# Deforming Meshes that Split and Merge

Chris Wojtan
Georgia Institute of Technology

Nils Thürey
ETH Zürich

Markus Gross
ETH Zürich

Greg Turk
Georgia Institute of Technology
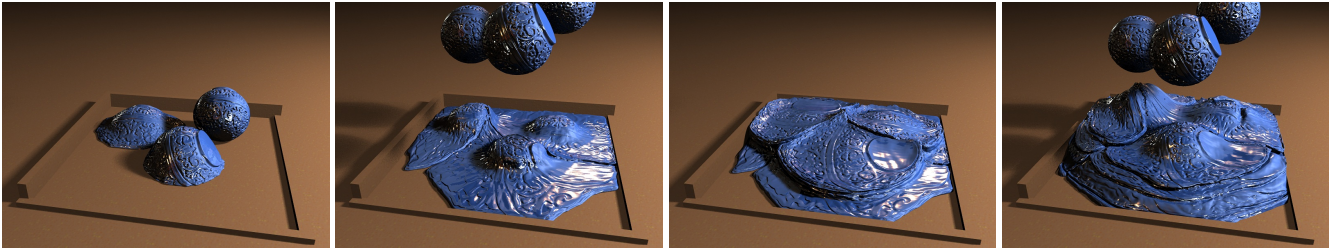
**Figure 1:** *Dropping viscoelastic balls in an Eulerian fluid simulation. Invisible geometry is quickly deleted, while the visible surfaces retain their details even after translating through the air and splashing on the ground.*

## Abstract

We present a method for accurately tracking the moving surface of deformable materials in a manner that gracefully handles topological changes. We employ a Lagrangian surface tracking method, and we use a triangle mesh for our surface representation so that fine features can be retained. We make topological changes to the mesh by first identifying merging or splitting events at a particular grid resolution, and then locally creating new pieces of the mesh in the affected cells using a standard isosurface creation method. We stitch the new, topologically simplified portion of the mesh to the rest of the mesh at the cell boundaries. Our method detects and treats topological events with an emphasis on the preservation of detailed features, while simultaneously simplifying those portions of the material that are not visible. Our surface tracker is not tied to a particular method for simulating deformable materials. In particular, we show results from two significantly different simulators: a Lagrangian FEM simulator with tetrahedral elements, and an Eulerian grid-based fluid simulator. Although our surface tracking method is generic, it is particularly well-suited for simulations that exhibit fine surface details and numerous topological events. Highlights of our results include merging of viscoplastic materials with complex geometry, a taffy-pulling animation with many fold and merge events, and stretching and slicing of stiff plastic material.

**Keywords:** Physically Based Animation, Deformable Meshes, Fluid Simulation, Topological Control

## 1 Introduction

The physical world is rich in materials that deform. In many instances, these materials easily undergo topological changes, allowing separation or joining of components. Such materials include water, toothpaste, bread dough, peanut butter, taffy, tar, and clay. In order to produce plausible animations of these deforming materials, we must develop simulation techniques that retain visual details, but at the same time allow topological changes to the surface. This is the focus of our work.

We introduce a method of tracking and updating the surface of a deforming object in a manner that gracefully allows for topology changes. Our approach uses a mesh to represent the surface of objects. The key benefit of using a mesh is that it allows the surface to be moved in a Lagrangian manner, using a high-order ODE integrator to move the vertices of the surface. This results in the retention of fine surface details. The drawback of using a mesh surface is that changing the topology of a mesh is notoriously tricky, both in terms of robustness and program complexity. Our approach avoids these difficulties by replacing those portions of the mesh where a topology change should occur with a simplified surface that is generated using a standard isosurface creation method. This new portion of the mesh is then stitched together with the untouched parts of the surface.

Our approach to topology change lets us decide which potential topological modifications are permitted and which ones should be prevented. This allows us to customize the behavior of the simulator in order to simulate materials with particular attributes. For example, we can allow surface separation events, but forbid surface merging. This might be appropriate for simulating bread dough that has been rolled in flour so that it no longer sticks to itself. One particular rule for marking where a merge event should be triggered is what we refer to as the *deep cell test*. This test allows surface merging when it is deeply embedded in the folds of a material, but it leaves alone potential merge events that are near the surface and that might produce visually distracting results.

The key attributes of our approach to surface tracking are as follows:

- Our algorithm performs topological splits and merges, which decreases the memory and computation requirements of dealing with complicated surfaces and permits interesting behaviors like the merging of water droplets.

- Our simulator retains thin features and fine surface details.

- The method is robust, even during complex topology changes.

- The surface tracker can be easily tuned to allow or forbid particular kinds of topology changes to better suit a material.

- Our surface tracker does not depend upon any particular simulation technique. In particular, we demonstrate results from

both an Eulerian fluid solver and an FEM simulator.

- We decouple the physics, topological, and surface detail resolutions in our simulator. By modifying the level of detail in each of these behaviors, we can customize our simulations to exhibit complicated phenomena with a limited amount of computation.

## 2   Related Work

Tracking surfaces with changing topologies is an important problem for a variety of fields - from numerical simulations to image processing problems [Kass et al. 1988]. Currently, the most commonly used method for tracking such surfaces are level set methods. They are especially popular in the field of physically based animation, as they can naturally handle topological changes and do not require a surface parametrization. Level set methods track the surface implicitly as the isosurface of a higher dimensional function, typically a signed distance function stored in a Eulerian grid. They were first presented by Osher and Sethian [1988], and have since been extended and refined in many ways. Adalsteinsson and Sethian [1995] presented a method to restrict the necessary computations to a narrow band, while Sethian [1996] developed the related class of fast marching methods. To accurately advect the implicit function, higher-order *WENO* schemes are typically used [Liu et al. 1994]. However, these more accurate variants of the level set approach require small time steps to ensure stability. Particle level set methods, developed by Enright et al. [2002b], combine Lagrangian advection with Eulerian level set schemes to achieve a more accurate surface evolution. In combination with semi-Lagrangian advection, such as the method by Selle et al. [2008], this allows arbitrarily large time steps. Our method likewise makes use of accurate Lagrangian advection, but in contrast to a particle level set, we store explicit mesh connectivity information. Closely related to level sets are the so-called *volume-of-fluid* methods, that explicitly track a surface by computing mass fluxes [Hirt and Nichols 1981; Sussman 2003], but they are not often used in computer graphics applications due to flotsam and jetsam artifacts.

Strain [2001] formulated semi-Lagrangian contouring (SLC) for surface tracking, and Bargteil et al. [2006] extended the idea for 3D fluid animation. Like our method, SLC computes signed distances exactly from a triangle mesh. However, the accuracy of SLC is limited by the resolution of the implicit surface representation, while our method retains high resolution details by only locally re-sampling the surface where topological changes occur. If we recomputed the entire surface every time step, our method would behave similarly to SLC.

