A Semi-Lagrangian Contouring Method for Fluid Simulation

ADAM W. BARGTEIL, TOLGA G. GOKTEKIN, JAMES F. O'BRIEN, and JOHN A. STRAIN
University of California, Berkeley

In this article, we present a semi-Lagrangian surface tracking method for use with fluid simulations. Our method maintains an explicit polygonal mesh that defines the surface, and an octree data structure that provides both a spatial index for the mesh and a means for efficiently approximating the signed distance to the surface. At each timestep, a new surface is constructed by extracting the zero set of an advected signed-distance function. Semi-Lagrangian backward path tracing is used to advect the signed-distance function. One of the primary advantages of this formulation is that it enables tracking of surface characteristics, such as color or texture coordinates, at negligible additional cost. We include several examples demonstrating that the method can be effectively used as part of a fluid simulation to animate complex and interesting fluid behaviors.

Categories and Subject Descriptors: 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*

General Terms: Algorithms

Additional Key Words and Phrases: Natural phenomena, physically based animation, computational fluid dynamics, surface tracking, level-set methods, semi-Lagrangian contouring

1. INTRODUCTION

The fundamental problem of tracking a surface as it is advected by some velocity field arises frequently in applications such as surface reconstruction, image segmentation, and fluid simulation. Unfortunately, the naïve approach of simply advecting the vertices of a polygonal mesh, or other explicit representation of the surface, quickly encounters problems such as tangling and self-intersection. Instead, a family of methods, known as level-set methods, has been developed for surface tracking. These methods represent the surface implicitly as the zero set of a scalar field defined over the problem domain. The methods are widely used, and the texts by Sethian [1999] and Osher and Fedkiw [2003], and Osher and Sethian's [1988] seminal article, provide an excellent introduction to the topic. One of the key issues that distinguishes various level-set and similar approaches is the representation of the scalar field, which must capture whatever surface properties are important to a given application.

In this article we present a surface tracking method that explicitly represents the surface as a set of polygons. However, rather than attempting to advect these polygons forward with the flow, we update the surface in time with an implicit representation: an advected signed-distance function, ψ, whose

This work was supported in part by California MICRO 04-066 and 05-044, and by generous support from Apple Computer, Alias, Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, and the Alfred P. Sloan Foundation. Adam Bargteil was supported by a Siebel Scholarship.

Author's address: Computer Science Department, University of California, Berkeley, 537 Soda Hall, Berkeley, CA 94720; email: {adamb,goktekin,job}@cs.berkeley.edu, strain@math.berkeley.edu.

zero set defines the surface. A new polygonal surface is generated by *contouring* or extracting the zero set of ψ. The value of ψ at a point $\boldsymbol{x}$, at current time t, is obtained by first tracing backward through the flow field to find the previous location $\boldsymbol{x}'$ at time $t - \Delta t$, and then returning the signed distance of $\boldsymbol{x}'$ from the previous surface. Using adaptive octree data structures, we can efficiently and reliably construct the new surface and corresponding signed-distance function.

The theoretical framework for this method comes from a series of articles by Strain [1999b, 1999c, 1999a, 2000, 2001] that described and analyzed a method for contour tracking in two dimensions. While the semi-Lagrangian procedure for backward advection does not change significantly when going from two- to three-dimensional problems, significant surface tracking issues arise when moving to three dimensions. This article discusses these issues, as well as the general method, and demonstrates how semi-Lagrangian surface contouring can be useful for animating the complex and interesting behavior of fluids.

One of the primary advantages of this method is that it enables tracking surface characteristics, such as color or texture coordinates. These properties can be easily stored directly on the polygonal mesh and efficiently mapped onto the new surface during semi-Lagrangian advection. The explicit surface representation also facilitates other common operations, such as rendering, while reconstruction from a scalar function allows operations that rely on an implicit representation. Finally, the method produces detailed, well-defined surfaces that are suitable for realistic animation and that do not jitter or exhibit other undesirable behaviors.

2. BACKGROUND

Our method pulls together solutions to a number of well-studied problems to arrive at a method for tracking surfaces. In this section we will first discuss other surface tracking methods and then discuss related work and the mathematical foundation for several of the individual components of our method.

2.1 Previous Surface Tracking Methods

Because surface tracking arises in a variety of contexts, the topic has received a significant amount of attention. Even in the limited context of fluid animation, there has been a great deal of excellent work on simulating fluids with free surfaces, including Foster and Metaxas [1996], Foster and Fedkiw [2001], Enright et al. [2002b], Carlson et al. [2002, 2004], Losasso et al. [2004], Goktekin et al. [2004], Hong and Kim [2005], Wang et al. [2005], Guendelman et al. [2005], and Zhu and Bridson [2005]. The methods available for tracking free surfaces of liquids can be roughly sorted into four categories: level-set methods, particle-based methods, particle level-set methods, and semi-Lagrangian contouring.

2.1.1 *Level-Set Methods.* Many of the most successful solutions to the surface tracking problem are based on level-set methods, which were originally introduced by Osher and Sethian [1988]. A complete review of level-set methods is beyond the scope of this article, and we recommend the excellent surveys by Sethian [1999] and Osher and Fedkiw [2003]. Level-set methods represent a surface as the zero set of a scalar function which is updated over time by solving a partial differential equation, known as the *level-set equation*. This equation relates change of the scalar function to an underlying velocity field. By using this implicit representation, level-set methods avoid dealing with complex topological changes. However, the scalar function is defined and maintained in the embedding three-dimensional space, rather than just on the two-dimensional surface. In practice, scalar function values need only be accurately maintained very near the surface, resulting in a cost that is roughly linear in the complexity of the surface. One difficulty with level-set methods is that they generally require very high-order conservation-law solvers, though fast semi-Lagrangian methods have been shown to work in some cases [Strain 1999b; Enright et al. 2005]. The most significant drawback to using level-set methods to track liquid surfaces is their tendency to lose volume in underresolved, high-curvature regions. See Enright et al. [2002a] for an excellent discussion of the reasons for this volume loss.

Bærentzen and Christensen [2002] built a sculpting system using a level-set surface representation which could be manipulated by a user with a variety of sculpting tools. Like us, they used adaptive grid structures to store the scalar field. However, they used a two-level structure rather than a full octree. They also used semi-Lagrangian methods to update their level-set function. However, when evaluating the distance function after the semi-Lagrangian path tracing, they interpolated distance values stored on a regular grid, while our explicit surface representation allows us to compute exact distances near the surface.

Sussman and Puckett [2000] coupled volume-of-fluid and level-set methods to model droplet dynamics in ink-jet devices. Volume-of-fluid [Hirt and Nichols 1981] techniques represent the surface by storing, in each voxel, a *volume fraction*—the proportion of the voxel filled with liquid. Any cell whose fraction is not one or zero contains surface. Unfortunately, this representation does not admit accurate curvature estimates, which are essential to surface tension computations. However, accurate curvature estimates are easily computed from level-set representations. Thus, the authors combined volume-of-fluid and level-set representations to model surface tension in ink droplets. Some volume-of-fluid methods build an explicit surface representation from the volume fractions stored in each voxel. The key difference between our method and volume-of-fluid methods is that we never compute volume fractions. Instead, our explicit representation is generated by contouring an advected signed-distance function.

