Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2007)
D. Metaxas and J. Popovic (Editors)

Liquid Simulation on Lattice-Based Tetrahedral Meshes

Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O'Brien, Jonathan R. Shewchuk

University of California, Berkeley

Abstract

We describe a method for animating incompressible liquids with detailed free surfaces. For each time step, semi-Lagrangian contouring computes a new fluid boundary (represented as a fine surface triangulation) from the previous time step's fluid boundary and velocity field. Then a mesh generation algorithm called isosurface stuffing *discretizes the region enclosed by the new fluid boundary, creating a tetrahedral mesh that grades from a fine resolution at the surface to a coarser resolution in the interior. The mesh has a structure, based on the body centered cubic lattice, that accommodates graded tetrahedron sizes but is regular enough to aid efficient point location and to save memory used to store geometric properties of identical tetrahedra. Although the mesh is warped to conform to the liquid boundary, it has a mathematical guarantee on tetrahedron quality, and is generated very rapidly. Each successive time step entails creating a new triangulated liquid surface and a new tetrahedral mesh. Semi-Lagrangian advection computes velocities at the current time step on the new mesh. We use a finite volume discretization to perform pressure projection required to enforce the fluid's incompressibility, and we solve the linear system with algebraic multigrid. A novel thickening scheme prevents thin sheets and droplets of liquid from vanishing when their thicknesses drop below the mesh resolution. Examples demonstrate that the method captures complex liquid motions and fine details on the free surfaces without suffering from excessive volume loss or artificial damping.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, Physically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism, Animation; I.6.8 [Simulation and Modeling]: Types of Simulation, Animation

Keywords: natural phenomena, physically based animation, computational fluid dynamics, tetrahedral mesh

1. Introduction

Convincingly animating the complex motion of a liquid's surface has proven challenging for both hand and computer-based methods. Currently, the most successful methods use computer simulation of the physical laws that govern real-world liquids. However, real liquids exhibit characteristics, such as fine surface detail and thin structures, that stress the capabilities of even the most sophisticated simulation methods.

The most commonly used methods for simulating incompressible liquids employ regular hexahedral grids or meshless collections of points, but some recent methods use unstructured tetrahedral meshes [FOK05, KFCO06, ETK*07]. Tetrahedral meshes offer advantages over regular grids: they more easily conform to complex boundaries, and tetrahedron sizes can be graded to focus computational effort where it is most beneficial. Tetrahedral meshes have been successful in simulations with no free surfaces—for instance, where a gas flows through a static simulation domain. But simulating free surfaces is substantially harder because the meshes must track the movement of those surfaces, and substantially more tetrahedra are usually necessary to faithfully track the surface detail and discretize the thin features that arise.

Moreover, dynamic liquids often form droplets, filaments, or thin sheets, which can disappear because of the finite resolution of the mesh. These artifacts are often called *volume loss*. In real life, surface tension prevents liquids such as water from forming arbitrarily small structures, but faithful simulation of surface tension effects on thin sheets exacts a prohibitive computational cost for the amount of water typically used in animations. For thin features, surface tension acts at a scale that is well below the finest feasible resolution in a simulation used for animation.

We address these problems with several techniques that are, to our knowledge, novel in fluid animation: a fast algorithm for generating semi-regular, graded, guaranteed-quality tetrahedral meshes that conform to a liquid's boundary; a novel thickening strategy for reducing volume loss

Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2007, San Diego, California, August 03 - 04, 2007

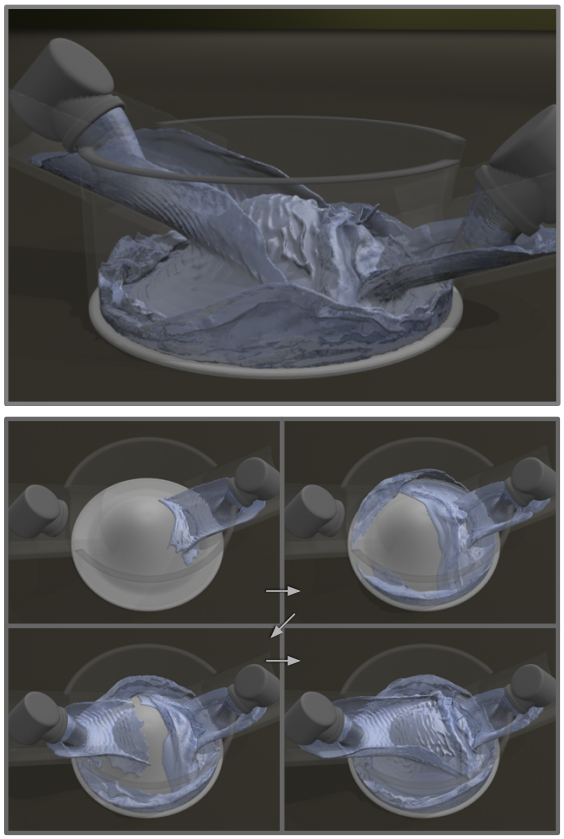

Figure 1: *Two views of a liquid pouring down a pair of chutes into a transparent container.*

and preventing the artifact of disappearing droplets or sheets of fluid; and the use of algebraic multigrid to speed up the pressure projection step commonly used to enforce the divergence-free velocity property of incompressible fluids. These techniques make it feasible to use large tetrahedral meshes.

The mesh generation algorithm we use, called *isosurface stuffing* [LS07], creates a tetrahedral mesh that conforms accurately to smooth liquid boundaries. It tiles space using the vertices and the octant centers of a balanced octree that encloses the fluid region. If the octree were refined so that all its leaf octants had the same size, the vertices in the interior of the tetrahedral mesh would lie on a body centered cubic lattice. Instead, the octree is refined so that the liquid surface (where high resolution is needed) is covered with fine leaf octants (all of the same size), but coarser leaf octants appear in the liquid interior, keeping the number of elements (and thus the computational cost) from exploding. The mesh vertices near the surface are warped, and some tetrahedra are subdivided, so that the mesh boundary conforms to the fluid surface. Isosurface stuffing is fast, and it guarantees that all the tetrahedra have good quality, with their dihedral angles bounded between 10.7° and 164.8°. We exploit the semi-regular structure of the mesh to obtain significant savings in memory and computation by reusing geometric information for identical tetrahedra and by using the octree for point location (needed for semi-Lagrangian advection).

To track the moving fluid surface we use *semi-Lagrangian contouring* (SLC), first developed by Strain [Str99, Str01] and then extended to three dimensions by Bargteil *et al.* [BGOS06]. At the beginning of a time step, SLC computes a new fluid boundary, in the form of a high-resolution surface triangulation, from the previous time step's fluid boundary and velocity field. From the new fluid boundary, isosurface stuffing creates a new tetrahedral mesh. The surface triangulation is used for rendering and as part of SLC, and the tetrahedral mesh is used for computing the fluid velocities in the current time step.