Point-based methods use unstructured Lagrangian marker particles to track the interface. These methods have been used in the field of computer graphics to simulate melting objects [Terzopoulos et al. 1989], liquids [Müller et al. 2003; Zhu and Bridson 2005; Adams et al. 2007] and deformable objects with topology changes [Pauly et al. 2005]. Point-based surface tracking has been demonstrated in [Keiser et al. 2005], but, in order to perform re-sampling or allow for topology changes, these methods still require a continuous surface approximation which introduces smoothing.

Purely mesh based surface trackers [Reynolds 1992; Glimm et al. 1998; Jiao 2007], on the other hand make it very difficult to account for topological changes. [McInerney and Terzopoulos 2000] use an underlying grid to compute topology changes for a mesh-based tracking, but are restricted to a strictly inward or outward motion. Other approaches have been suggested that focus on detecting and re-meshing self-intersections using a set of heuristic rules [Bredno et al. 2003; Lachaud and Taton 2005; Brochu 2006]. Pons

and Boissonat [2007] proposed a mesh-based approach for topology changing interfaces that requires a tetrahedralization of the interface points. This technique produces efficient and clean topological changes for smooth surfaces, though it is asymptotically slower than our method and its tetrahedralization makes topological control more difficult.

We overcome the difficulties of re-meshing by locally replacing the invalid parts of the mesh with one generated by an isosurface creation method, such as marching cubes [Lorensen and Cline 1987]. The work most similar in spirit to ours is that of Jian Du et. al [2006]; their approach also locally replaces invalid geometry with an isosurface. However, Du et al. uses a conservative re-sampling procedure by merging overlapping isosurface regions into convex boxes, while our method allows arbitrary concavities. Furthermore, their intent was to prevent geometry self-intersections, while our aim is to control topology changes such as merging and tearing sheets of material. A related technique by Bischoff and Kobbelt [2005] converts a CAD model to a polygon mesh by sewing together closed polygonal patches and effectively removes mesh orientation information during the conversion process. We only deal with oriented manifold triangle meshes, which simplifies several calculations and allows for clean self-intersection tests.

The control of topological changes is important in image segmentation, where the topology is known beforehand and should not change. This can be done using concepts from *digital topology* [Rosenfeld 1979]. They have more recently been used to control topology changes of level sets [Bischoff and Kobbelt 2003]. Instead, our algorithm is based on the *complex cell test* [Varadhan et al. 2004], but we modify it to retain fine details on the tracked surface while merging undesired folded sheets.

To demonstrate the versatility of our method, we will show simulations of deformable objects based on a Lagrangian *finite element method* (FEM), and liquid simulations performed with an Eulerian fluid simulator. Deformable models were first used by Terzopolous et al. [1988], while animations based on finite elements were first presented by O'Brien and Hodgins [1999]. These methods were since then extended by the use of shape matching [Müller et al. 2005], or by enforcing incompressibility [Irving et al. 2007].

The underlying simulation we will use in the following is based on the work of Bargteil et al. [2007]. Similar to Sifakis et al.[2007] and Wojtan and Turk [2008], we embed a high resolution mesh into a body-centered cubic (BCC) lattice. Although these methods are able to simulate deformable objects with thin features, they cannot handle topological merges of the embedded mesh, which makes simulations such as the taffy pulling shown in Figure 9 very difficult and time consuming.

Our second set of examples was created with a Eulerian fluid simulator. Fluid simulations in computer graphics were first demonstrated by Kass and Miller in [1990]. The approach we are using for our simulations was pioneered by Stam [1999], and Fedkiw et al. [Foster and Fedkiw 2001; Enright et al. 2002b]. The combination of operator splitting, semi-Lagrangian advection, and particle level set based surface tracking was afterwards extended to be coupled to thin shells [Guendelman et al. 2005], treat interface discontinuities [Hong and Kim 2005], handle multiple interacting liquids [Losasso et al. 2006], and animate viscoelastic fluids [Goktekin et al. 2004]. A more accurate coupling with solid objects was presented by Batty et al. [2007]. Approaches to reduce the high amount of computations are model reduction [Treuille et al. 2006], or the use of procedural models for detail [Kim et al. 2008]. An excellent overview of the algorithms that we use using can be found in the book by Bridson [2008]. Our approach to handling topological changes should also work in combination with other types of
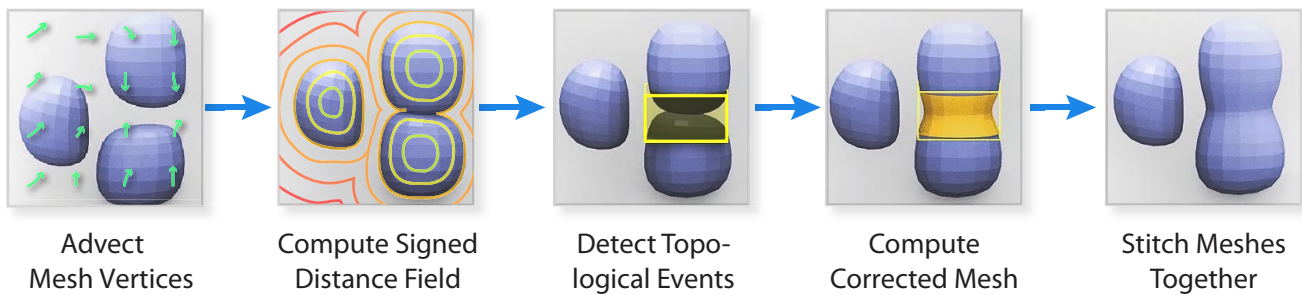
**Figure 2:** *Overview of our topology modification pipeline.*

solvers, such as Smoothed Paricle Hydrodynamics [Müller et al. 2003] or lattice Boltzmann based solvers [Thürey and Rüde 2004].

## 3  Overview of Approach

Our approach to surface tracking does not depend strongly on the characteristics of the physics simulation that is being used. We assume that we have a simulator that begins with the initial conditions of the simulation (the geometry and associated material parameters) and that integrates the governing equations of motion to produce a new simulation state at the next time-step. The simulator may explicitly update the location of the material's surface, or it may implicitly specify the position of the surface with a velocity field each time step. For example, the simulator may use an Eulerian grid to calculate the motion of a fluid using the Navier-Stokes equations. The velocity field that is generated based on the fluid motion is given to our surface tracker, and the tracker moves the fluid surface–possibly modifying the topology in the case of events such as splitting of a stream into drops or merging sheets of fluid together.

The main steps of our surface tracking algorithm are shown in Figure 2. The tracker is given a closed manifold mesh $M$ that represents the surface of the given material. The surface tracker first moves the vertices of this mesh according to the update rule given by the simulator: typically via interpolation of simulation nodes or numerical integration through a velocity field, as will be described in Section 4. Next, the space around the mesh is finely sampled on a regular grid, and the result of this sampling procedure is a signed distance field $D$. The sign of a sample in the distance field is determined by whether the sample is inside or outside of the mesh $M$, and the magnitude is the distance to the nearest part of the mesh. This distance field is then traversed, and the eight distance samples at the corners of each cell are examined to determine whether a topological event may be taking place at the given cell. There are several possible decision processes that may be used, and one that we favor is the *deep cell test*, which will be explained in Section 5. This results in a list of cells that are marked for modification. We clip the surface mesh $M$ to the boundary of this collection of marked cells, retaining only the portions of the mesh that are outside of the marked cells. This will be described in Section 6. We then form new mesh components inside the marked cells using marching cubes, and the clipped exterior mesh is joined to the interior mesh components. The result is a new mesh that has been modified in regions of topological change. We give details about each of these steps in the sections that follow.

