerian grid-based fluid simulation, we need to handle boundary conditions for the free surface. This can be done with several different techniques, and we will discuss two

```
Compute inside/outside classification of mesh
Assign new fluid cells
Calculate fluid velocity
Advect surface vertex positions
If mesh collides with solid boundary
  Update vertex positions
```

Figure 4: Pseudo-code for one timestep of Eulerian fluid simulation

specific methods here.

2.3.1 Constant cell weights

The easiest way to implement the boundary conditions at the liquid surface is to simply assume that every fluid cell represents a unit of mass $\rho\Delta x^3$, where Δx is the width of a fluid cell. Unfortunately, the embedded surface mesh will not necessarily occupy the exact same volume as the simulation elements in general, so the simulation will overestimate the total mass in the system. This is noticeable in thin regions of liquid, when a surface sheet splashes into a larger body of water. The artificially large mass injects additional momentum into the system and creates a larger-than-expected splash. Although this behavior may actually be desirable in a special effects environment and more pleasing to watch, it is nevertheless inaccurate.

Errors in this constant-cell-weight strategy are most visible when thin surface regions stretch out into even thinner surfaces. Here, the volume of these surface features is preserved, so the volume per unit length drops as the length increases. However, the mass per unit length stays the same with this strategy, so a large amount of mass and momentum is added into the system whenever a small region thins out.

Although the constant cell weight method is less accurate, it is quite simple to implement. We used this method in the simulations by Wojtan et al. [25, 26] and Thürey et al. [23].

2.3.2 Ghost Fluid Method

Alternatively, we can use a more accurate fluid discretization, called the *ghost fluid method* [8, 1]. This is a second-order approximation that essentially uses a linear extrapolation to assign pressure values outside of the free surface. Because the fluid pressure should drop to zero exactly at the surface, an extrapolation will give negative pressures outside of the fluid surface. Combining the ghost fluid method with a surface mesh has yielded high quality visual results in the simulations of [5].

2.4 Algorithm Overview

Figure 4 shows pseudo-code for a single time step of an Eulerian fluid simulation with a mesh-based embedded surface. The simulation first computes an inside/outside test for