Although semi-Lagrangian contouring tracks a fluid's free surface well, sheets, filaments, and droplets of fluid that become thinner than the grid's finest resolution are lost, as with any grid-based level set tracking method. This behavior leads to noticeable volume loss and ugly artifacts of disappearing liquid. To combat this problem, we propose a simple particle-based method that mimics some of the effects of surface tension, albeit at a much coarser scale, without the excessive cost of simulating surface tension in a physically accurate way.

Our particle-based method thickens regions where the fluid is dangerously thin by redistributing mass from nearby thin regions (which causes tendrils of liquid to break up rather than become thinner) and adding some thickness artificially. A few parameters allow the user to control the effects of the thickening algorithm.

Finally, we show how an algebraic multigrid solver can be used to efficiently perform the pressure projection required to ensure that the fluid behaves incompressibly. Our algebraic multigrid solver proved to be easy to implement, despite the complex free surfaces. It significantly outperforms the commonly used alternative of preconditioned conjugate gradients.

Our results, like the example shown in Figure 1, show that tetrahedral meshes can be used to animate liquids that feature substantial surface detail and thin sheets of liquid. The graded tetrahedron sizes and the use of a triangular surface mesh even finer than the boundary of the tetrahedral mesh make it possible to capture fine simulation detail at the surface, while using a computationally feasible number of tetrahedra.

2. Related Work

Fluid simulation by numerical methods has been widely adopted by the visual effects industry. Early computer graphics techniques for simulating smoke and water [FM96, Sta99, FF01] are still commonly used. These methods have been extended to more exotic materials and phenomena, including fire [NFJ02], explosions [FOA03], viscoelastic materials [GBO04], and bubbles [ZYP06]. Techniques are

available to model the interaction of fluids with rigid bodies [CMT04], deformable bodies [CGFO06], thin membranes [GSLF05], and other fluids [LSSF06]. Treuille *et al.* [TLP06] use model reduction techniques to make fluid simulations run in real time.

It is difficult to realistically simulate free liquid surfaces with good detail, stability, and speed. Methods to enhance detail without losing too much speed include vorticity confinement [FSJ01] and vortex particles [SRF05]. Thürey *et al.* [TKPR06] propose methods for maintaining detail when applying control forces to guide a simulation toward keyframes.

Although most papers discuss regular hexahedral grids and Eulerian discretization, some focus on more flexible geometric structures. Losasso *et al.* [LGF04] use an octree to simulate gases and liquids. Klingner *et al.* [KFCO06] use an unstructured tetrahedral mesh for gases and smoke. Some authors eschew grids entirely in favor of meshless collections of Lagrangian particles [CD97, DC96, MKN*04, PTB*03, TPF89].

We choose semi-structured tetrahedral grids. Mesh generation is a huge field, too large to summarize here; see Bern and Eppstein [BE92] and Owen [Owe98] for surveys. We use isosurface stuffing [LS07] because no other boundary-conforming meshing algorithm offers the same combination of speed, numerical robustness, and guaranteed quality.

Surface tracking is crucial both for simulation (determining where the liquid is) and for generating a surface to render. Methods based on grids and isosurfaces, such as the particle level set method (PLS) [EMF02] and semi-Lagrangian contouring (SLC) [BGOS06] are largely successful at creating high-quality surfaces. Their main difficulty is the disappearance of features smaller than the resolution of the grid. Proposals for coping with this volume loss include replacing the thin sheets with particles [GSLF05, KCC*06]. Our approach is to combine SLC with a variation of the particle-based surface tracking method of Zhu and Bridson [ZB05].

3. Simulation Methods

Our simulation steps are illustrated in Figure 2. At the beginning of time step t, we have a triangulation S_t of the liquid surface and a tetrahedral mesh M_t that approximates the volume bounded by S_t. A time step consists of the following operations.

1. Create a new surface mesh S_{t+1} for the next time step by a combination of semi-Lagrangian contouring and our thickening method.
2. Generate a tetrahedral mesh M_{t+1} that approximates the volume bounded by the liquid surface S_{t+1}, with some additional thickening.
3. Advect the fluid velocity field from M_t to M_{t+1} using generalized semi-Lagrangian advection [FOKG05]. Update the velocities on M_{t+1} by explicit time integration of the external forces, and apply pressure correction to make the velocities divergence-free.

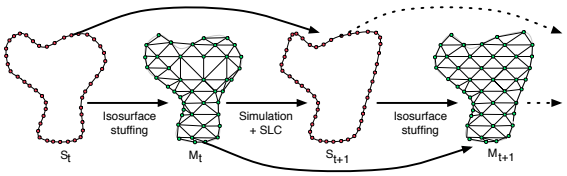

Figure 2: *Each time step uses one surface triangulation and one tetrahedral mesh. A new time step's surface S_{t+1} is created by semi-Lagrangian advection: tracing paths backward in time through the velocity field on M_t to locate a point relative to S_t. The isosurface stuffing algorithm generates a tetrahedral mesh M_{t+1} (with coarser surface resolution) from S_{t+1}. Velocities are interpolated/extrapolated from M_t.*

The following sections discuss these steps in detail. Many of the components we use are taken from prior work, and we focus on how to integrate and change the components so they work together.

3.1. Governing Equations and Discretization

Inviscid, incompressible liquids are governed by the Euler equations,

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\,\mathbf{u} + \frac{\mathbf{f}}{\rho} - \frac{\nabla p}{\rho}, \tag{1}$$

subject to the volume conservation constraint

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

where $\mathbf{u}$ is the fluid velocity field, p is the pressure field, t is time, ρ is density (which we will assume constant everywhere), $\mathbf{f}$ is a field of external forces, and $\nabla = [\partial/\partial x, \partial/\partial y, \partial/\partial z]^{\mathsf{T}}$.

We integrate Equation (1) forward in time using operator splitting; itegrating each term of Equation (1) separately as follows.

$$\mathbf{u}^{*} = \text{Advect}(\mathbf{u}_t), \tag{3}$$

$$\mathbf{u}^{**} = \mathbf{u}^{*} + \frac{h}{\rho}\mathbf{f}, \tag{4}$$

$$\nabla^2 p = \frac{\rho}{h}\nabla\cdot\mathbf{u}^{**}, \text{and} \tag{5}$$

$$\mathbf{u}_{t+1} = \mathbf{u}^{**} - \frac{h}{\rho}\nabla p, \tag{6}$$

where h is the length of a time step, $\mathbf{u}_t$ and $\mathbf{u}_{t+1}$ are velocity fields at two successive time steps whose domains are two successive tetrahedral meshes M_t and M_{t+1}, and $\mathbf{u}^{*}$ and $\mathbf{u}^{**}$ are intermediate velocity fields over M_{t+1}.