## 4  Lagrangian Mesh Update

The first step of our method is to update the location our mesh $M$ with the information provided by our physics simulator. The mesh consists solely of vertices with three-dimensional locations and triangles with three pointers to vertices. In the absence of topology changes, we can move the entire surface of the mesh by simply moving the locations of the vertices and keeping the original triangle connectivity. We proceed to move the vertices in this way during our Lagrangian mesh update step, and we process any topological events afterward.

Our fluid simulations use a grid-based Eulerian Navier-Stokes solver, which produces a velocity field $V$ at each time step. We choose to accurately advect $M$ through $V$ using the explicit fourth-order Runge-Kutta method. Our finite element simulations, on the other hand, describe the surface vertex locations as a function of simulation node positions. In this case, the tetrahedral computational elements are themselves repositioned with a Lagrangian ODE solver, so we do not need to separately advect our surface mesh. Instead, the location of each vertex is determined via barycentric interpolation within each element.

Both of our physics simulation environments employ Lagrangian techniques for repositioning the vertices of our surface mesh. This Lagrangian update has an important strength, in that it inherently preserves surface details.

An alternative to our approach is the level set method, which uses a series of fixed grid points to accurately advect the surface through space [Sethian 1999]. The grid points do not move along with the surface in this case; instead, the surface is implicitly defined by scalar values at the grid points, and an explicit surface is obtained via interpolation. This Eulerian approach re-samples the surface mesh at every simulation step, which quickly leads to numerical smoothing of surface features. Furthermore, purely Eulerian methods make it impossible to represent a surface with details smaller than the grid resolution.

Some methods choose to combine Eulerian and Lagrangian techniques in order to better preserve surface details. This style of surface tracking can begin with a level-set technique and re-sample the surface at each timestep [Bargteil et al. 2006] or augment the level-set with Lagrangian particles [Enright et al. 2002b; Hieber and Koumoutsakos 2005; Mihalef et al. 2007]. Such hybrid Lagrangian techniques significantly increase the amount of detail retained during each simulation step through the use of extra information afforded by the Lagrangian particles. However, because the surface is routinely re-sampled, it is difficult to retain sub-grid resolution surface details using these hybrid methods.

## 5  Detection of Topological Events

After we update our surface mesh, we decide whether we should split any surfaces apart or merge them together. We first calculate a signed distance field $D$ from our surface mesh, and then we examine the structure of this field to help decide where any topological
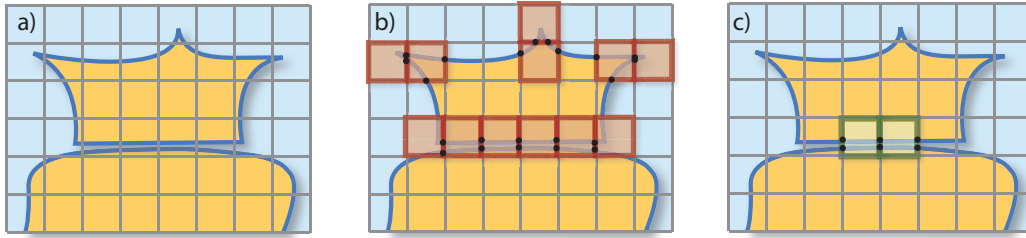
**Figure 3:** *A two-dimensional illustration of our deep cell test. Figure (a) shows an input surface mesh M (dark blue line) with a visualization of the corresponding signed distance field D, where orange points are inside of the surface, and light blue points are outside. Next is a figure showing all complex cells (b). The rightmost figure shows all deep cells (c). Note that the deep cells only label geometry necessary for a topological change, while the complex cells aggressively label important surface details.*

events should take place.

## 5.1 Signed Distance Field Calculation

We choose to place our signed distance function on a regular grid that encloses our surface mesh $M$. The first thing we note is that the only places where topological changes can occur are at grid cells that intersect $M$ (the surface cells). We then calculate the distance to the triangle mesh at each grid point that touches a surface cell by calculating the exact distance to each nearby triangle and taking the minimum. We then use a voxelization method [Nooruddin and Turk 2000] to determine which grid points lie inside of $M$, and which lie outside. We assign a positive signed distance to the grid points outside of $M$ and a negative sign to the points inside of $M$. Because we only calculate the signed distance at grid cells touching the surface, and because we only sample the nearest triangles for each distance query, this calculation of $D$ is efficient.

## 5.2 Topological Event Detection Mechanisms

At this point, we have two surface representations: an explicit surface mesh $M$, and a signed distance function $D$ that implicitly defines a surface at a specified grid resolution. We can contrast these two surface representations to give us an idea of where the surface is topologically complex.

One way to do this comparison is using *digital topology* [Rosenfeld 1979]. Although digital topology is a powerful tool for detecting topological events, it is inappropriate for our problem, because it requires a strict CFL condition for the movement of the surface. In addition, digital topology will only detect actual topology changes, leaving alone homeomorphic transformations. We would like to simplify our surfaces in invisible regions, such as large folds, even though this is not technically a topological change.

Another method we could use for detecting topology changes involves *complex cell tests*. Complex cell methods essentially decide whether our surface $M$ is too complicated to represent with a piecewise linear isosurface extracted from $D$. Varadhan *et al.* [2004] use a complex cell test in addition to a star-shaped test to decide whether their isosurfaces are topologically equivalent to an input mesh. Our method for detecting topological events uses a similar test, but we tailor ours to minimize re-sampling of the visible surface.

We can think of $D$ as a network of cubic cells with edges connecting each grid point to its 6 closest neighbors. In brief, a cell in $D$ is complex if the marching cubes algorithm [Lorensen and Cline 1987] will produce a significantly different surface from $M$ in that cell. More specifically,

- A *complex edge* is an edge in $D$ that intersects $M$ more than once.
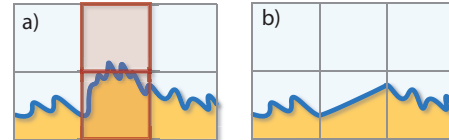


**Figure 4:** *This two-dimensional example shows how the complex cell test will aggressively re-sample detailed surface features, even in the absence of topological changes. In (a), we show cells that the complex cell test will mark due to multiple edge intersections, and we show the re-sampled surface in (b).*

- A *complex face* is a square face in $D$ that intersects $M$ in the shape of a closed loop or touches a complex edge.