2.1.2 *Particle-Based Methods.* A number of researchers [Terzopoulos et al. 1989; Desbrun and Gascuel 1995; Foster and Metaxas 1996; Desbrun and Cani 1996; Cani and Desbrun 1997; Stora et al. 1999; Müller et al. 2003, 2004; Premože et al. 2003; Zhu and Bridson 2005; Pauly et al. 2005] have used particles to track surfaces. In many of these methods, the simulation elements are particles, which are already being tracked throughout the volume of the deforming liquid or solid. The surface can then be implicitly defined as the boundary between where the particles are and where they aren't. The particles can be visualized directly, or can be used to define an implicit representation using blobbies or moving least-squares methods. Premože et al. [2003] went a step further and used particle positions and velocities to guide a level-set solution. Mueller et al. [2004] and Pauly et al. [2005] used special particles, called *surfels*, to represent the surface. Surfels store a surface normal as well as position and there are generally many more surfels than simulation particles. The principal drawback of these methods is that generating high-quality time-coherent surfaces can be difficult: directly visualizing the particles is insufficient for high-quality animations, methods which convert the particles to some other representation on a per-frame basis often lack temporal coherence, and methods which must run sequentially through the frames or run during the simulation are often quite costly. Additional difficulties arise when trying to ensure a good sampling of the surface.

2.1.3 *Particle Level-Set Methods.* To address the volume loss of level-set methods, Enright and his colleagues [2002a, 2002b, 2005] built on the work of Foster and Fedkiw [2001] to develop particle level-set methods. These methods track the characteristics of the fluid flow with Lagrangian particles, which are then used to fix the level-set solution, essentially increasing the effective resolution of the method. Recently, these methods have been extended to work with octrees [Enright et al. 2005; Losasso et al. 2004], allowing for very high-resolution surface tracking. These methods represent the current state of the art on tracking liquid surfaces for animation, but do have some drawbacks. In particular, the published particle correction rules choose a single particle to provide the signed-distance value. Since there is no guarantee that the same particle will be chosen at subsequent timesteps, the method is extremely susceptible to high-frequency temporally incoherent perturbations of the surface. The artifacts are most noticeable when the surface thins out below the grid resolution and particles happen to be near some of the sample points, but not others. Also, the method has a large number of parameters and rules, such as the number of particles per cell and the reseeding strategy, which need to be decided, often in an application-specific way. Finally, the method tends to produce very smooth surfaces with very little detail, which is desirable in some, but not all, applications. Despite these drawbacks, the particle level-set methods have been very successful and represent a significant step forward in the area of surface tracking for liquid simulations.

2.1.4 *Semi-Lagrangian Contouring.* Recently, Strain [1999b, 1999c, 1999a, 2000, 2001] has written a series of articles building a theoretical framework culminating in the formulation of surface tracking as a contouring problem. He demonstrated his semi-Lagrangian contouring method on a variety of two-dimensional examples. Our method is based on the method presented by Strain [2001], but with variations and extensions to deal with problems that arise in three-dimensional computer animation. While our method bears a number of similarities to level-set methods and takes advantage of many techniques developed for those methods, we are not directly solving the level-set equation. By formulating surface tracking as a contouring problem, we avoid many of the issues that complicate level-set methods. In particular, we do not have the same volume loss issues which prompted the particle level-set methods: while we do not explicitly conserve volume, our semi-Lagrangian path tracing tends to conserve volume in the same way as the Lagrangian particles in the particle level-set method.

2.2 Implicit Representations

The octree structure we use to build and index the polygonal mesh is quite similar to adaptively sampled distance fields [Frisken et al. 2000]. These structures adaptively sample distance fields according to local detail and store samples in a spatial hierarchy. The key difference between adaptively sampled distance fields and our surface representation is that we store a polygon mesh in addition to distance samples. This polygon mesh is used for exact evaluation of the distance function near the surface. Additionally, our splitting criterion is different from that presented by Frisken et al. [2000].

An alternative structure for storing narrow-band level-set functions is the dynamic tubular grid of Nielsen and Museth [2006]. This structure can be combined with run-length encoding schemes [Houston et al. 2006], providing extremely compact, high-resolution representations of level-set functions. While the asymptotic times for their structure match ours, they are able to exploit cache coherence to provide extremely fast run times for most level-set operations. Integrating the methods presented here with this data structure is a promising area for future work.

2.3 Contouring

As our title suggests, we formulate surface tracking as a contouring problem. The contouring problem has been well studied in computer graphics and a number of approaches have been suggested. The oldest and most widely used is marching cubes, which was first presented by Wyvill et al. [1986], and later named and popularized by Lorensen and Cline [1987]. Marching cubes suffers from a tendency to create ill-shaped triangles. This problem is fixed to some degree by dual contouring [Ju et al. 2002], which also provides adaptive contouring and an elegant means of preserving sharp boundaries. Dual contouring depends on normal estimates at edge crossings and is very sensitive to inaccuracies in these normal estimates. Unfortunately, in our method we do not have accurate normal information until after the contouring step, when we have the triangle mesh. More recently, Boissonnat and Oudot [2003] presented a contouring technique which uses Delaunay triangulation methods to generate provably good triangulations. However, this method appears to be prohibitively expensive for something which must run at every timestep. Yet another alternative is marching triangles [Hilton et al. 1996], which takes a surface-based rather than volume-based approach to contouring. Marching triangles requires significantly less computation time and fewer triangles, and produces higher-quality triangles than marching cubes. Unfortunately, marching triangles is not guaranteed to produce closed, manifold meshes in the presence of sharp or thin features.

2.4 Semi-Lagrangian Methods

Semi-Lagrangian methods have been widely used in computer graphics since they were introduced by Stam [1999] to solve the nonlinear advection term of the Navier-Stokes equations. These methods provide the foundation for our surface tracking method. Consequently, we briefly discuss the mathematical foundation of semi-Lagrangian methods. Our discussion follows that of Strain [1999b].

Consider the simplest linear hyperbolic PDE

$$\phi_t + \boldsymbol{v}(\boldsymbol{x}, t) \cdot \nabla\phi = 0, \tag{1}$$

where ϕ is a scalar field and $\boldsymbol{v}(\boldsymbol{x}, t)$ is a velocity function. Equation (1) passively advects ϕ through the velocity field $\boldsymbol{v}$. Semi-Lagrangian methods are based on the observation that Equation (1) propagates ϕ values along characteristic curves $\boldsymbol{x} = \boldsymbol{s}(t)$ defined by

$$\dot{\boldsymbol{s}}(t) = \boldsymbol{v}(\boldsymbol{s}(t), t), \qquad \boldsymbol{s}(0) = \boldsymbol{x_0}. \tag{2}$$

Thus we can find ϕ values at any time t by finding the characteristic curve passing through $(\boldsymbol{x}, t)$, following it backward to some previous point $(\boldsymbol{x_0}, t_0)$ where the value of ϕ is known, and setting $\phi(\boldsymbol{x}, t) = \phi(\boldsymbol{x_0}, t_0)$. This observation forms the basis of the *backward characteristic* or *CIR* scheme developed by Courant, Isaacson, and Rees [1952], which is the simplest semi-Lagrangian scheme. Given ϕ at time t_n, CIR approximates $\phi(\boldsymbol{x}, t_{n+1})$ at any point $\boldsymbol{x}$ at time $t_{n+1} = t_n + \Delta t$ by evaluating the previous speed $\boldsymbol{v}(\boldsymbol{x}, t_n)$, approximating the backward characteristic through $\boldsymbol{x}$ by a straight line

$$\boldsymbol{s}(t) \approx \boldsymbol{x} - (t_{n+1} - t)\boldsymbol{v}(\boldsymbol{x}, t_n), \tag{3}$$

and interpolating ϕ at time t_n to the point

$$\boldsymbol{s}(t_n) \approx \boldsymbol{x} - (\Delta t)\boldsymbol{v}(\boldsymbol{x}, t_n). \tag{4}$$

Then $\phi(\boldsymbol{x}, t_{n+1})$ is set equal to the interpolated value, $\phi(\boldsymbol{s}(t_n), t_n)$.