To implement Equation (3), velocities are advected by the method of Feldman *et al.* [FOKG05], a generalization of the semi-Lagrangian advection method of Stam [Sta99]. Specifically, we assign velocities to the new mesh M_{t+1}, where velocities are stored at face barycenters, by tracing the path of each barycenter backward in time through the piecewise linear velocity field defined over M_t. When we reach the end of a path, we determine the velocity at the endpoint by interpolation on M_t, and assign it to the barycenter in M_{t+1}. (To

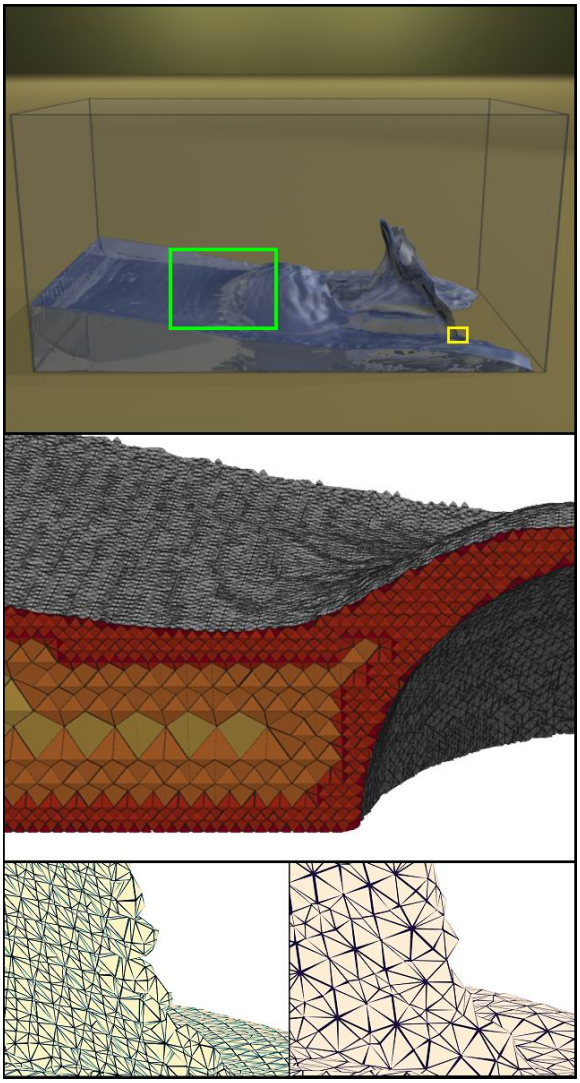

Figure 3: *Top: A wave of liquid flowing over a hemispherical obstacle. Center: Cutaway view of the tetrahedra enclosed by the green box in the top image. Bottom: Portions of the surface triangulation and tetrahedral mesh, respectively, enclosed by the yellow box in the top image.*

be precise, we only assign the scalar component of velocity normal to the face.)

Equations (5) and (6) together perform *pressure correction*, which ensures that $\mathbf{u}_{t+1}$ satisfies the volume conservation (divergence-free velocity) condition, Equation (2). Observe that Equation (5) entails the solution of a Poisson equation. We discretize it with a finite volume method, yielding a system of linear equations.

We take our methods for discretizing the derivative operators and interpolating the velocities from Klingner *et al.* [KFCO06]. We summarize them briefly here; see the original work for details. We store one pressure value at the center of each tetrahedron. At the center of each triangular face, we store one scalar value representing the normal component of the velocity.

The gradient of the pressure normal to a face f is denoted

$$\nabla p \cdot \mathbf{n}_f = \frac{p_2 - p_1}{d_f}, \tag{7}$$

where p_1 and p_2 are the pressures at the centers of the two tetrahedra adjoining f, and d_f is the perpendicular distance between those centers. The average divergence within a tetrahedron is denoted

$$(\nabla \cdot \mathbf{u})_{\text{avg}} = \frac{1}{V} \sum_{i=1}^{4} z_i A_i s_i, \tag{8}$$

where V is the volume of the tetrahedron, and for each tetrahedron face indexed $i \in \{1,2,3,4\}$, A_i is the area of face i, z_i is the velocity component normal to face i, and s_i is a sign ($+1$ or -1) that orients face i's normal outward.

To interpolate velocities within a tetrahedral mesh, we first compute a velocity vector at each tetrahedron center from its four face normal velocities, following Elcott *et al.* [ETK*07]. As the authors point out, this step does not cause artificial smoothing. To interpolate at an arbitrary point, we use generalized barycentric interpolation [WSHD04] over dual cells of the mesh.

When we trace paths backward through time for semi-Lagrangian advection (to advect both the free surface and the velocities), we find that using a smoothed version of the velocity field for integrating the path does not produce objectionable artifacts. We take advantage of this observation to speed up advection substantially: we compute velocity vectors at the primal mesh vertices by generalized barycentric interpolation; then based on these vectors, we use standard barycentric interpolation (which is appreciably faster) over the tetrahedra to perform backward tracing.

However, once we have traced a path back to its endpoint one time step earlier, we use the more expensive generalized barycentric interpolation to evaluate the velocity at the endpoint. If we were to use standard barycentric interpolation for this evaluation, successive errors would accumulate over time, manifesting as artificial damping of the velocity field, as Klingner *et al.* [KFCO06] report.

We differ from Klingner *et al.* by storing pressures and velocities at the barycenters of the tetrahedra and triangular faces, instead of at the circumcenters. The gradient and interpolation operators were designed with the geometric properties of circumcenters in mind, but we find that storing quantities at the barycenters works well in practice.

3.2. Surface Tracking by Semi-Lagrangian Contouring

Surface tracking is the act of computing an updated liquid boundary S_{t+1} from the previous time step's boundary S_t and the velocity field $\mathbf{u}_t$ defined over the mesh M_t. We use semi-Lagrangian contouring (SLC), devised for two dimensions by Strain [Str01] then extended to three dimensions by

Bargteil *et al.* [BGOS06] (who also provide an open-source implementation). We briefly review the method here.

SLC begins by defining a *signed distance function* $f(\mathbf{p})$ over $\mathbb{R}^3$ that is positive inside S_t, negative outside, and zero on the boundary. See Bargteil *et al.* for details on how to use an octree to approximate the signed distance function. (The function is exact near the surface, but at a distance we approximate by interpolating over the octree.) Note that $f(\mathbf{p})$ is the function used as an input to isosurface stuffing (described in Section 3.3) to construct the graded tetrahedral mesh M_t, which approximately fills the surface $S_t = \{\mathbf{p} : f(\mathbf{p}) = 0\}$. (The resolution of M_t is chosen so its boundary edges generally have twice the length of the edges of S_t.)