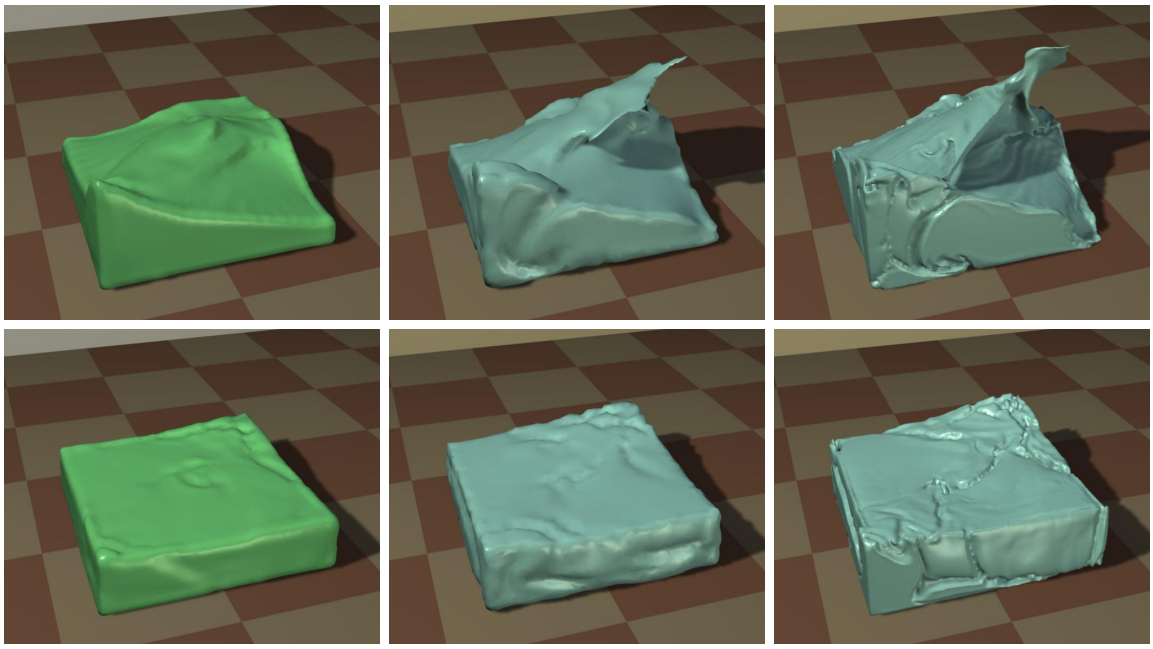

Figure 5: Different examples of surface tracking, from left to right: Level set, low-resolution triangle mesh, and high-resolution triangle mesh. The top row shows how mesh-based surface tracking can better capture thin surface structures, and higher-resolution meshes capture extremely detailed surface features. The bottom row shows how surface artifacts can accumulate when the surface is much higher resolution than the physics simulation (far right).

each fluid cell. Next, each simulation cell that is located inside of the mesh is considered a "fluid" cell, while the cells outside of the mesh are ignored. Next, we solve one time step of the incompressible Navier-Stokes equations to get a new fluid velocity for each fluid cell. Finally, we advect the surface vertices through the Eulerian fluid cell velocities using standard techniques (e.g. 4th order Runge-Kutta), and then adjust these vertex positions in the case of any collisions with solid obstacles.

2.5 Problems with differing surface and simulation resolutions

When a very high resolution surface is advected through a comparatively low resolution physics simulation, interesting visual artifacts can occur. In particular, the high resolution features captured by the surface (a high resolution mesh or high resolution level set) cannot be communicated to the physics simulation. As a result, the physics simulation does nothing to correct unphysical high resolution features, such as surface kinks and small voids. Some examples of this type of phenomena can be seen in the right column of Figure 5. One way to solve this problem is to adapt the Eulerian simulation grid to match up with the surface geometry. Later in this course, we will describe a very effective method for achieving this matching between the surface and grid resolutions using Voronoi regions for the Eulerian fluid elements [5].

2.6 Limitations

In this section, we introduced a straightforward method for coupling a grid-based Eulerian fluid simulation to a mesh-based Lagrangian surface. As such, it inherits the benefits of both the Eulerian and Lagrangian simulation techniques. By using a semi-Lagrangian advection scheme, we can guarantee that the fluid simulation will be unconditionally stable for large time steps. The Lagrangian nature of the surface mesh also avoids the accumulation of re-sampling errors that are common in Eulerian surface tracking schemes.

Although it is quite basic, the surface tracking method described in this section is already quite good. Unlike many Eulerian surface tracking schemes, this method has no smoothing errors caused by continually re-sampling the surface from a grid. The only numerical errors in this scheme are due to the fluid simulation or the time integration. Figure 3 shows how this simple mesh-based surface tracker compares to other common surface-tracking schemes in the Zalesak sphere test.

One major problem with this approach as presented so far is that it does nothing to address topology changes in the surface geometry. Lively fluid animations can have drastic changes in the connectivity of surface geometry, so it is important that we address this issue. We will discuss this problem and present some solutions in more detail later in the course. Another problem with this approach is that surface triangles can become very distorted, causing many computational problems. We will discuss some ways to maintain a high triangle mesh in the section of this course titled "Maintaining Surface Mesh Quality."

3 Maintaining surface mesh quality

As we update the position of surface vertices during a fluid simulation, the surface mesh will deform. If we do not change the connectivity of the triangle mesh, then triangles in the surface mesh can become severely distorted — some triangles may become much too large, while other triangles may have near-zero area. To address this problem, we perform various "mesh-maintenance" operations at the end of each advection step. This section of the course will elaborate on this idea and provide some tools to improve the quality of the surface mesh.