For linear PDEs, such as Equation (1), the Lax-Richtmyer equivalence theorem [LeVeque 1990] guarantees that CIR will converge to the exact solution as $\Delta t, \Delta x \to 0$ if it is stable and consistent.

The stability properties of the CIR scheme are excellent. Each new value $\phi(\boldsymbol{x}, t_{n+1})$ is a single interpolated value of ϕ at time t_n, so unconditional stability is guaranteed in any norm where the interpolation does not increase norms. For example, CIR with linear interpolation is unconditionally stable in the 2-norm. In general, semi-Lagrangian schemes satisfy the CFL condition by shifting the stencil, rather than restricting the timestep. Thus information propagates over long distances in one timestep.

Consistency (loosely speaking, the local accuracy of the method), however, is conditional. The global error of CIR is

$$O\left(\frac{(\Delta x)^2}{\Delta t}\right) + O(\Delta t), \tag{5}$$

due to the $O((\Delta x)^2)$ error in linear interpolation accumulated over $O(1/(\Delta t))$ timesteps, plus the $O(\Delta t)$ error due to freezing F and approximating the characteristics by straight lines. Thus CIR is consistent to $O(\Delta t)$ if a condition $\Delta t \geq O(\Delta x)$ is satisfied, contrary to the usual hyperbolic condition $\Delta t \leq C\Delta x$. This condition is extremely convenient, because $\Delta t = O(\Delta x)$ balances time and space resolution in this first-order accurate scheme.

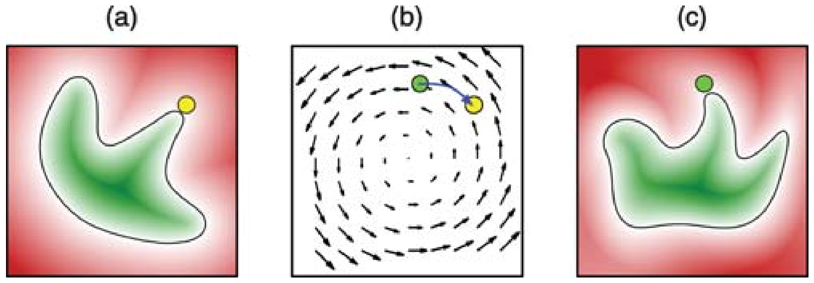

Fig. 1. An overview of our method. At each timestep we begin with an explicit surface representation, from which we can build a signed-distance function (a) and a velocity field given by the fluid simulator (b). We then define a field function, the zero set of which will be our new surface. To get the value of the field function at the green point (c), we trace backward through the flow field to find the yellow point (b), which is the image of the green point at the previous timestep. We then evaluate the signed distance of the yellow point to the previous surface (a). We can evaluate this field function at every point in the domain and extract the zero set (c).

For nonlinear PDEs, CIR still converges when the solution is smooth. But nonsmooth shock solutions of conservation laws move at the wrong speed because CIR is not in conservative form. Since level-set solutions have no shocks, CIR is a natural scheme for moving interfaces.

3. METHOD OVERVIEW

The surface tracking problem can be phrased as: given a surface representation and a velocity field at time t, build a representation of the surface at time $t + \Delta t$. We begin with a triangle mesh and an octree annotated with signed-distance field samples. We could try to advect the mesh points through the flow field, but would quickly encounter significant topological difficulties. Instead, we avoid topological issues by updating the surface using an implicit representation. The implicit representation is then used to construct a new mesh at the current timestep. More specifically, we define a scalar-valued function which relates the surface at the current timestep to the surface at the previous timestep. Next, we extract the zero set of this function using a contouring algorithm. Finally, a new signed-distance field is computed through a process known as *redistancing* (see Figure 1).

4. EXPLICIT REPRESENTATION

One of the key differences between our method and other surface tracking methods is that we build an explicit representation of the surface at every timestep. This explicit representation is a closed, manifold triangle mesh, which is stored as an array of vertices and an array of triangles. The vertices are shared between triangles, allowing for easy computation of smooth vertex normals and other common mesh operations. The distance tree (see Section 6) provides a spatial index for the mesh. The explicit representation provides our method with several advantages. First, it allows us to compute *exact* signed-distance values near the mesh. Second, it allows us to store properties on mesh vertices, rather than at points near the mesh. Finally, it allows us to take advantage of the many tools and algorithms which have been developed in computer graphics for manipulating and rendering triangle meshes.

5. IMPLICIT REPRESENTATION

To avoid the topological difficulties of directly updating an explicit surface representation, we update the surface in time through an implicit representation (see Figure 2). We define a scalar-valued field function, $\psi(\boldsymbol{x})$, which relates the surface at the current timestep to the surface at the previous timestep.

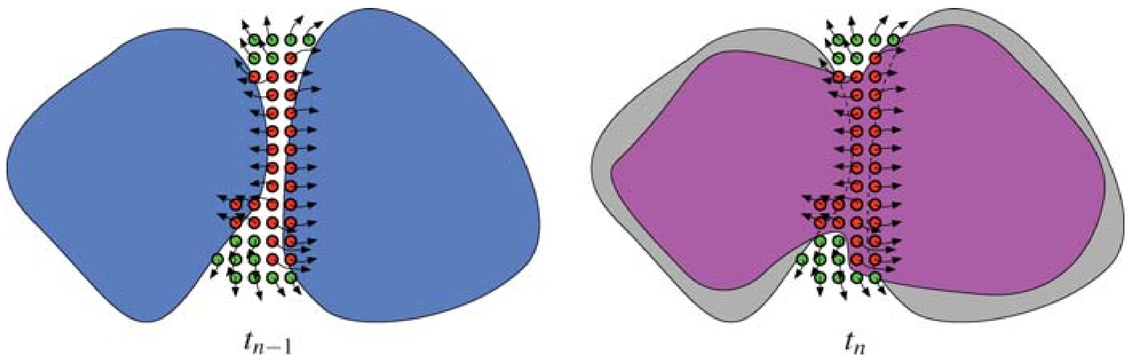

Fig. 2. An example of how our implicit representation accommodates merging surfaces. The red grid points trace back through the velocity field to points inside the surface. The green grid points trace back to points outside the surface. When the contouring algorithm runs, it will look for zero crossings only between positive and negative (green and red) grid points and create a surface that does not pass between two grid points of the same color. Thus, without even explicitly determining that a topological change has occurred, the method handles the change.