The next step is to implicitly construct a second signed distance function $f^+(\mathbf{p})$ that represents the liquid surface after it moves for the duration of a time step. The final step is to use Marching Cubes to construct a high-resolution triangulated approximation S_{t+1} of the surface $\{\mathbf{p} : f^+(\mathbf{p}) = 0\}$.

To calculate the function $f^+(\mathbf{p})$ at a given point $\mathbf{p}$, we use the same semi-Lagrangian advection method, described in Section 3.1, that we use to advect the velocity field: trace a path from $\mathbf{p}$ backward in time through the velocity field $\mathbf{u}_t$ defined over M_t. We also modify the function $f^+(\mathbf{p})$ to prevent penetration of obstacles, as described in Section 3.3.1, and to thicken the liquid where necessary, as described in Section 3.4.

3.3. Mesh Generation

The tetrahedral meshes we use for simulation conform to the volume occupied by the fluid, as Figure 3 shows. This approach has several merits. First, it simplifies the handling of boundary conditions during pressure correction, as the mesh boundary is the fluid boundary. Second, computation is only expended on the region occupied by fluid. Lastly, because the mesh is recreated each frame, we can grade the mesh so that it is fine near the surface, where we want detail, but coarse in the interior, speeding up every part of our simulation method.

Our mesh generation algorithm, called isosurface stuffing [LS07], takes a signed distance field as its input, and produces a mesh whose boundary approximates the field's zero-surface, so the algorithm enjoys a natural marriage with a level-set surface tracking method, in our case semi-Lagrangian contouring. It builds a balanced octree that covers the liquid surface with leaf octants, all the same size. This octree serves as a skeleton for a tetrahedral *background grid* inspired by the Delaunay triangulation of the body centered cubic lattice. (Producing the background grid is not quite as simple as subdividing each octant into tetrahedra; some tetrahedra span two octants.) Vertices of the background grid that are too close to the liquid surface are warped so that they lie on the surface. New vertices are added where edges of the background grid cut the surface. Finally, some background tetrahedra are subdivided into smaller tetrahedra to accommodate the new vertices, using stencils of precomputed tetrahedra in the same fashion that Marching Cubes uses stencils of precomputed triangles. All the vertices on the boundary of the final mesh lie on the surface of the liquid.

Isosurface stuffing is extremely fast and numerically robust because it does little geometric computation (compared to most meshing algorithms). The tetrahedra it produces are mathematically guaranteed to have high quality. (See the original paper [LS07] for details.) It is these features that collectively make it feasible to generate a new, large mesh for every time step.

3.3.1. Modifications for Obstacles

Our simulations include interactions with solid obstacles such as walls and containers. Semi-Lagrangian contouring generally produces a signed distance function that does not perfectly conform to obstacles, because of numerical errors and the discretization of time and the velocity field. To prevent liquid from penetrating a rigid obstacle, we build a signed distance function for the obstacle. The input to our mesher is a function that is the minimum of two signed distance functions—one that is positive inside the liquid, and one that is negative inside the obstacle.

The opposite problem can occur too: persistent thin gaps between a liquid and an obstacle. We fix these problems after the mesh is built. Vertices that are within one eighth of the edge length of the leaf octants from an obstacle are "snapped" onto the obstacle boundary by orthogonal projection. We have found that snapping vertices this way does not introduce badly shaped tetrahedra.

We classify a face of the mesh as a liquid-obstacle interface if all of its vertices lie on the obstacle. All other boundary faces are liquid-air interfaces. When we build the linear system for the discretization of Equation (5), we treat liquid-obstacle faces with closed, non-slip boundary conditions, and we treat liquid-air faces with open boundary conditions, following Klingner *et al.* [KFCO06].

3.3.2. Exploiting Properties of the Mesh

Isosurface stuffing makes it feasible to use a large number of tetrahedra (in our examples, about 1–3 million). Besides its speed and numerical robustness, a more subtle benefit is that its meshes are semi-structured: in the interior, they use only four shapes of tetrahedra (in graded sizes). In our simulations, roughly two thirds of the tetrahedra have one of these four shapes. Therefore, many tetrahedra have the same volume, face areas, and face normals (possibly scaled by a power of two), which are needed for our finite volume discretization (Section 3.1). We save a substantial amount of memory by not storing these quantities for tetrahedra having standard shapes. Thus we realize some of the storage savings usually associated with structured grids, without sacrificing grading or boundary conformity.

Another way we exploit the structure of our meshes is by using the octree for point location. When we perform semi-Lagrangian advection of the velocities, we need to trace each

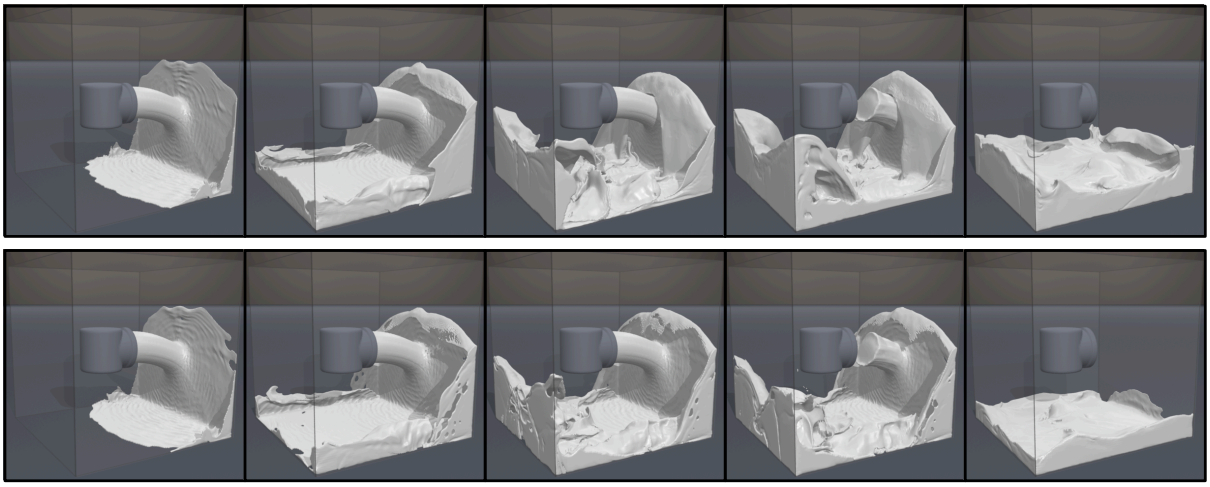

Figure 4: *A comparison of images taken from animations generated with our thickening scheme (top) and without (bottom). The example with no thickening suffers from visible holes and artifacts where thin structures vanish. Observe the difference in liquid volume between the last frames of the two sequences: without thickening, the simulation has lost appreciable volume.*

face barycenter of the new mesh M_{t+1} backward in time through the velocity field defined over the old mesh M_t. Likewise, advection of the surface also entails tracing a path through M_t. The first step in doing so is to find the tetrahedron in M_t that contains the starting point. We walk down the octree until we reach the finest octant containing the point, then we use standard walking point location to home in on the right tetrahedron.