- A *complex cell* is a cubic cell in $D$ that has any complex edges or complex faces, or any cell that has the same sign of $D$ at all of its corners while also having explicit geometry from $M$ embedded inside of it. The act of testing a cell for complexity is called the *complex cell test*.

This complex cell test can be used to mark a region in space where any topological changes occur in our surface. However, a straightforward application of this test will also mark detailed surfaces as topologically complex (see Figure 4). If we wish to preserve surface details like sharp corners, we must significantly modify this test.

## 5.3 Deep Cell and Self-Intersection Tests

Because we are primarily interested in the fidelity of the visual surface, we would prefer that most surface re-sampling occurs only in invisible regions or in the presence of major topological changes. To avoid the excessive re-sampling of highly detailed surfaces, we do not wish to mark all complex cells as topological events. Instead, we start by contrasting the interface of our explicit surface $M$ with the interface of our implicit surface $D$. In the interest of surface detail preservation, we ignore any complex cells that are sufficiently close to both the interfaces. We only mark a complex cell that is at least one cell away from the isosurface interface, as illustrated in Figure 3. This *deep cell test* is necessary because subtle bumps in surface geometry can still trigger any complex cell test, but only cells with geometry fluctuations larger than a cell length will be marked by our deep cell test.

The algorithm up to this point allows a great deal of control over which cells should be marked and re-sampled. For example, a simple method for preserving thin features is to only allow merging, but avoid splitting. This is done by not marking deep cells with corners that lie completely outside of the mesh. There are only two types of deep cells (those that are completely inside the implicit surface, and those that are completely outside), and each type corresponds to a merging or splitting event. By never marking deep cells that are
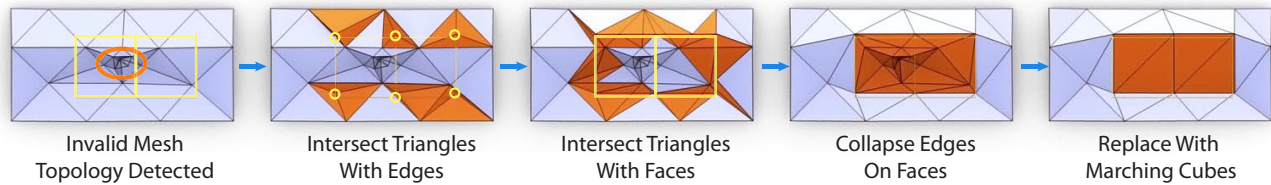
**Figure 5:** *The steps involved in locally replacing the mesh with a topologically simplified piece that is generated using Marching Cubes.*
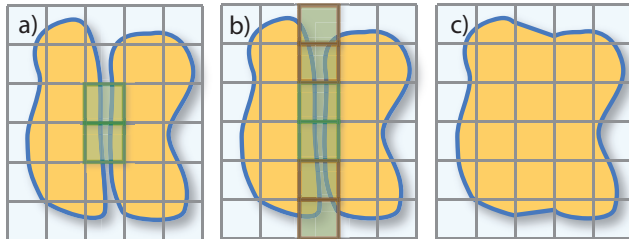


**Figure 6:** *A two-dimensional illustration of the cell-marching step. In (a), all deep cells are marked. We march outward along complex edges and faces until the marked region is topologically simple (b). The right figure (c) shows a topological change after re-sampling the marked cells.*

outside of the mesh, we can avoid splitting events and preserve thin features like sheets and spikes.

In addition to topological changes triggered by surface proximity, we also choose to mark cells that indicate significantly large self-intersections in $M$. This test is performed while marching rays through the mesh in the voxelization phase of our signed distance calculation. Whenever a ray intersects a triangle, we determine whether the ray is entering or leaving the surface by calculating the dot product of the ray direction with the triangle normal: a negative dot product indicates that the ray is entering the surface, while a positive dot product indicates an exit ray. In our implementation, we initialize an integer variable to zero at the start of the ray march, and then we increment the value if the ray enters the surface and decrement the value if the ray leaves. As the ray passes through each grid node, every node that does not have an integer value of zero or one necessarily is in a region of multiple overlapping or inside-out surfaces. We mark any cells touching these grid nodes as topologically complex. To ensure that this check is robust at triangle edges and vertices, we can use any techniques from the large literature for calculating robust ray-triangle intersections. We obtained robust behavior by simply jittering any vertices that lie too close to a ray.

Once we have marked every cell that we wish to re-sample, we want to replace $M$ in these cells with a topologically-simple isosurface extraction. However, we need to ensure that every cell on the boundary of our marked cell region provides a topologically simple interface with marching cubes. In other words, the boundary must contain no complex edges and no complex faces. We execute a flood fill algorithm, starting with the initially marked cells and marching outward across complex faces, until the entire region of marked cells is bound by topologically simple cube faces (see Figure 6). Once this region has a clean interface with marching cubes, we can perform surgery on the mesh.

## 6 Altering the Mesh Topology

At this stage in the algorithm, we have a mesh $M$, a distance field $D$, and a list of marked grid cells. Each of these marked cells describes

a region of space in which we will locally remove the topologically complex explicit surface $M$ and replace it with the topologically simple extracted isosurface of $D$. The surface inside of these cells is computed with a marching cubes algorithm and connected to the original mesh at the cell boundaries. To maintain a manifold surface mesh, it is important to ensure that the interface between the isosurface and the explicit surface matches up perfectly. We spend the rest of this section explaining how to compute this interface efficiently and robustly.

### 6.1 Basic Interface Computation Algorithm

Here, we describe the basic method for matching the interface between an explicit triangle mesh and an extracted isosurface. We also explain this algorithm graphically in Figure 5.

1. After marking topologically complex cells, we find each triangle that intersects a cell edge on the boundary of the marked region. We then calculate the intersection point between the triangle and edge, and we subdivide the triangle into three new triangles that share a vertex at the intersection point. We call this newly created vertex a *type 1* vertex.

2. Next, we find all triangle edges that intersect a cell face on the boundary of the marked region. We split each triangle edge at the point where it intersects the face, subdividing the two original triangles into four and inserting a new vertex on the face. We call this face vertex a *type 2* vertex.

3. At this point, the triangle edges between type 1 and type 2 vertices describe a single piecewise linear curve connecting type 1 vertices along each boundary cell face. We want to simplify this curve until it is a straight line between type 1 vertices. We repeatedly collapse the edges between type 1 and type 2 vertices until all type 2 vertices are removed from the mesh.

4. After all subdivisions have been performed, no triangles will cross the faces of any marked cells. That is, each triangle will lie completely inside or completely outside of the marked region. We delete all triangles that lie completely inside of the marked region.

5. After all edge collapses have been performed, at most one vertex lies on each boundary edge, and the mesh intersects the marked cell faces along a single line segment in each boundary cell face – the boundary faces are topologically simple and line up perfectly with a piecewise linear isosurface extraction of the signed distance field $D$. We use marching cubes to generate a triangle mesh in the marked region, and we connect the meshes together at the type 1 vertices.