The surface at the current timestep will be the zero set of this function,

$$S_n = \{\boldsymbol{x} : \psi(\boldsymbol{x}) = 0\}. \tag{6}$$

For a point $\boldsymbol{x}$ at the current timestep, the function, ψ, first uses backward path tracing, a semi-Lagrangian integration technique, to find the point $\boldsymbol{x}'$ at the previous timestep which flows to $\boldsymbol{x}$. It then returns the distance from $\boldsymbol{x}'$ to the surface, S_{n-1}, at the previous timestep. If we denote the backward path tracing as $\boldsymbol{b}(\boldsymbol{x}) : \mathcal{R}^3 \to \mathcal{R}^3$ and let $\phi_n(\boldsymbol{x})$ be the signed distance from $\boldsymbol{x}$ to the surface S_n,

$$\psi_n(\boldsymbol{x}) = \phi_{n-1}(\boldsymbol{b}(\boldsymbol{x})) = \phi_{n-1}(\boldsymbol{x}'). \tag{7}$$

Essentially, we are advecting the signed-distance function through the velocity field given by the fluid simulator. In solving this advection term, our method differs from the simple CIR scheme discussed earlier in two ways. First, instead of the simple linear backward path tracing, we use a second-order Runge-Kutta scheme (also known as the midpoint method with an Euler predictor)

$$\boldsymbol{x_{n-1/2}} = \boldsymbol{x}(t_{n-1/2}) = \boldsymbol{x_n} - \frac{\Delta t}{2}\boldsymbol{v}(\boldsymbol{x_n}, t_n), \tag{8}$$

$$\boldsymbol{x_{n-1}} = \boldsymbol{x}(t_{n-1}) = \boldsymbol{x_n} - (\Delta t)\boldsymbol{v}(\boldsymbol{x_{n-1/2}}, t_n), \tag{9}$$

where $\boldsymbol{v}(\boldsymbol{x}, t)$ is the velocity function. It is important to note that, while this method traces back through the velocity field with second-order accuracy, the velocity field is frozen over the course of the timestep, leading to first-order accuracy in time. The second difference is that, when evaluating ϕ at points near the surface, we do *not* interpolate values stored on a grid. Instead, we compute *exact* distance values. These changes only improve the accuracy (consistency) of our method and do not affect the unconditional stability.

To compute the exact distance from a point $\boldsymbol{x}'$, we compute the distances d_i to all the nearby triangles. The distance to the surface is $\min_i d_i$. Schneider and Eberly [2002] detailed a method for computing the distance from a point to a triangle. This operation is relatively expensive, but many triangles can be pruned, especially when $\boldsymbol{x}'$ is very close to the surface, by using standard bounding-box techniques and our octree data structure (see Section 6).

Signing the distance values turns out to be somewhat difficult near sharp corners. Let $\boldsymbol{y}$ and $\boldsymbol{n}(\boldsymbol{y})$ denote the closest point on the surface to $\boldsymbol{x}'$ and its normal, respectively. When $\boldsymbol{y}$ lies strictly inside a

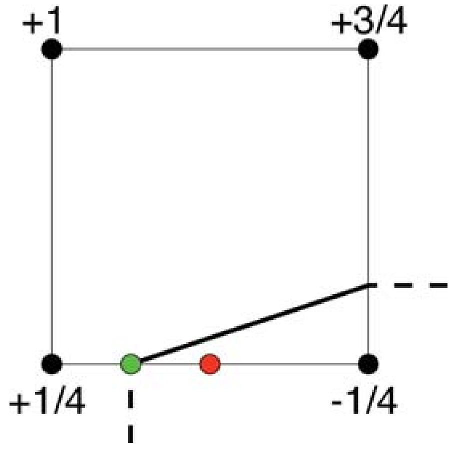

Fig. 3. The figure shows a part of the surface passing through a grid cell. The cell's vertices have been annotated with signed-distance values. Linear interpolation of these values incorrectly chooses the red point as the zero crossing along the bottom edge. The green point is the actual zero crossing, which will be found with our exact evaluation.

triangle then the sign can be easily computed as

$$s = (\boldsymbol{x}' - \boldsymbol{y}) \cdot \boldsymbol{n}(\boldsymbol{y}), \tag{10}$$

where $\boldsymbol{n}(\boldsymbol{y})$ is the normal of the triangle containing $\boldsymbol{y}$. However, if the nearest point in the mesh lies on more than one triangle (i.e., on an edge or vertex of the mesh), the triangles do not always agree on the sign. These situations can be resolved by computing an angle-weighted pseudonormal for each edge and vertex of the mesh and using these pseudonormals to determine the sign when the nearest point is on an edge or vertex of the mesh. Bærentzen and Aanæs [2002] provided a proof that this procedure results in accurate signing (in exact arithmetic).

The ability to compute exact distances is one of the chief advantages of having an explicit surface representation. Interpolation can produce substantial errors (see Figure 3) which are compounded over time. In fact, this interpolation error is one of the most significant drawbacks to semi-Lagrangian methods in general. When used for velocity advection, interpolation produces such significant smoothing that researchers have proposed a number of methods to add detail back to the flow [Fedkiw et al. 2001] or avoid semi-Lagrangian advection altogether [Zhu and Bridson 2005]. In this work, we are able to leverage the advantages of semi-Lagrangian advection, without incurring the interpolation error that would otherwise undesireably smooth surface detail.

6. THE DISTANCE TREE

Our implementation makes heavy use of a structure we call the *distance tree*. The distance tree is a balanced octree subdivision of the spatial domain. The octree vertices are annotated with signed-distance values and each cell of the octree contains a list of the triangles with which it intersects. The distance tree serves three purposes:

(1) It provides a fast spatial index for the mesh so that nearby triangles can be found quickly.
(2) It provides a fast, approximate signed-distance function, which is sufficient when evaluating the signed distance far from the surface.
(3) It guides the contouring algorithm, quickly identifying cells which have vertices of different sign and, thus, contain triangles.

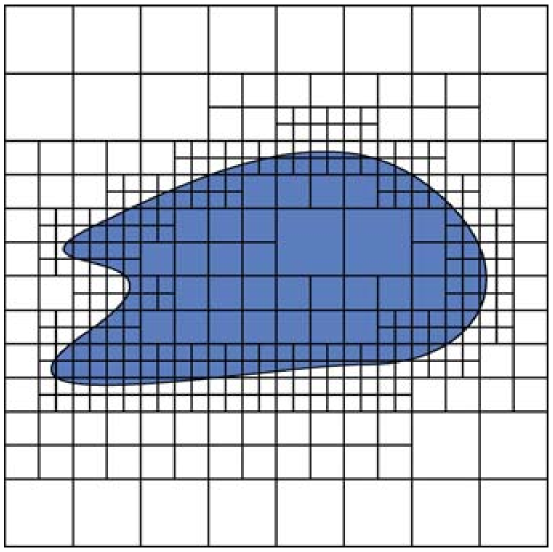

Fig. 4. A two-dimensional distance tree. Distance samples are stored at the octree vertices and triangle lists are stored in cells which intersect the surface. This distance tree could be generated using our implementation of Criterion (13), which considers ψ only at cell centers.

6.1 Approximating the Signed-Distance Function

When computing the signed distance from a point $\boldsymbol{x}'$ to a surface, S, we first find the smallest octree cell, $\boldsymbol{C}$, containing $\boldsymbol{x}'$. If $\boldsymbol{C}$ is at the finest level of the octree, then $\boldsymbol{x}'$ may be near the surface and all the triangles in the up to 27 cells in the concentric triple[1] of $\boldsymbol{C}$ are considered when computing the minimum distance to the surface. By storing the nearest distance seen so far and using standard bounding-box techniques, many of these triangles can be pruned before computing distances, especially when $\boldsymbol{x}'$ is very near the surface. If the computed distance is less than $\boldsymbol{C}$'s edge length, then the distance is guaranteed to be exact. Otherwise, the computed distance is a very good estimate but may be slightly larger than the actual distance. Contrariwise, if $\boldsymbol{C}$ is not at the finest level of the octree or if there are no triangles in the concentric triple of $\boldsymbol{C}$, then $\boldsymbol{x}'$ is not near the surface and we do not require an exact distance. An approximation with the correct sign is sufficient. In this case, we use trilinear interpolation of the distance values stored at the vertices of $\boldsymbol{C}$.