Occasionally, a path is traced right outside the mesh M_t, so we need to extrapolate velocities. For this task, we use the efficient octree extrapolation technique of Losasso *et al.* [LGF04] on our mesh generator's octree.

3.4. Thickening

Like other grid level-set surface tracking methods, semi-Lagrangian contouring cannot resolve sheets or tendrils of fluid that become thinner than the leaf octants of the octree used store the signed distance function. The visual results can include small droplets spontaneously vanishing, large holes opening up in thin sheets of liquid, and rapid loss of total fluid volume, as shown in the bottom row of Figure 4. Moreover, the resolution of the tetrahedral mesh M_t is typically a factor of two coarser than the resolution of the surface S_t. Although this sampling strategy allows us to maintain finely detailed surfaces, it can create circumstances where M_t has no tetrahedra near a thin part of the fluid volume bounded by S_t, so the fluid cannot be simulated.

To solve both problems, we have developed a thickening technique that modifies the signed distance function $f(\mathbf{p})$ in regions where the fluid is thin. We heuristically detect these regions and place particles approximately on their medial axes. The radii of the particles are proportional to the distance from the medial axis to the surface (Section 3.4.1). These particles thicken the liquid by modifying the signed distance function $f(\mathbf{p})$ (Sections 3.4.2 and 3.4.3). A second-order Runge-Kutta method advects the particles forward through the velocity field $\mathbf{u}$.

We use different thickening approaches for surface tracking and mesh generation, because we desire different behavior from each. For surface tracking, we prefer a smooth surface to a bumpy one, even at the cost of some volume loss. For meshing, we do not want to leave any of the surface S_t hanging outside the simulation mesh M_t, but bumps in the mesh surface are harmless and do not affect the appearance of the rendered liquid surface.

Given that we use particles in thin regions, why do we not use them everywhere as a replacement for semi-Lagrangian contouring? In our experience, particle level set methods tend to create bumpy surfaces. These can be fixed by smoothing, albeit with a concomitant loss of detail. We turned to SLC because it gives us better detail for a fixed resolution grid.

3.4.1. Medial Axis Estimation

The medial axis of a surface is the set of centers of spheres that touch the surface at more than one point, but are entirely enclosed by the surface [Blu67]. Three-dimensional medial axes are unstable and difficult to compute. Several researchers, such as Foskey *et al.* [FLM03], propose simplified approximations of the medial axis, but even these are computationally expensive. We do not need much accuracy, and we need the approximation only in thin regions, so we have found the following simple strategy to be adequate.

For each vertex $\mathbf{x}_i$ of S_t, shoot a ray from $\mathbf{x}_i$ in the negative normal direction $-\mathbf{n}_i$ as illustrated in Figure 5, and check if the ray hits S_t again within a "thinness" threshold γ. If so, compute the midpoint $\mathbf{m}_i$ between $\mathbf{x}_i$ and the first point where the ray intersects S_t. The point $\mathbf{m}_i$ is a candidate that may be near the medial axis of the thin region. We estimate the minimum distance d_i from $\mathbf{m}_i$ to S_t, using the

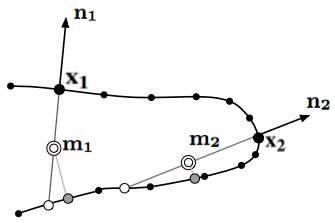

Figure 5: *Finding points that lie approximately on the medial axis. In this example,* $\mathbf{m}_1$ *is used but* $\mathbf{m}_2$ *is rejected.*

signed distance function for S_t, and throw away $\mathbf{m}_i$ if d_i is substantially shorter than the distance from $\mathbf{x}_i$ to $\mathbf{m}_i$ (see Figure 5). For surface thickening, we choose γ to be 1.2 times the edge length of the smallest leaf octant of the octree that represents S_t (and the signed distance function). For mesh thickening, we choose γ to be 1.2 times the edge length of the mesh generator's smallest leaf octants.

3.4.2. Mesh Thickening

Given the surface S_t, we define its signed distance function $f(\mathbf{p})$. From $f(\mathbf{p})$ and the particles sampled near the medial axis of S_t, we create a "thickened" function $f^*(\mathbf{p})$, which isosurface stuffing takes as input to generate the mesh M_t.

The thickened function f^* is the sum of the original function f and an *offset function* that thickens $f^*(\mathbf{p})$ enough so that M_t encloses the entire liquid surface S_t. The offset is

$$\text{offset}(\mathbf{p}) = \frac{\sum_i W(\mathbf{p}-\mathbf{m}_i,h)(\gamma_{\text{mesh}}-d_i)}{\sum_i W(\mathbf{p}-\mathbf{m}_i,h)},$$

where γ_{mesh} and h are respectively 0.6 times and 4 times the edge length of the mesh generator's smallest leaf octant. Following Müller *et al.* [MCG03], W is the kernel

$$W_{\text{poly6}}(\mathbf{z},h) = \frac{315}{64\pi h^9}\begin{cases}(h^2-|\mathbf{z}|^2)^3 & 0\le\mathbf{z}\le h,\\ 0 & \text{otherwise.}\end{cases}$$

3.4.3. Surface Thickening

We compute the next surface S_{t+1} by thickening the signed distance function $f(\mathbf{p})$ for S_t. Thickening of $f(\mathbf{p})$ for S_t is done differently then the thickening method used for input to the meshing algorithm (Section 3.4.2) because our goal for the surface is to prevent liquid disappearance while maintaining a smooth surface and avoiding creating too much extra volume. We prevent excessive volume gain by redistributing the volume in medial axis particles from one location to another, in a way that tends to make thin regions break apart.