3.1 Why do we care about mesh quality?

The importance of mesh quality has been addressed thoroughly by others [21, 12], so we will only lightly touch on the concept here. Triangles with poor quality metrics can adversely affect a simulation in many ways:

- **Visual artifacts:** If we allow triangles to become arbitrarily large, then giant jagged spikes may pervade the visible surface in our fluid animations.
- **Memory and time limitations:** Triangles with small areas use space very inefficiently. Without any restrictions on triangle area, our simulations can get bogged down by millions of invisible triangles. This is bad for memory usage, and it is a waste of time to run computations on so many vertices.
- **Geometric computations:** We may decide to use different forms of geometric computation, such as collision detection, ray tracing, topology changes, or boundary integration. Triangles with poor quality can create serious robustness problems for these operations.
- **Numerical stability:** If we wish to use any numerical integration techniques on our surface mesh, such as surface smoothing or surface tension calculations, then we need to maintain a high quality triangle mesh.

3.2 Measuring triangle quality

A very simple way to detect troublesome triangles is by measuring the lengths of all edges in the mesh. If any edge is smaller than some minimum edge length, then it may create problems for our simulation. Similarly, if any edge is larger than some maximum length, then it may be too bloated and distorted to accurately sample the surface. Some triangles may have perfectly acceptable edge lengths but have very small areas (see Figure 6), so a simple edge-length measurement is not enough to find poorly-shaped triangles. In addition to measuring edge length, we can also measure the area and penalize very large or very small area measurements, or we can measure triangle angles and penalize extreme angles that are too large or too small.

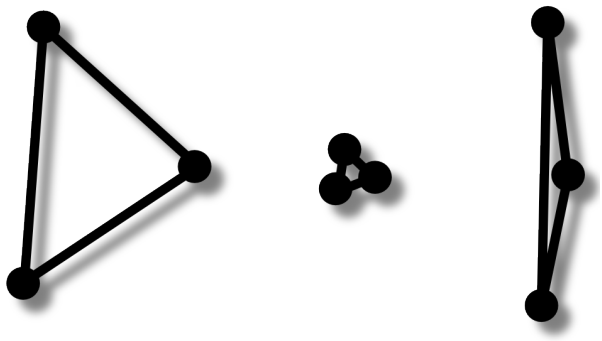

Figure 6: The leftmost triangle is of acceptable quality: it has large area, and its edge lengths are nearly equal. The middle triangle has a very small area, and its edge lengths are too short. The rightmost triangle has good-sized edge lengths, but its area is still too small.

The problem of finding a metric for triangle quality in a way that is meaningful and useful is very well studied. This section only briefly touches on some of the simplest measurements — just enough to detect the problematic triangles in our mesh so we can get rid of them. For a much more thorough discussion of triangle quality measures (and linear element quality in general), see [21].

3.3 Mesh quality operations

Once we have determined which triangles are unacceptable, we need to actually remove them from our triangle mesh. This section explains how to perform several different mesh operations in order to locally improve triangle quality. The operations covered in this section are the *edge split*, the *edge flip*, the *edge collapse*, and *null-space smoothing*. These operations are relatively independent from each other and can be performed in any order, though certain orders are more sensible than others. For a point of reference, the mesh-based surface tracking software *El Topo* [6] uses these operations in the same order that we present them in this chapter. The simulations in [27, 25, 23, 26] use only the *edge split*, followed by the *edge collapse* operations.

Each of these operations may be performed in a way that guarantees the resulting triangle mesh to be collision-free, and we include those details here these details here for completeness. These collision-free constraints can be safely disregarded if you don't care whether your surface has self-intersections, or if those intersections will be fixed by some topology-changing operation later. These mesh maintenance operations may also be extended to handle non-manifold triangle meshes, though we omit those details here.