6.2 General Splitting Criterion

We make use of two different methods for building distance trees in this work. Most often, we wish to build a distance tree to resolve the zero set of our field function ψ. However, it is also useful to build a distance tree from an existing triangle mesh. Our octrees are always built in a top-down manner where each cell is split based on some variation of the following splitting criterion:

Split any cell whose edge length exceeds its minimum distance to the surface. (11)

Splitting ends when the tree reaches a predetermined maximum depth. Criterion (11) results in a three-color octree, as described by Samet [1990], where each cell of the octree has one of three types: interior, exterior, and boundary (see Figure 4).

[1] If cell $\boldsymbol{C} = \{\boldsymbol{x} : \|\boldsymbol{x} - \boldsymbol{c}\|_\infty \leq r\}$ has center $\boldsymbol{c}$ and edge length $2r$ then its concentric triple T is given by $T = \{x : \|\boldsymbol{x} - \boldsymbol{c}\|_\infty \leq 3r\}$.

In general, Criterion (11) builds octrees with several useful properties:

—Adjacent cells differ in size by no more than a factor of 2, producing a smooth mesh and simplifying procedures such as neighbor finding and triangulation of the vertices.

—A cell's size is proportional to its distance to the surface.

—If ϕ is the signed distance to the surface at vertices and we extend ϕ into each cell by trilinear interpolation, then, because cells vary in size, ϕ will be discontinuous. However, the jumps in ϕ decrease in size in cells near the surface because of the triangle inequality. Thus the interpolated ϕ is nearly continuous near the surface.

—Cells coarsen very rapidly away from the surface: if there are N childless cells touching the surface, then the entire tree contains only $O(N \log N)$ cells. Hence the surface is resolved accurately at minimal cost.

6.3 Building a Distance Tree to Resolve ψ

When building a new octree at the beginning of each timestep, we are essentially trying to resolve our approximation

$$\psi_{n+1}(\boldsymbol{x}) = \phi_n(\boldsymbol{x} - (\Delta t)\boldsymbol{v}(\boldsymbol{x}_{\boldsymbol{n-1/2}}, t_n)) \tag{12}$$

to the signed-distance function $\phi_{n+1}(\boldsymbol{x})$. The octree is built recursively from the root cell C_0 by the following splitting criterion:

$$\textit{Split every cell where } |\psi_{n+1}| \textit{ is larger than the edge length.} \tag{13}$$

Thus we apply Criterion (13) as if ψ_{n+1} were a distance function. Redistancing every timestep keeps

$$\psi_{n+1} = \phi_n + (\Delta t)\boldsymbol{v} \cdot \nabla\phi_n + O(\Delta t) = \phi_n + O(\Delta t) \tag{14}$$

within $O(\Delta t)$ of the signed-distance function ϕ_n. Thus in the limit, $\Delta t = O(\Delta x) \to 0$, Criterion (13) reduces to (11), yielding the properties noted above. In practice, we use the value of ϕ at the cell's center to determine whether we should split the cell. To deal with the fact that ψ_{n+1} is not a distance function and that the value at the cell's center may not be the minimum over the cell, we multiply the edge length by some constant before doing the comparison. We have found that 1/3 works well in practice—always dividing near the surface, without spuriously dividing too many cells. Notice that we can vary this constant to achieve high-resolution bands of varying width around the surface.

6.4 Building a Distance Tree from a Triangle Mesh

When building an octree from a triangle mesh (either in initialization, or after some geometric operation has been applied to the triangle mesh) we use the following splitting criterion:

$$\textit{Split every cell whose concentric triple intersects the surface.} \tag{15}$$

This test is efficiently implemented using Green and Hatch's [1995] cube/triangle intersection test. Notice that we need not check every cell in the concentric triple of C individually, but can just increase the size of C. In practice we have found it sufficient to increase the cell size by a factor of 2, rather than 3, but such trees may not satisfy all the properties listed above.

7. CONTOURING

Once we have resolved ψ on our distance tree, we need to create an explicit representation of our surface at the new timestep. Creating this explicit representation amounts to extracting the zero set of ψ and is an instance of the contouring problem, which has been well studied in computer graphics. For its simplicity, robustness, and speed, we choose to use a marching-cubes method in our implementation. Our implementation is based on Bloomenthal's [1994]. Our cubes are the leaf cells in the distance tree which have vertices of differing sign. We divide each cube into six tetrahedra to simplify the implementation. Additionally, when finding the zero crossing along any edge (which will eventually be a vertex in the triangle mesh), we use a secant method to speed up convergence and evaluate our full composite field function, including exact evaluation of the previous signed-distance function. Consequently, the vertices of our polygon mesh are guaranteed to lie on the implicit surface (within an ϵ tolerance). In fact, each vertex in our polygon mesh can be mapped to some point on some triangle in the mesh at the previous timestep. We take advantage of this fact when advecting surface properties. The marching-cubes algorithm works well for our purposes because each triangle generated by marching cubes sits strictly inside a single cell of the distance tree, making the distance tree an especially effective spatial index. Furthermore, we use the distance tree we have already built to guide the marching cubes, avoiding the need to build a second structure to determine the topology of the new mesh. Near the surface, our distance tree is refined to the maximum level and looks like a uniform grid. Consequently, we need not worry about patching the marching-cubes solution.

Our choice of contouring algorithm does result in some limitations. In addition to creating poorly shaped triangles, marching cubes is nonadaptive. That is, the sampling is as dense in flat regions as in regions of high curvature. Unfortunately, the nonadaptive nature of marching cubes limits the resolution we can achieve in high-curvature areas, but is necessary to ensure compatibility. To address this lack of resolution in high-curvature areas, Strain [2001] split line segments whose centers were far from the surface, yielding arbitrarily high accuracy. Unfortunately, this splitting technique is not easily extended to three dimensions as splitting a triangle either creates an incompatible triangulation or produces even more poorly shaped triangles. It is also very difficult to guarantee that we will still have a manifold when the inserted vertices are moved to the surface. Alternatively, several adaptive contouring methods [Shu et al. 1995; Shekhar et al. 1996; Poston et al. 1998] seek to use adaptive grids and regain compatibility through various crack-patching techniques. Such methods could easily be used here and we plan to explore adaptive methods in future work.

Although we did not find it necessary, after the contouring step the mesh can be processed in any way that preserves the closed-manifold invariant. This optional processing might include smoothing the surface, improving the shape of the triangles, or any other operation that returns a closed manifold. A new distance tree can then be built from this modified mesh using Criterion (15). A new distance must be built only if the mesh is modified.