After the medial axis particles have been advected, we divide them into two groups, Q_{small} and Q_{big}. Q_{small} contains particles with $d_i \le \beta_{\text{slc}}$ and Q_{big} contains those with $d_i > \beta_{\text{slc}}$, where d_i is the thickness defined in Section 3.4.1, and β_{slc} depends on the size of the background contouring grid (see below). For each particle $q_s \in Q_{\text{small}}$, we look for the subset Q_{near} of particles in Q_{big} that are within a distance of r_{redis} from q_s. We then redistribute the radius of q_s to the particles in Q_{near} by uniformly incrementing the radius of particles in Q_{near} by $d_s/|Q_{\text{near}}|$. Although it might seem more sensible to distribute volume instead of radius, or to non-uniformly distribute based on distance from q_s, our strategy is fast and works well in practice.

After the redistribution step, we increment the radius of the particles in Q_{big} by r_{ib} (see below). We delete from Q_{small} the particles that found nearby Q_{big} particles in the step above, and increase the radius of the remaining particles in Q_{small} by r_{is}, thereby thickening the liquid slightly in the thin regions.

In the animations we present in Section 4, β_{slc} is 0.5γ, r_{ib} is 0.25 times the smallest grid size, and r_{is} is 0.15 times the smallest grid size. By choosing r_{redis} to be between 0 and 3 times the smallest grid size, we control how difficult it is to break thin sheets of liquid apart.

After we update the radius of the particles, we construct an implicit function L_{par} defined by the particles' positions and radii, as described by Zhu and Bridson [ZB05]. We unite L_{par} with the signed distance function f generated by the original SLC method by defining a new signed distance function

$$f^{**}(\mathbf{p}) = \max(f(\mathbf{p}), L_{\text{par}}(\mathbf{p})),$$

whose zero-surface is triangulated by Marching Cubes to yield the new surface S_{t+1}.

3.5. Algebraic Multigrid for Incompressibility

We enforce the incompressibility of the fluid by solving Poisson's equation (5), yielding a pressure field p such that the final velocity field $\mathbf{u}_{t+1}$, computed by Equation (6), is divergence-free. Discretizing Equation (5) yields a large, sparse, symmetric linear system, which we solve efficiently using algebraic multigrid.

Multigrid speeds up a standard iterative method by using it to compute a quick, inaccurate solution, then correcting the error in the approximate solution with help from an approximate solution to a coarser version of the problem, which can be solved more quickly. The coarser version is recursively solved by using an even coarser version of the problem. The base case is a linear system small enough to solve quickly with a direct method. See McCormick [McC87] or Trottenberg *et al.* [TOS01] for a detailed description of multigrid.

Suppose the system we wish to solve is $A_q\mathbf{x}_q = \mathbf{b}_q$, where $\mathbf{x}_q$ is unknown. The hierarchy of linear systems is $A_k\mathbf{x}_k = \mathbf{b}_k$, with k ranging from 1, the coarsest system, to q, the finest. Each A_k in some sense approximates the finest matrix A_q.

Multigrid uses a sequence of linear *prolongation operators* $P_{k,k-1}$ that expand a coarse solution (with few unknowns) to a finer one (with more), and a sequence of linear *restriction operators* $R_{k-1,k}$ that reduce a fine solution to a coarser one (with some loss of information). In the original, geometric formulation of multigrid, the fine and coarse

solutions are nodal values on regular grids of two different resolutions, prolongation is done (exactly) by interpolating, and restriction is done (approximately) by projection. The matrices for the coarser levels, A_k are formed by applying the discretization stencils on the coarsened meshes.

V-cycle multigrid begins by applying a *relaxation operator* $\mathbf{S}(A_k, \mathbf{x}_k, \mathbf{b}_k)$, which returns an approximation solution (usually with poor accuracy) to the system $A_k\mathbf{x}_k = \mathbf{b}_k$. We use two Gauss-Seidel iterations. Next, it computes the residual of the system, restricts it to the next coarser grid, and recursively solves the coarse problem on the restricted residual. It prolongs the coarse solution and adds it, as a corrective term, to the fine solution. Finally, the relaxation operator is applied again. We use full-cycle multigrid, which uses V-cycle multigrid as a subroutine as follows.

```
VCycleMultigrid(x_k, b_k, k)
    if k = 1
        x_k ← A_1^{-1} b_k
    else
        x_k ← S(A_k, x_k, b_k)
        r_{k-1} ← R_{k-1,k}(b_k − A_k x_k)
        w_{k-1} ← 0
        VCycleMultigrid(w_{k-1}, r_{k-1}, k − 1)
        x_k ← x_k + P_{k,k-1} w_{k-1}
        x_k ← S(A_k, x_k, b_k)
FullCycleMultigrid(x, b)
    r_q ← b − A x
    for k ← q to 2
        r_{k-1} ← R_{k-1,k} r_k
    z_1 ← A_1^{-1} r_1
    for k ← 2 to q
        z_k ← P_{k,k-1} z_{k-1}
        VCycleMultigrid(z_k, r_k, k)
    x ← x + z_q
```

Simulation domains with sinuous shapes make it difficult to geometrically coarsen a mesh in a way suitable for multigrid—clustering unstructured tetrahedra does not create larger tetrahedra. This problem can occur even for regular hexahedral grids on irregularly shaped domains. We circumvent this problem with algebraic multigrid, which uses only the finite volume matrices, and not the geometry, to construct the prolongation and restriction operators. From the prolongation and restriction operators the system matrices can be defined by $A_{k-1} = R_{k-1,k}A_kP_{k,k-1}$.

To form the restriction and prolongation operators, we partition the level k variables into *coarse* and *fine* variables. Coarse variables survive into the coarser level $k-1$; fine variables do not. $P_{k,k-1}$ is a matrix that prolongs a level $k-1$ solution by assigning each fine level k variable the average of the values of its coarse neighbors from the level $k-1$ solution. (Variables i and j are *neighbors* if A_{ij} is non-zero.) Some researchers advocate using weighted averages based on matrix coefficients [McC87], but we find that simple averages converge faster. They also save memory and time by obviating the need to compute weights. For restriction, we use the popular Galerkin projection method [TOS01], in which $R_{k-1,k} = P_{k,k-1}^{\mathsf{T}}$.

	Sim	Mesh	SLC	Tets (M)	Tris (M)
Chutes	61	59	350	1.0	2.1
Spray (T)	37	53	169	0.8	1.4
Spray (NT)	40	54	176	0.8	1.4
Angel	30	24	276	1.0	1.4
Dam Break	49	42	196	1.3	1.2

Table 1: *Simulation times for our examples, in seconds per frame of animation.* Sim *denotes the time used for velocity advection and pressure correction.* Mesh *is the time for mesh generation.* SLC *is the time used by the surface tracker. The last two columns show the average numbers, in millions, of tetrahedra in the simulation mesh and triangles in the surface triangulation. (T) and (NT) indicate thickening and no thickening.*

Each level k variable must have at least one neighbor labeled *coarse* (possibly itself). A good partition has *coarse* variables that support good interpolation of values from the coarser to the finer level, but also keep the number of *coarse* variables modest, so that the coarser linear system is as small as possible. The following greedy algorithm for labeling variables, though not optimal, strikes a good balance between simplicity and effectiveness in practice.