3.3.1 Edge split

If an edge is longer than a user-defined maximum edge length, we subdivide it by introducing a new vertex (see Figure 7). The new vertex can be placed at the edge midpoint, which will not introduce any new intersections. However, we may wish to offset the new vertex from the

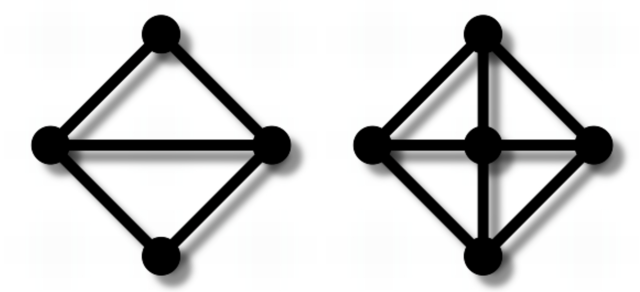

Figure 7: Edge split operation

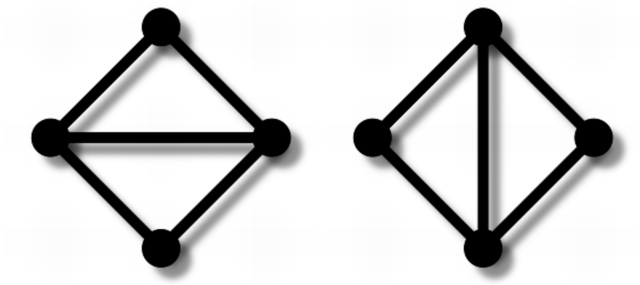

Figure 8: Edge flip operation

current surface using a subdivision scheme to maintain curvature. We begin by introducing the new vertex at the edge midpoint, and we then compute the predicted location of the new vertex via the subdivision scheme. These two points define a *pseudomotion*. We check the new vertex and its incident triangles and edges as it moves from the edge midpoint to its predicted point to ensure that it does not collide with any existing mesh elements (which do not move during this pseudomotion). If a collision does occur, we can revert to using the edge midpoint, which is guaranteed to not introduce any new intersections.

3.3.2 Edge flip

We employ an edge flip operation as a way of maintaining good triangle aspect ratios. For each edge incident on two triangles, we check whether the distance between the two points *not* on the edge is less than the length of the edge. If so, we remove the edge and create a new edge between these two points (see Figure 8). A simple way of ensuring that this operation does not introduce any intersections is to check that no existing edge intersects the two new triangles and that no point lies inside the tetrahedron formed by the two new and two old triangles. We also reject the edge flip if it introduces a change in volume greater than a user-defined maximum volume change (we usually set this maximum volume change to be $0.1\xi^3$, where ξ is the average desired edge length; for simulations involving extremely thin surfaces, this may need to be further reduced).

Flipping a single edge may introduce new triangles with poor aspect ratios, so we itera-

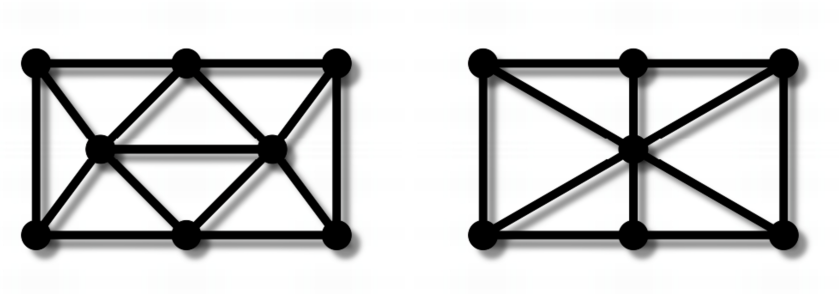

Figure 9: Edge collapse operation

tively sweep over all edges in the mesh until no flip is performed or until we reach a maximum number of sweeps (in [6], we set this maximum to five). We also require that the new edge length decrease by a minimum amount to prevent the same edge from flipping back and forth on subsequent sweeps.