By taking advantage of the details of our method, we can very efficiently achieve limited smoothing in two ways. First, we can define a second composite function to be the combination of path tracing backward in time followed by the evaluation of a high-order polynomial interpolant of the distances at the vertices of the octree. This function is quite similar to the functions used in semi-Lagrangian level-set methods [Strain 1999b; Enright et al. 2005]. When marching cubes encounters an edge whose vertices have different signs, we find a point which evaluates to zero for each composite function. The final mesh vertex is an average of these two points. By constraining the mesh vertex to be on the edge of the marching-cubes grid, we still guarantee a consistent, closed, manifold triangulation. While this smoothing technique may be quite useful in some applications, we did not use this method for any of the results in this article. Second, repeatedly using the same grid for contouring can produce grid artifacts. For example, a sphere of fluid falling under gravity will develop creases along the coordinate axes. Such artifacts are a form of aliasing and can be reduced by jittering the grid each timestep. Most of the examples in this article used grids which were slightly larger than the simulation domain. These grids were then randomly perturbed so that grids at adjacent timesteps were slightly offset from one another.

This jittering limits the reusability of our octrees, but since we build new octrees every timestep, this limitation is not significant.

8. REDISTANCING

After the triangle mesh at the current timestep has been extracted, we must assign true distance values to the vertices of our octree. This problem, referred to as *redistancing*, has been well studied by the level-set community and a number of methods have been suggested. Strain [1999a] suggested redistancing by performing an exact evaluation at every vertex of the octree. This method is relatively efficient since the tree coarsens rapidly away from the surface and works well in two dimensions. However, in three dimensions, we have found it to be prohibitively expensive and unnecessary. Instead, we perform exact evaluation at all vertices of the cells that contain triangles, but then run a fast marching method [Sethian 1996; Losasso et al. 2004] over the remaining vertices. In our method, there may be some parts of the domain where the octree was refined but did not result in any triangles, such as when the surface becomes thinner than the resolution of the tree. Consequently, our octree, unlike those used by Losasso et al. [2004], does not necessarily coarsen away from the surface. To address this problem, we coarsen parts of the tree which have been refined but did not generate surface. We do this coarsening in two steps. First, we propagate the triangle lists up the tree so that the triangle list of a cell is the union of the triangle lists of a cell's descendants. Second, we remove all the children of any cell whose concentric triple does not contain any triangles.

Our redistancing method comprises three steps:

—coarsen the octree;

—compute exact distances at vertices of cells which contain triangles;

—run a fast marching method over the remaining vertices.

9. TRACKING SURFACE PROPERTIES

One of the primary advantages of our method is the ability to track surface properties, such as color, texture coordinates, or even simulation variables, accurately at negligible additional cost. As pointed out earlier, every vertex in a polygon mesh corresponds to some point on some triangle in the previous mesh. Thus, semi-Lagrangian advection provides a mapping between surfaces at adjacent timesteps. If vertex $\boldsymbol{v}$ in the current mesh maps to point $\boldsymbol{p}$ in the old mesh and some surface property was stored at $\boldsymbol{p}$, this property can be copied to $\boldsymbol{v}$. In this way we can track surface properties on the actual surface as we build the surface, so we do not incur any significant additional cost. Previous methods, such as the one proposed by Rassmussen et al. [2004], have been limited to tracking properties in the volume near the surface and interpolating them to the surface. Such methods incur significant cost, introduce substantial smoothing, and blur properties between nearby surfaces.

In many applications there is no value actually stored at $\boldsymbol{p}$. Instead, the properties are stored at the vertices of the triangle containing $\boldsymbol{p}$. In these cases the problem is slightly more involved. In many cases it is sufficient to use barycentric interpolation to compute a value at $\boldsymbol{p}$ and copy this interpolated value to $\boldsymbol{v}$. However, for some applications this interpolation can produce unwanted smoothing. A simple alternative is to set the value at $\boldsymbol{p}$ to the value stored at the vertex nearest $\boldsymbol{p}$. Unfortunately, this approach may introduce unwanted aliasing. Essentially, we are having trouble because we are resampling the surface at every timestep. However, if we know something about the property we are tracking, we may be able to "clean up" the blurred signal. For example, in our examples with checkerboard textures, we tracked reference coordinates which were passed to a simple function to determine color. Since we know that the tracked value should always be a point on the initial surface we could find the point on the

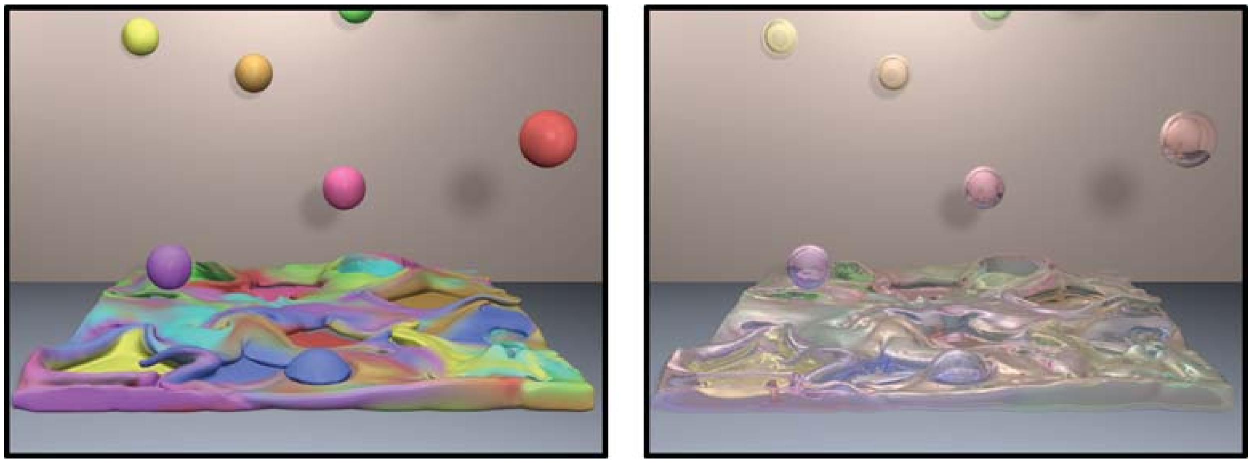

Fig. 5. These images shows an invisible tank being filled as multicolored balls of fluid fall into it. The resulting surface contains complex geometric details which retain the different colors of the balls. The left image was rendered with a matte shader, while the right image was rendered with a colored glass shader.

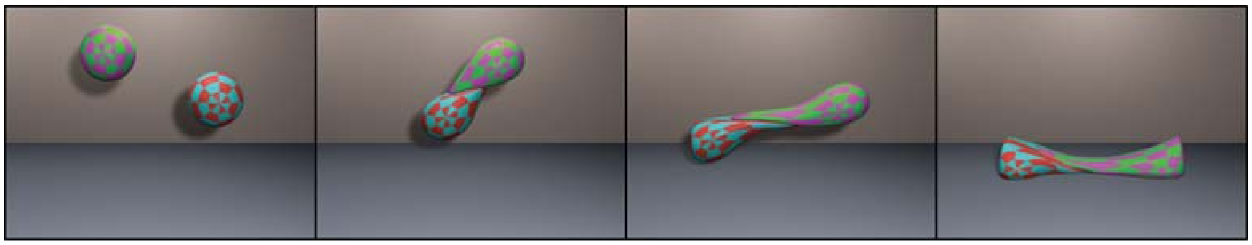

Fig. 6. Two balls of viscoelastic fluid are thrown at each other and merge.

initial mesh which was closest to the value the tracking method supplied. In this way, we ensured that, at every timestep, every vertex in the mesh mapped back to some point on the initial surface. Once we had this mapping we could copy any property stored on the initial surface, whether it be the reference coordinates, texture coordinates, or color values. Thus, if image textures were preferred over procedural textures, texture coordinates could be copied instead of reference coordinates.