1. Assign each variable a counter storing its number of neighbors.
2. Select a variable at random; label it *coarse*.
3. Label the neighbors of the new *coarse* variable *fine*.
4. For each unlabeled neighbor of each new *fine* variable, decrement the count of its number of unlabeled neighbors.
5. Select the unlabeled variable that neighbors a *fine* variable and has the most unlabeled neighbors. Label it *coarse*.
6. Repeat steps 3 to 5 until every variable is labeled.

Our algebraic multigrid scheme is not novel, but we are not aware of its prior use in graphics for fluid animation. Our simulations use between 0.8 million and 1.3 million tetrahedra. For these systems, our algebraic multigrid solver is between 1.8 and 2.1 times faster than a preconditioned conjugate gradient solver. In a test with a five million tetrahedron mesh, our solver was three times faster than preconditioned conjugate gradients. We believe that the solver would work similarly well for fluid simulations with other discretizations, and that the performance benefit would grow for larger systems.

4. Results and Discussion

We have a prototype implementation, written in C++, which we used to animate several scenarios, depicted in Figures 1, 4, 6, and 7 (also seen in Figure 3). We ran the simulations single-threaded on 2.8 or 3.2 GHz Intel processors with 3 or 4 GB of memory. All figures were rendered by PIXIE.

Table 1 shows the running times. The cost of velocity advection and pressure projection is roughly equal to the cost

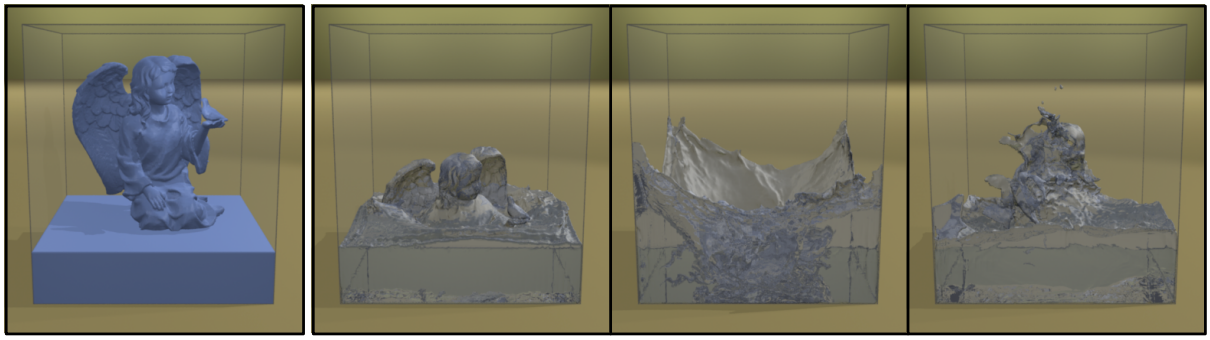

Figure 6: *A sequence beginning with a fluid configuration in the shape of an angel. Once the simulation begins, the structure falls, creating a complex splash pattern.*

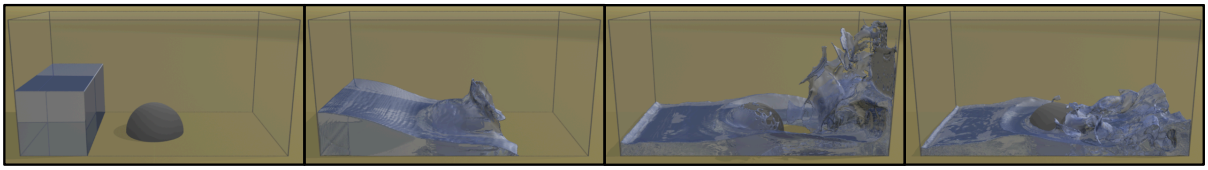

Figure 7: *A sequence wherein a block of liquid is released at one side of a closed container. The wave of water flows over an obstacle and crashes against the far side of the container.*

of mesh generation. The running times of the spray examples with and without thickening show that the cost of thickening is negligible. In all our simulations, surface tracking dominates the overall cost. More than a third of the surface tracking time is in a fast marching method that approximates the signed distance function. Most of the remaining time is spent tracing the velocity field backward. We believe there is substantial room for speed improvement through faster algorithms for signed distance approximation and path tracing. For example, it might be possible to exploit temporal coherence to avoid recomputing the signed distance field from scratch at every time step, and our point location and path tracing does not take advantage of spatial locality.

Our methods are not without flaws. Some of our examples exhibit a small amount of volume loss. However, we are not aware of a liquid simulation method that preserves volume perfectly. Our thickening scheme has adjustable parameters that allow a user to reduce this effect, although excessive use of the thickening scheme can result in volume *gain* artifacts. Another imperfection is that our discretization of the Poisson operator is not linearly consistent, so our free surfaces move even when gravity is the only force applied to a pool of water. This artifact is not apparent in dynamic scenes, but it is visible in scenes with very slow moving water. The octree method of Losasso *et al.* [LGF04] exhibits similar artifacts when the octree grades from coarse to fine along a direction orthogonal to the direction of gravity.

Future work includes adding viscoelasticity to a simulation. This goal requires additional differential operators, defined in a consistent and stable manner. It is probably straightforward to extend our methods to multiple-phase liquids, as the meshing algorithm can create high-quality tetrahedra on both sides of an interface. We could also couple our liquids to moving rigid bodies using the methods of Klingner *et al.* [KFCO06] and Chentanez *et al.* [CGFO06], who also use tetrahedral meshes.

Acknowledgments

We thank the Berkeley graphics group for helpful criticism and comments, and Adam Bargteil for comments and help with the SLC code. This work was supported in part by California MICRO 03-067 and 04-066, by the National Science Foundation under Awards CCF-0430065 and CCF-0635381, by generous support from Apple Computer, Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, the Hellman Family Fund, and by two Alfred P. Sloan Research Fellowships.

References

[BE92] BERN M., EPPSTEIN D.: Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, Du D.-Z., Hwang F., (Eds.), vol. 1 of *Lecture Notes Series on Computing*. World Scientific, Singapore, 1992, pp. 23–90.

[BGOS06] BARGTEIL A. W., GOKTEKIN T. G., O'BRIEN J. F., STRAIN J. A.: A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics 25*, 1 (Jan. 2006), 19–38.

[Blu67] BLUM H.: A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form* (1967), 362–380.