Our method of modifying the topology of a mesh is similar to that of Du et al. [2006], but with a couple of significant differences. First, Du et al. only perform re-meshing when they detect a mesh self-intersection, whereas we also use this surface re-sampling for the more general purpose of permitting meshes to merge and split. Second, their approach forces their groups of marked cells to be in rectangular blocks, which means they often have to mark a much larger
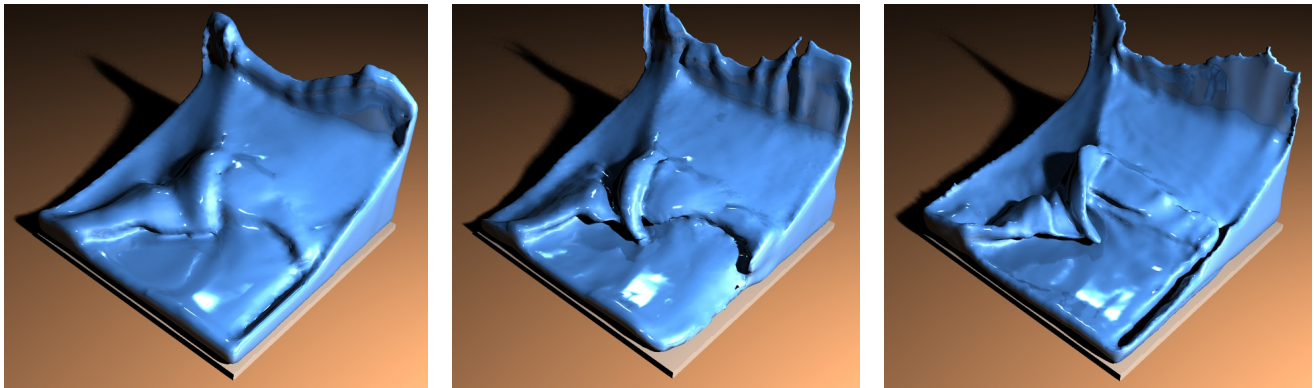
**Figure 7:** *Comparison between different surface tracking methods. Left: Level set. Middle: Particle level set. Right: Our mesh-based tracker.*
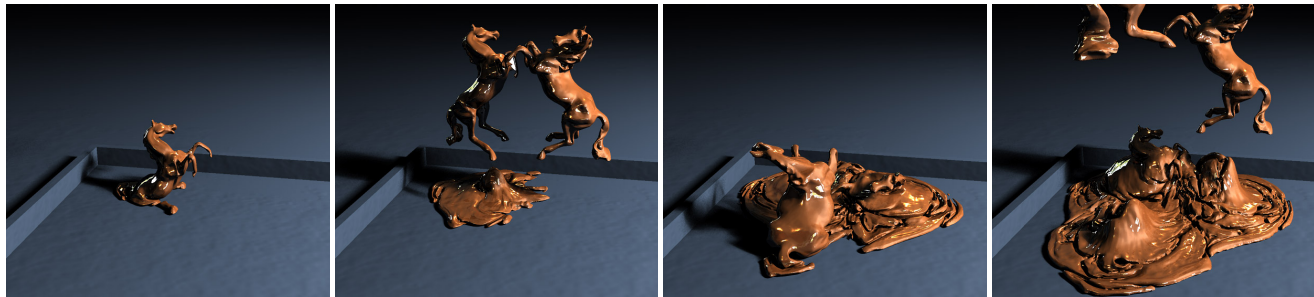


**Figure 8:** *These images from an animation show viscoelastic horses being dropped onto one another. Many topological merges occur, yet details of the surfaces are kept.*

portion of the mesh for modification. We allow our marked cells to be more sparsely distributed, such as a marked set of cells whose union forms concave regions like the shape of the letter *L*. Allowing these more general marked cell configurations necessitates extra checks during stitching, and we describe this next in Section 6.2.

### 6.2 Robustness

For robust simulation and intersection tests, we must ensure that our triangles are well-shaped. We adaptively subdivide surface triangle edges when they become too long, and we collapse edges when they become too short or when their triangles have bad aspect ratios. We also perform edge flips after the creation of type 1 vertices if any newly created triangle has particularly small angles. We avoid any edge-flip and edge-collapse operations that will create nonmanifold geometry. The edge-collapse operations may cause some re-sampling, but only in very small features. The triangle subdivision does not remove any surface details.

We also must take care to maintain simple topology along boundary faces before sewing the meshes together. Specifically, performing an edge decimation in step 3 of the above algorithm can deliver collateral damage to the surrounding cells by creating an additional curve between type 1 vertices on the face of a different boundary cell. If these edges are collapsed in the wrong order, we can easily force ourselves into creating nonmanifold geometry. For this reason, we delay an edge collapse operation if it will prematurely connect other type 1 vertices. Typically, collapsing other edges first will quickly resolve this degenerate case.

Unfortunately, on rare occasions we can arrive at a gridlock situation in which no safe edge collapses can be performed, and we cannot produce a clean interface to the isosurface. In such situations, we simply add the surrounding cells to the existing list of marked cells and repeat the flood fill steps in Section 5.3, replacing

the nonmanifold geometry with the topologically simple isosurface.

## 7 Integration with Physics

For integration with physics animation systems, any changes to surface topology must be communicated to the physics calculations. We have integrated our surface tracker into both a finite element viscoelasticity solver and an Eulerian fluid solver. To couple the surface tracker with an embedded mesh FEM solver, the finite element mesh should be recomputed after any major topological changes. For this reason, we only calculated topological changes immediately before re-mesh events in the simulator. Coupling with an Eulerian fluid simulator is similar. We use the fluid velocity field to move the surface, and we use the surface to update the active grid cells in the simulation (the actual location of the fluid). We chose to only calculate topological changes once per frame of the output animation.

## 8 Results

In this section we describe the results of our surface tracking method using two different simulators and a variety of materials properties. Please see the accompanying video for animations of these results.