There are still plenty of open problems in the area of texturing liquid surfaces. In particular, it is difficult to deal with large discontinuities in surface properties, which occur when two surfaces merge, or a surface splits. Creating detail where a surface stretches is also an open problem.

10. RESULTS AND DISCUSSION

We have tested this surface tracking method coupled with a fluid simulation on several examples such as the ones shown in Figures 5 and 6. We also tested it in the spiraling analytical test field from Enright et al. [2002a]. Figure 7 shows two objects being advected in this divergence-free velocity field to a mid-point after which the field reverses. The sphere was restored to a nearly identical shape (see Figure 8), while the bunny exhibited a small amount of smoothing. The surface of the bunny was textured by a spot-generating reaction-diffusion system that ran on the surface as the object was being advected

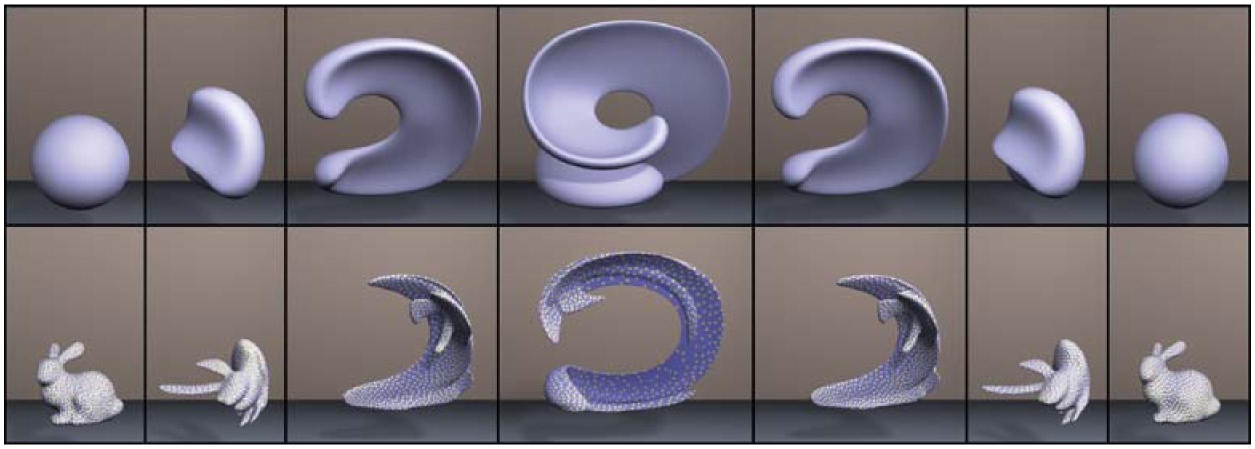

Fig. 7. This figure shows the behavior generated when two different surfaces are passed through an analytical flow field that stretches and distorts them. The first three images show the object flowing along the field, the last three show the behavior when the distorted object then flows back along the reverse field. The bunny is textured using a reaction-diffusion system that is running on the surface during the sequence.

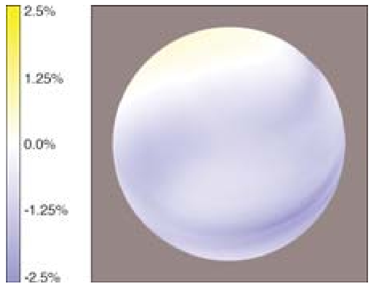

Fig. 8. This figure shows the error in the final frame of the sphere example in Figure 7. The color maps to the error as a percentage of the sphere's radius, with blue points slightly inside and yellow points slightly outside.

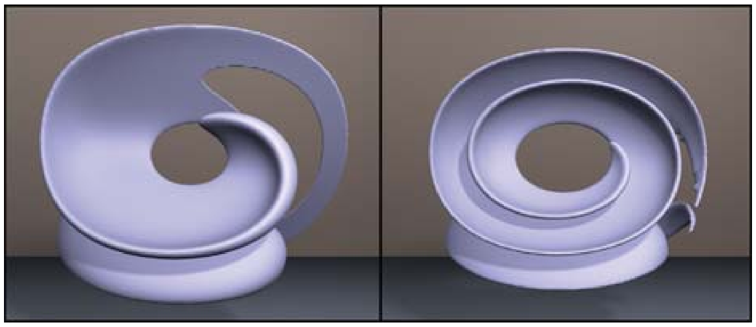

Fig. 9. In this figure we show the result generated when we continue to distort and stretch an object past the point where it thins out and tears.

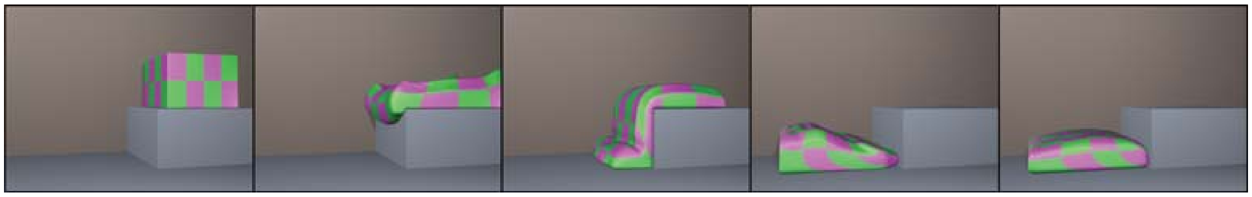
Fig. 10. This sequence shows a thick viscoelastic fluid sliding off of a shelf. A checkerboard texture is mapped onto the surface. It is interesting to notice that the corners of the checkerboard texture stay sharp, despite the significant deformation.

[Turk 1991; Witkin and Kass 1991]. At each timestep, the morphogens were advected along with the surface and then allowed to react. In Figure 9 we show the result of running the sphere through the flow field for several revolutions to highlight the behavior generated when the surface thins below the resolution of the octree's finest level.

All of our fluid examples used a standard regular-grid Eulerian fluid simulator with the elasticity model of Goktekin et al. [2004]. The fluid simulator and the surface tracking module were only very loosely coupled: the fluid simulator provided the surface tracker with a velocity function and, in turn, the surface tracker provided the simulator with the signed-distance function. Because our fluid simulator has a regular grid its resolution is notably coarser than the surface tracker, which uses an octree. The idea of using different resolutions for the fluid and surface is not new; Foster and Fedkiw [2001] used different timesteps for their fluid and surface calculations and Goktekin et al. [2004] found that increasing the spatial resolution of the surface tracking grid dramatically reduced volume loss. As noted by Losasso et al. [2004], using different spatial resolutions can produce artifacts. For example, pieces of surface could appear connected when the simulator thinks they are disconnected and vice versa. Additionally, surface features may be maintained when a more detailed fluid simulator would smooth them away. In general, we found the increased surface resolution to be worth these artifacts. Ideally we would use a multiresolution fluid simulation, like the octree method of Losasso et al. [2004]. We plan to incorporate a multiresolution fluid simulator as part of our future work.

For most of our examples the surface tracking module took roughly 1 min/timestep at an effective resolution of 512^3. The fluid simulation also required about 1 min/timestep. Both the fluid simulator and the surface tracking module took 11 timesteps per frame. Thus it took about 2 days to simulate 10 s of animation, with roughly half the time spent solving for the velocity field and half the time updating the surface. It is important to note that, given a perfect semi-Lagrangian path tracer, the method could take arbitrarily large timesteps. Decoupling the timesteps of the fluid simulator and surface tracker, so that the surface tracker runs only once per frame, is an interesting area of future work.