[CD97] CANI M.-P., DESBRUN M.: Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics 3*, 1 (Jan. 1997), 39–50.

[CGFO06] CHENTANEZ N., GOKTEKIN T. G., FELDMAN B. E., O'BRIEN J. F.: Simultaneous coupling of fluids and deformable

bodies. In *2006 Symposium on Computer Animation* (Sept. 2006), pp. 83–89.

[CMT04] CARLSON M., MUCHA P. J., TURK G.: Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 377–384. Special issue on Proceedings of SIGGRAPH 2004.

[DC96] DESBRUN M., CANI M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96* (Aug. 1996), pp. 61–76.

[EMF02] ENRIGHT D. P., MARSCHNER S. R., FEDKIW R. P.: Animation and rendering of complex water surfaces. *ACM Transactions on Graphics 21*, 3 (July 2002), 736–744. Special issue on Proceedings of SIGGRAPH 2002.

[ETK*07] ELCOTT S., TONG Y., KANSO E., SCHRÖDER P., DESBRUN M.: Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics 26*, 1 (Jan. 2007).

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Computer Graphics (SIGGRAPH 2001 Proceedings)* (Aug. 2001), pp. 23–30.

[FLM03] FOSKEY M., LIN M., MANOCHA D.: Efficient computation of a simplified medial axis. In *8th Symposium on Solid Modeling and Applications* (June 2003), pp. 96–107.

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. In *Graphics Interface 1996* (May 1996), pp. 204–212.

[FOA03] FELDMAN B. E., O'BRIEN J. F., ARIKAN O.: Animating suspended particle explosions. *ACM Transactions on Graphics 22*, 3 (July 2003), 708–715. Special issue on Proceedings of SIGGRAPH 2003.

[FOK05] FELDMAN B. E., O'BRIEN J. F., KLINGNER B. M.: Animating gases with hybrid meshes. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 904–909. Special issue on Proceedings of SIGGRAPH 2005.

[FOKG05] FELDMAN B. E., O'BRIEN J. F., KLINGNER B. M., GOKTEKIN T. G.: Fluids in deforming meshes. In *2005 Symposium on Computer Animation* (July 2005), pp. 255–259.

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *Computer Graphics (SIGGRAPH 2001 Proceedings)* (Aug. 2001), pp. 15–22.

[GBO04] GOKTEKIN T. G., BARGTEIL A. W., O'BRIEN J. F.: A method for animating viscoelastic fluids. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 463–468. Special issue on Proceedings of SIGGRAPH 2004.

[GSLF05] GUENDELMAN E., SELLE A., LOSASSO F., FEDKIW R.: Coupling water and smoke to thin deformable and rigid shells. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 973–981. Special issue on Proceedings of SIGGRAPH 2005.

[KCC*06] KIM J., CHA D., CHANG B., KOO B., IHM I.: Practical animation of turbulent splashing water. In *2006 Symposium on Computer Animation* (Sept. 2006), pp. 335–344.

[KFCO06] KLINGNER B. M., FELDMAN B. E., CHENTANEZ N., O'BRIEN J. F.: Fluid animation with dynamic meshes. *ACM Transactions on Graphics 25*, 3 (Aug. 2006), 820–825. Special issue on Proceedings of SIGGRAPH 2006.

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 457–462. Special issue on Proceedings of SIGGRAPH 2004.

[LS07] LABELLE F., SHEWCHUK J. R.: Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. *ACM Transactions on Graphics 26*, 3 (Aug. 2007). Special issue on Proceedings of SIGGRAPH 2007.

[LSSF06] LOSASSO F., SHINAR T., SELLE A., FEDKIW R.: Multiple interacting liquids. *ACM Transactions on Graphics 25*, 3 (Aug. 2006), 812–819. Special issue on Proceedings of SIGGRAPH 2006.

[McC87] MCCORMICK S. F.: *Multigrid Methods*. Society for Industrial and Applied Mathematics, Philidelphia, 1987.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *2003 Symposium on Computer Animation* (Aug. 2003), pp. 154–159.

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. In *2004 Symposium on Computer Animation* (July 2004), pp. 141–151.

[NFJ02] NGUYEN D., FEDKIW R., JENSEN H.: Physically based modeling and animation of fire. *ACM Transactions on Graphics 21*, 3 (Aug. 2002), 721–728. Special issue on Proceedings of SIGGRAPH 2002.

[Owe98] OWEN S. J.: A survey of unstructured mesh generation technology. In *7th International Meshing Roundtable* (July 1998), pp. 239–267.

[PTB*03] PREMOŽE S., TASDIZEN T., BIGLER J., LEFOHN A., WHITAKER R.: Particle-based simulation of fluids. *Computer Graphics Forum 22*, 3 (Sept. 2003), 401–410.

[SRF05] SELLE A., RASMUSSEN N., FEDKIW R.: A vortex particle method for smoke, water, and explosions. *ACM Transactions on Graphics 24*, 3 (July 2005), 910–914. Special issue on Proceedings of SIGGRAPH 2005.

[Sta99] STAM J.: Stable fluids. In *Computer Graphics (SIGGRAPH '99 Proceedings)* (Aug. 1999), pp. 121–128.

[Str99] STRAIN J. A.: Semi-Lagrangian methods for level set equations. *Journal of Computational Physics 151*, 2 (May 1999), 498–533.

[Str01] STRAIN J. A.: A fast semi-Lagrangian contouring method for moving interfaces. *Journal of Computational Physics 169*, 1 (May 2001), 1–22.

[TKPR06] THÜREY N., KEISER R., PAULY M., RÜDE U.: Detail-preserving fluid control. In *2006 Symposium on Computer Animation* (Sept. 2006), pp. 7–12.

[TLP06] TREUILLE A., LEWIS A., POPOVIĆ Z.: Model reduction for real-time fluids. *ACM Transactions on Graphics 25*, 3 (Aug. 2006), 826–834. Special issue on Proceedings of SIGGRAPH 2006.

[TOS01] TROTTENBERG U., OOSTERLEE C. W., SCHÜLLER A.: *Multigrid*. Academic Press, London, 2001.

[TPF89] TERZOPOULOS D., PLATT J., FLEISCHER K.: Heating and melting deformable models (from goop to glop). In *Graphics Interface 1989* (June 1989), pp. 219–226.

[WSHD04] WARREN J., SCHAEFER S., HIRANI A. N., DESBRUN M.: Barycentric coordinates for convex sets. Advances in Computational and Applied Mathematics, 2004.

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 965–972. Special issue on Proceedings of SIGGRAPH 2005.

[ZYP06] ZHENG W., YONG J.-H., PAUL J.-C.: Simulation of bubbles. In *2006 Symposium on Computer Animation* (Sept. 2006), pp. 325–333.