We compare our surface tracker to two other popular surface tracking methods in Figure 7. This figure shows how three different surface trackers perform on a dam-break fluid example. The left portion of the figure shows the result of using a pure level set approach with MacCormack advection [Selle et al. 2008]. Note that most of the surface details are washed away by this method. The middle image shows the result of the particle level set approach (note that we do not explicitly render the escaped particles). Here, more of the surface details are kept. The right image shows the result of our
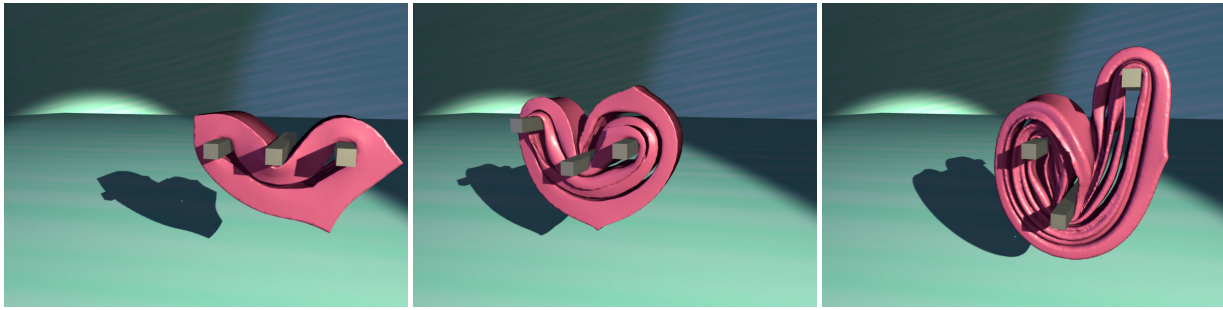
**Figure 9:** *A virtual taffy-pulling machine creates complicated surface folds.*
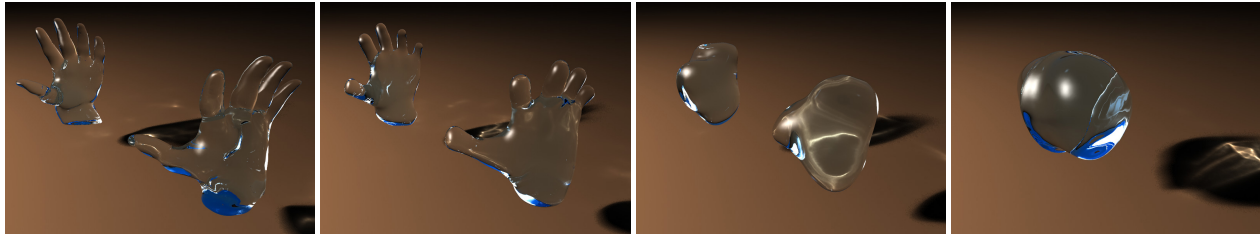


**Figure 10:** *Clapping hands influenced by strong surface tension forces.*

surface tracking method. Note the detailed features that are visible near the middle of this image where a wave has folded over onto a flat region of the liquid. The same Eulerian grid fluid solver and the same computational grid resolution of $64^3$ was used for all three simulations.

Figures 8 and 1 show examples of dropping complex geometry onto a flat surface. In Figure 8, the objects that are being dropped are model horses. The underlying visco-elastic simulation [Goktekin et al. 2004] had a resolution of $60 \times 60 \times 90$ cells in both cases, while the marching cubes reconstruction was run with a three times higher resolution. The thin legs of the horses would be difficult to resolve using only a distance field representation. Figure 1 shows several ornately carved spheres in free-fall. The details that are carved onto the surface of these models are retained by our method because regions of the surface are not re-sampled until a topological event is triggered, and most areas of the visible surface are never re-sampled at all. Note the detailed folds that are created and retained when the surfaces push into one another. Each sphere initially has more than 420 thousand triangles. In total, 15 spheres are dropped. Due to the merging of sheets, the final surface has roughly 1.6 million triangles. The simulation of this example took 63 seconds on average per frame.

Figure 9 shows a taffy pulling device. The initial torus of soft material is stretched and folded by three bars. In the absence of topological merging, each cycle of this mechanism would double the size of the surface. Our simulation merges the folded portions of the surface, which has the consequence of keeping the size of the surface tractable. Moreover, the surface of our taffy retains geometric details from the folding process that would be washed out by a level set surface tracker. The physical simulation for this example was performed using our FEM simulator.

Figure 10 shows how we can use a mesh-based surface tension technique [Brochu 2006; Wojtan and Turk 2008], effectively computing high resolution surface tension forces in an Eulerian fluid simulator. This particular example illustrates how topological changes significantly affect simulated phenomena. Two liquid hands are heavily influenced by surface tension effects, first forming individual droplets and then merging together. Without the topological change, these two droplets would simply bounce off each other. Figure 13

shows frames from an animation of several plastic blocks dropped onto a plane. The blocks immediately merge and then splash outward, creating a thin sheet. By the end of the simulation, this sheet is far thinner than the topological grid resolution, but we prevented its deletion with the simple method for topological control described in section 5.3. Our final example shows how our method can separate thin features. Figure 12 features a stiff plastic cow being stretched out and then ripped into pieces by two passing blocks.

Figure 11 shows a direct comparison between our surface tracking and other state-of-the-art methods. We performed tests with the "Zalesak's Sphere" example as defined by Enright et al. [2002a]



| LS | PLS | SLC | ours |
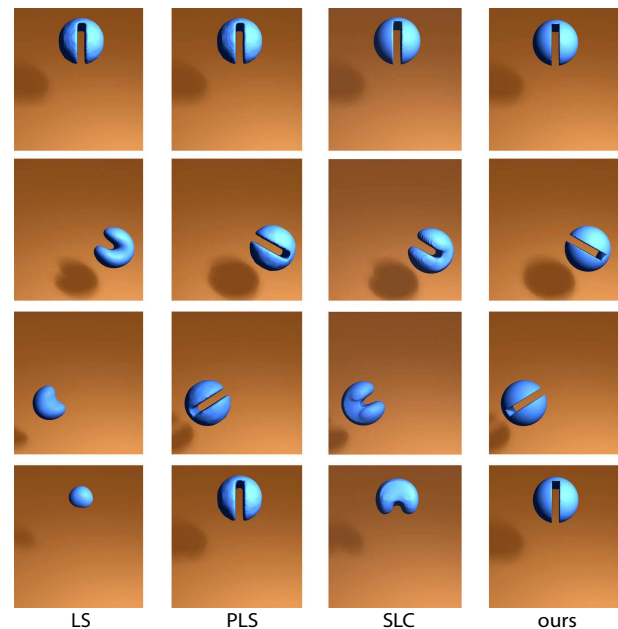
**Figure 11:** *Results from the Zalesak Sphere example for implementations of a level set (far left), particle level set (middle left), semi-Lagrangian contouring (middle right), and our method (far right).*
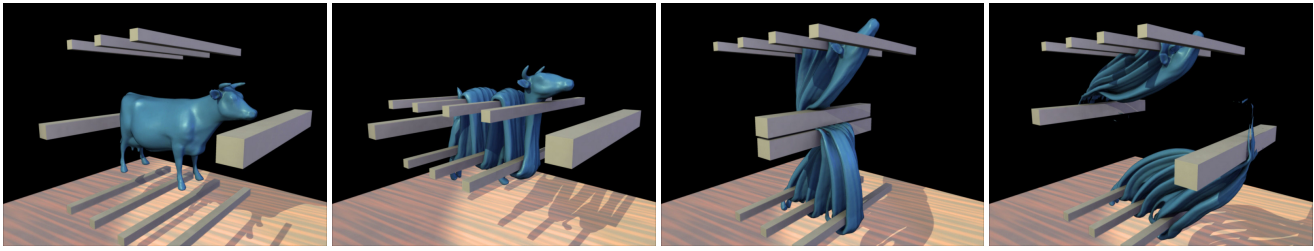
**Figure 12:** *A stretched cow that is torn when two bars scissor together.*

using implementations of a level set, particle level set, semi-Lagrangian contouring, and our method. Note that our method perfectly preserves the sharp corners because of the deep cell test.