3.3.3 Edge collapse

If an edge is shorter than a user-defined minimum edge length, we attempt to collapse it by replacing it with a single vertex as shown in Figure 9. As with edge splitting, we treat only manifold edges, skipping edges incident on more than two triangles. We use a subdivision scheme to choose the location of the new single vertex in the general case. However, we also use an eigen-decomposition of the quadric metric tensor to detect vertices that lie on ridges or creases [13]. If one edge end point lies on a ridge and the other lies on a smooth patch of the surface, we set the new vertex position to be the position of the existing vertex on the ridge. In other words, we wish to prevent vertices moving off of the ridge, which tends to introduce bumps or jagged edges.

To ensure collision safety, we can use the same pseudomotion collision detection described during our *edge split* explanation earlier, this time with two vertices in motion: the end points of the edge. These end vertices will have the same predicted location: the location chosen by the subdivision algorithm. If a collision is detected during this pseudomotion, we can try again, this time moving the vertices towards the edge midpoint. Unlike edge splitting, however, we have no safe fallback vertex location. If we insist on having a collision-free mesh and cannot find a collision-free trajectory, then the edge collapse must be abandoned.

We use simple minimum and maximum edge lengths for determining when to split and collapse edges. In practice, we compute the average edge length when the mesh is initialized and set the minimum and maximum edge length parameters to be some fractions of the initial average length. This has the effect of keeping the edge lengths within some range of the initial average using split and collapse operations. In our examples, we allow edge lengths to vary between 0.5 and 1.5 of the initial average edge length; however, these parameters did not require tuning and our system remains stable for other values. More sophisticated criteria for triggering a split or collapse exist, such as detecting triangles whose areas are too

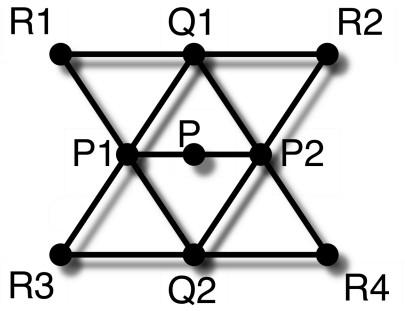

Figure 10: Butterfly subdivision

small or too large, or aspect ratios that are too far from unity (c.f. [13]).

When choosing locations for new vertices during an edge collapse or split operation, there are a number of schemes that can be used. We use traditional butterfly subdivision [7] due to the simplicity of its implementation and because it is free of parameters. Quadric error minimization schemes [11] are promising, but in our experience the simplicity and quality of butterfly subdivision made it more attractive.

Butterfly subdivision determines the location of a new edge midpoint P_{new} as

$$P_{\text{new}} = \frac{1}{16}(8(P_1 + P_2) + 2(Q_1 + Q_2) - (R_1 + R_2 + R_3 + R_4)),$$

where P_1 and P_2 are end points of the edge, Q_1 and Q_2 are the vertices on the two triangles incident on the edge which are not the edge end points, and $R_1 \ldots R_4$ are the vertices on the four triangles adjacent to the triangles incident on the edge (see Figure 10). We have also found that the adaptive butterfly subdivision scheme of [28] worked very well, because it does not assume that all vertices have a valence of six. This adaptive subdivision scheme helps ensure high degrees of smoothness even for abnormal triangle configurations.

One potential danger to keep in mind is that non-manifold geometry can be created by collapsing an edge with a valence-3 vertex (see Figure 11. The edge collapse will result in two non-manifold triangles (each sharing the same three vertices). If your system is indifferent to non-manifold geometry, then you can ignore this warning. If, however, you must ensure that your surface geometry is manifold (e.g. for simpler implementation of shape operators), then we can do one of two things: we can either forbid this edge collapse, or we can allow the collapse and delete any resulting non-manifold geometry. The first option is far simpler to implement.