Out of the total time spent on our surface tracking algorithm, the core topology handling takes approximately 60% of the computation. The calculation of the signed distance field requires about 20%, while the other steps, such as the mesh update, the mesh subdivision, and interfacing with the simulation code take the last 20%. The ratio between computations for surface tracking and simulation strongly depends on the complexity of each part. For the example of Figure 1, with a very complex surface mesh and a coarse simulation, the surface handling takes up the majority of the computations. For a more complex simulation, such as the $200 \times 100 \times 100$ fluid simulation of Figure 10, the fluid simulation requires roughly 2/3 of the total runtime. In this case, the overall computations took 12.9 seconds per frame on average, while for a smaller example, such as the $64^3$ simulation of Figure 7, each frame took on average slightly less than one second.

The Lagrangian FEM simulations, on the other hand, featured far fewer topological changes, so computation times were dominated by the physics simulation. The cow animation in Figure 12 finished with 12 thousand elements and 250 thousand surface triangles, the four cubes simulation in Figure 13 finished with 105 thousand elements and 206 thousand surface triangles, and the simulation of three cubes at the start of our video finished with 24 thousand elements and 45 thousand surface triangles. Each of these simulations took about 3 minutes per frame and spent less than 1% of the time on topological detection and meshing code.

## 9    Discussion and Limitations

There are several limitations to our approach. Chief among these is that the method identifies topological events and locally re-meshes based on the particular cell size of the distance field. Changes to the mesh can be noticeable if this cell size is too large. In addition, our method's topological classification will ignore small features like handles that lie completely within a cell. Another topological concern is that the extracted isosurface may not match the surface mesh at ambiguous marching cubes faces, leaving a quadrilateral hole in the mesh. We correct this behavior by triangulating the hole after the mesh surgery, though we could use more principled topological disambiguation in the future. In section 5.3, we proposed a strategy for maintaining thin sheets by re-sampling regions where surfaces merge together, and preserving regions that would split apart. This technique is effective in many of our examples, but it does not properly preserve complicated regions that exhibit both splitting and merging behavior.

Another drawback to our approach is that the re-meshing depends only on the sampled distance field, and it ignores the actual geometry of the mesh. This means that the joint between the original mesh and the new portions of the mesh can at times appear rough. We think that it may be possible to take into account more information about the original mesh during isosurface creation, and we view this as a fruitful avenue for future work. One final drawback of our approach is that the changes to the mesh may result in changes to the volume of the material. For example, when two portions of the taffy are folded together, any empty regions in the cells that are re-meshed will be turned to taffy. This effect is slight if the cell sizes are small, but it can be noticeable for long simulations. There are several approaches to volume control that can correct for this problem, but we have not yet applied any of these methods to our simulator. For example, we could accurately compute the changed volume before and after the re-meshing step and use a method such as [Kim et al. 2007] to locally correct this.

We think that our method has a number of strengths that make it attractive for simulating a variety of material types. Level set methods for surface tracking tend to aggressively merge surface components that come close to one another. This has a tendency to wash away the detailed folds that are formed when surfaces are pressed together. Our deep cell test only flags such merge events when they occur deep in the folds, and in most cases it leaves the visible portions of such folds alone. It would be possible to achieve detailed results similar to ours by using a surface tracker that *never* makes any topological changes, but simply allows pushed-together surfaces to lie next to each other. There is a costly consequence to this, however. In such a system, folded materials or complex mixing would result in a huge, overly-complex surface mesh. Most of the geometric detail would be inside folds and would never be seen, and this retained extra detail would be extremely expensive both in terms of memory and computation (updating many more vertices). Our method avoids this pitfall by joining surfaces when they meet and by keeping only the visible portion of the mesh, resulting in a considerable saving in mesh size. Our surface tracker lives between the two extremes of level sets and deforming meshes that do not allow topological changes, and our approach retains the best features of these two approaches.

Many physical simulation methods restrict the amount of detail in the visible surface to that of the underlying physics calculations, and some maintain a separate resolution for the physics calculations and the surface topology calculations. Our method provides three separate detail resolutions: the physics resolution, the topological resolution, and the visible surface resolution. We found this quite useful for exploiting low resolution physics calculations while displaying high resolution topology changes and even higher resolution surface details. We found that the physics simulator was the bottleneck for our computations, because the surface update step is relatively efficient, and because all of our topological calculations are computed sparsely in time and locally in space. We found it immensely useful to maintain decoupled resolutions for physics, topological changes, and surface detail to effectively provide the illusion of complicated behavior with inexpensive calculations.
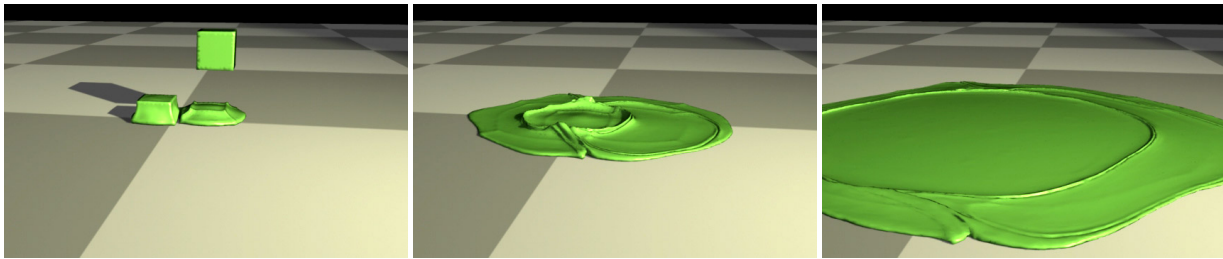
**Figure 13:** *Dropped blocks that merge and spread into a thin sheet.*

## 10    Conclusion and Future Work

We have presented a technique for updating the surface of a deforming mesh that allows topological changes. The method is robust because the topology changes are carried out using a standard isosurface creation method that only uses a sampled distance field, rather than the full complexity of the mesh. The method handles all manner of difficult situations, such as twisted and tangled surfaces. The motion of the surface mesh can be guided by any velocity field, allowing a wide latitude in choice of physics simulators.

There are a number of extensions to our basic technique that we are considering as future work. We would like to couple our surface tracking method to other kinds of simulators, such as smoothed particle hydrodynamics or the lattice Boltzmann method. Other isosurface creation methods should be easy to use instead of marching cubes. In particular, we think that a dual contouring method of isosurface creation may allow us to preserve more surface details during topology changes. Related to this is the possibility of using local mesh operations such as edge collapses that recognize and preserve sharp features. Finally, we are interested in exploring other rules for triggering topology changes that are appropriate for other materials and applications.

## Acknowledgments

## References

ADALSTEINSSON, D., AND SETHIAN, J. 1995. A fast level set method for propagating interfaces. *J. Comp. Phys. 118*, 269–277.

ADAMS, B., PAULY, M., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. *ACM Trans. Graph. 26*, 3, 48.

BARGTEIL, A. W., GOKTEKIN, T. G., O'BRIEN, J. F., AND STRAIN, J. A. 2006. A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph. 25*, 1, 19–38.

BARGTEIL, A. W., WOJTAN, C., HODGINS, J. K., AND TURK, G. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. 26*, 3, 16.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3, 100.

BISCHOFF, S., AND KOBBELT, L. 2003. Sub-voxel topology control for level-set surfaces. *Comput. Graph. Forum 22(3)*, 273–280.

BISCHOFF, S., AND KOBBELT, L. 2005. Structure preserving cad model repair. *Comput. Graph. Forum 24(3)*, 527–536.

BREDNO, J., LEHMANN, T. M., AND SPITZER, K. 2003. A general discrete contour model in two, three, and four dimensions for topology-adaptive multichannel segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 5, 550–563.

BRIDSON, R. 2008. *Fluid Simulation for Computer Graphics*. A K Peters.

BROCHU, T. 2006. *Fluid Animation with Explicit Surface Meshes and Boundary-Only Dynamics*. Master's thesis, University of British Columbia.

DU, J., FIX, B., GLIMM, J., JIAA, X., AND LIA, X. 2006. A simple package for front tracking. *J. Comp. Phys 213*, 2, 613–628.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys. 183*, 1, 83–116.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3, 736–744.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *SIGGRAPH '01*, ACM, New York, NY, USA, 23–30.

GLIMM, J., GROVE, J. W., AND LI, X. L. 1998. Three dimensional front tracking. *SIAM J. Sci. Comp 19*, 703–727.

GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. 2004. A method for animating viscoelastic fluids. *ACM Trans. Graph. 23*, 3, 463–468.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph. 24*, 3, 973–981.

HIEBER, S. E., AND KOUMOUTSAKOS, P. 2005. A Lagrangian particle level set method. *J. Comp. Phys. 210*, 1, 342–367.

HIRT, C. W., AND NICHOLS, B. D. 1981. Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *J. Comp. Phys. 39*, 201–225.

HONG, J.-M., AND KIM, C.-H. 2005. Discontinuous fluids. *ACM Trans. Graph. 24*, 3, 915–920.

IRVING, G., SCHROEDER, C., AND FEDKIW, R. 2007. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph. 26*, 3, 13.

JIAO, X. 2007. Face offsetting: A unified approach for explicit moving interfaces. *J. Comput. Phys. 220*, 2, 612–625.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90*, ACM, New York, NY, USA, 49–57.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: active contour models. *Int. Journal Computer Vision 1(4)*, 321–331.

KEISER, R., ADAMS, B., GASSER, D., BAZZI, P., DUTRE, P., AND GROSS, M. 2005. A unified Lagrangian approach to solid-fluid animation. *Proc. of the 2005 Eurographics Symposium on Point-Based Graphics*.

KIM, B., LIU, Y., LLAMAS, I., JIAO, X., AND ROSSIGNAC, J. 2007. Simulation of bubbles in foam with the volume control method. *ACM Trans. Graph. 26*, 3, 98.

KIM, T., THUEREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27*, 3, 50.

LACHAUD, J.-O., AND TATON, B. 2005. Deformable model with a complexity independent from image resolution. *Comput. Vis. Image Underst. 99*, 3, 453–475.

LIU, X.-D., OSHER, S., AND CHAN, T. 1994. Weighted essentially non-oscillatory schemes. *J. Comput. Phys. 115*, 1, 200–212.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87*, ACM, New York, NY, USA, 163–169.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM Trans. Graph. 25*, 3, 812–819.

MCINERNEY, T., AND TERZOPOULOS, D. 2000. T-snakes: Topology adaptive snakes. *Medical Image Analysis 4*, 2, 73–91.

MIHALEF, V., METAXAS, D., AND SUSSMAN, M. 2007. Textured liquids based on the marker level set. *Computer Graphics Forum 26*, 3, 457–466.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, 154–159.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3, 471–478.

NOORUDDIN, F. S., AND TURK, G. 2000. Interior/exterior classification of polygonal models. In *VIS '00: Proceedings of the conference on Visualization '00*, IEEE Computer Society Press, Los Alamitos, CA, USA, 415–422.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 137–146.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*, 12–49.

PAULY, M., KEISER, R., ADAMS, B., DUTRÉ;, P., GROSS, M., AND GUIBAS, L. J. 2005. Meshless animation of fracturing solids. *ACM Trans. Graph. 24*, 3, 957–964.

PONS, J.-P., AND BOISSONNAT, J.-D. 2007. Delaunay deformable models: Topology-adaptive meshes based on the restricted Delaunay triangulation. *Proceedings of CVPR '07*, 1–8.

REYNOLDS, C. W., 1992. Adaptive polyhedral resampling for vertex flow animation, *unpublished*. http://www.red3d.com/cwr/papers/1992/df.html.

ROSENFELD, A. 1979. Digital topology. *American Mathematical Monthly 86*, 621–630.

SELLE, A., FEDKIW, R., KIM, B., LIU, Y., AND ROSSIGNAC, J. 2008. An unconditionally stable maccormack method. *J. Sci. Comput. 35*, 2-3, 350–371.

SETHIAN, J. A. 1996. A fast marching level set method for monotonically advancing fronts. *Proc. of the National Academy of Sciences of the USA 93*, 4 (February), 1591–1595.

SETHIAN, J. A. 1999. *Level Set Methods and Fast Marching Methods*, $2^{nd}$ ed. Cambridge Monograph on Applies and Computational Mathematics. Cambridge University Press, Cambridge, U.K.

SIFAKIS, E., SHINAR, T., IRVING, G., AND FEDKIW, R. 2007. Hybrid simulation of deformable solids. In *Proc. Symposium on Computer Animation*, 81–90.

STAM, J. 1999. Stable fluids. In *SIGGRAPH '99*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128.

STRAIN, J. A. 2001. A fast semi-lagrangian contouring method for moving interfaces. *Journal of Computational Physics 169*, 1 (May), 1–22.

SUSSMAN, M. 2003. A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *J. Comp. Phys. 187/1*.

TERZOPOULOS, D., AND FLEISCHER, K. 1988. Deformable models. *The Visual Computer 4*, 306–331.

TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. 1989. Heating and melting deformable models (from goop to glop). In *the Proceedings of Graphics Interface*, 219–226.

THÜREY, N., AND RÜDE, U. 2004. Free Surface Lattice-Boltzmann fluid simulations with and without level sets. *Proc. of Vision, Modelling, and Visualization VMV*, 199–208.

TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. *ACM Trans. Graph. 25*, 3, 826–834.

VARADHAN, G., KRISHNAN, S., SRIRAM, T., AND MANOCHA, D. 2004. Topology preserving surface extraction using adaptive subdivision. In *Proceedings of SGP '04*, ACM, 235–244.

WOJTAN, C., AND TURK, G. 2008. Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27*, 3, 47.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 3, 965–972.