3.3.4 Null-space smoothing

A powerful mesh improvement technique was recently introduced by Jiao [13]. Applying a Laplacian filter to the vertex locations would move each vertex to the average of its neighbors locations. This usually has the desirable effect of equalizing edge lengths. However, it will also shrink the volume enclosed by the surface and smooth away sharp features. We instead move the vertices only in the null-space of their associated quadric metric tensors. If the

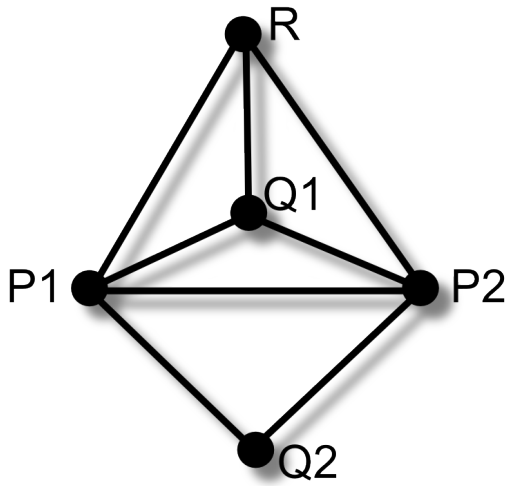

Figure 11: Example of an edge collapse that will result in non-manifold geometry. If P_1 and P_2 collapse into a single point P, then we are left with two triangles sharing the exact same 3 vertices (P, R, and Q_1).

vertex is on a flat or smoothly curved patch of surface, the null space will correspond to the plane tangential to the surface at the vertex. If the vertex is on a ridge, the null space will be the infinite line defined by the ridge, and the smoothing operation preserves the ridge feature. If the vertex is at a corner, the null space will be empty and the vertex will not move, preserving the corner. To ensure no mesh intersection, we treat the global smoothing operation as a pseudotrajectory on all vertices and apply collision resolution as if the surface was moving under the influence of an external velocity field.

References

[1] Christopher Batty, Stefan Xenos, and Ben Houston. Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. In *Proceedings of Eurographics*, 2010.

[2] Robert Bridson. *Fluid Simulation for Computer Graphics*. A K Peters, 2008.

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

[4] Tyson Brochu. Fluid animation with explicit surface meshes and boundary-only dynamics. Master's thesis, University of British Columbia, 2006.

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

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

[7] N. Dyn, D. Levine, and J.A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM transactions on Graphics (TOG)*, 9(2):160–169, 1990.

[8] D. Enright, D. Nguyen, F. Gibou, and R. Fedkiw. Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *Proc. 4th ASME-JSME Joint Fluids Eng. Conf., number FEDSM2003–45144. ASME.* Citeseer, 2003.

[9] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 2001. ACM.

[10] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01*, pages 23–30, New York, NY, USA, 2001. ACM.

[11] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[12] X. Jiao, A. Colombi, X. Ni, and J. Hart. Anisotropic mesh adaptation for evolving triangulated surfaces. In *Proceedings of the 15th international meshing roundtable*, pages 173–190. Springer, 2006.

[13] Xiangmin Jiao. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys.*, 220(2):612–625, 2007.

[14] ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jarek Rossignac. Advections with significantly reduced dissipation and diffusion. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):135–144, 2007.

[15] Doyub Kim, Oh-young Song, and Hyeong-Seok Ko. Stretching and wiggling liquids. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, pages 1–7, New York, NY, USA, 2009. ACM.

[16] Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. Low viscosity flow simulations for animation. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–18, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[17] 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.

[18] Stanley Osher and Ronald Fedkiw. *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2003.

[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1994.

[20] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *J. Sci. Comput.*, 35(2-3):350–371, 2008.

[21] Jonathan R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In 11^{th} *Int. Meshing Roundtable*, pages 115–126, 2002.

[22] Jos Stam. Stable fluids. In *SIGGRAPH '99*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[23] Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. *ACM Trans. Graph.*, 29(4):1–10, 2010.

[24] Chris Wojtan. *Animating physical phenomena with embedded surface meshes*. PhD thesis, Georgia Institute of Technology, 2010.

[25] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3):1–10, 2009.

[26] 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.

[27] Chris Wojtan and Greg Turk. Fast viscoelastic behavior with thin features. *ACM Trans. Graph.*, 27(3):47, 2008.

[28] D. Zorin, P. Schröder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *ACM SIGGRAPH 1996 papers*, page 192. ACM, 1996.