In Figure 10 we show the behavior when a thick viscoelastic fluid is allowed to flow off a shelf into a basin. This surface is textured by advecting reference coordinates along with the flow and applying a procedural checkerboard texture. Figure 11 shows beginning and ending frames using both an off-the-shelf procedural shader, which includes a displacement map, and a reaction-diffusion system. The motion of the spots on the surface occurs both from the motion of the surface and from the reaction-diffusion system seeking equilibrium on the moving surface.

Figures 12 and 13 show two streams of liquid that are being sprayed toward each other. As the streams oscillate from side-to-side, they collide and produce a thin, web-like surface between them. The motion of the two streams causes this thin surface to form a spiral shape as the streams separate. Similar effects can be seen in real-world footage.

All of our images were rendered with the open-source renderer Pixie [Arikan 2005]. Since we generated a polygonal mesh for each frame, we could take advantage of standard rendering techniques,

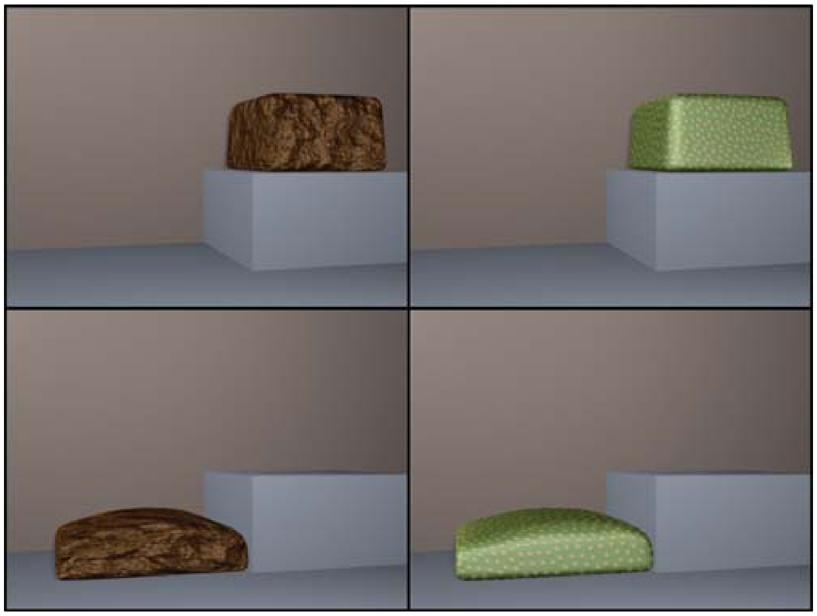

Fig. 11. This figure shows the beginning and ending frames of an animation similar to that shown in Figure 10. The left images were rendered with an off-the-shelf procedural texture which includes a displacement map, while the images on the right were generated with a reaction-diffusion texture.

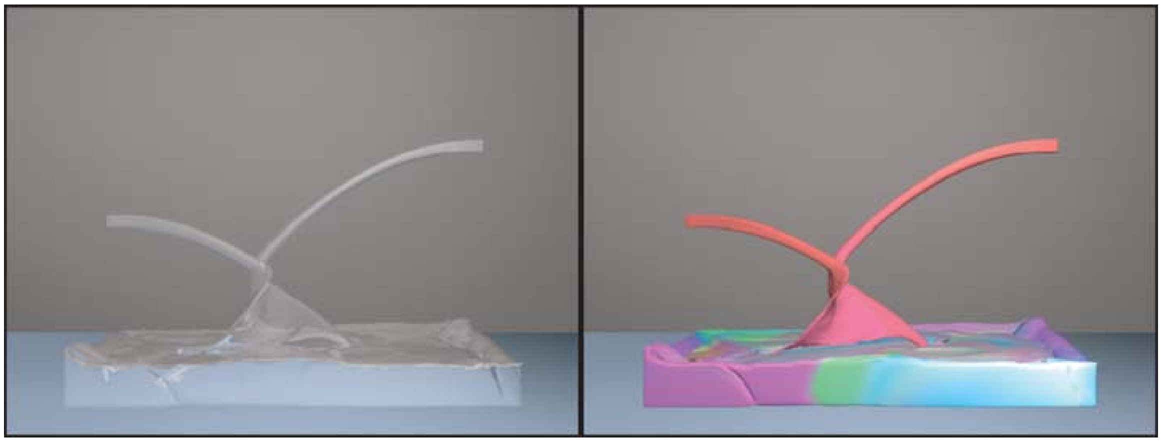

Fig. 12. These images show closeups of the fluid web created between two intersecting sprays. The left image is rendered realistically, the right is rendered with a matte shader where the color has been advected with the flow.

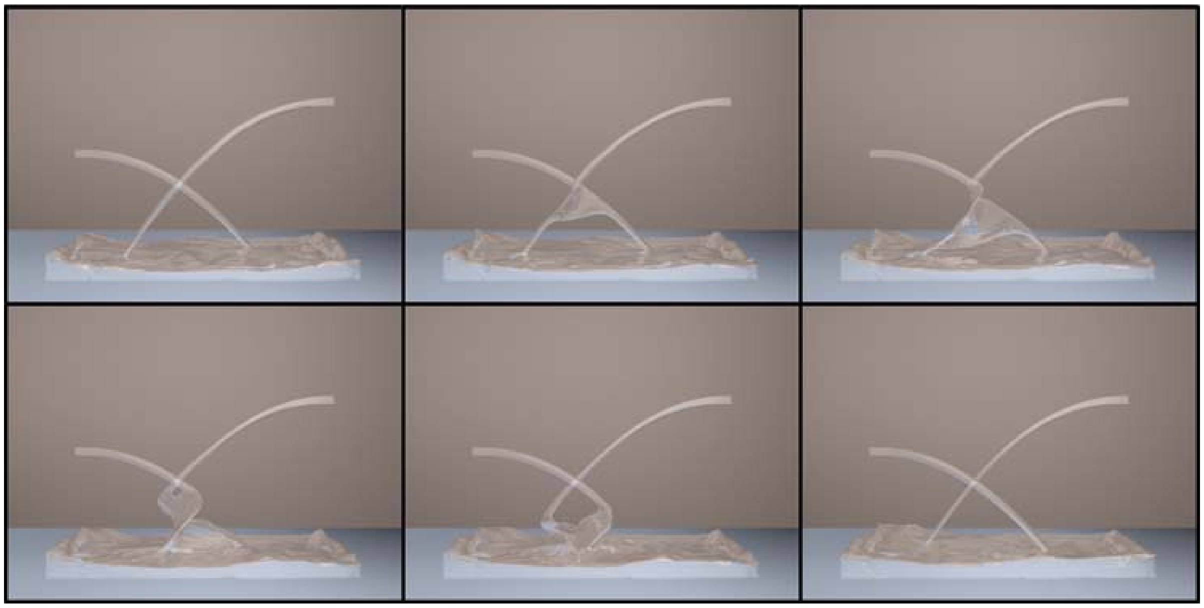
Fig. 13. This sequence shows a realistic rendering of two liquid sprays. As the sprays move from side to side, they periodically intersect and create a web-like spiral pattern.

allowing for very fast rendering times; most of our renderings took less than 3 min/frame. Many of our examples were rendered with a matte shader so that the surface detail can be seen. A number of our examples were also rendered with a glass shader (using water's index of refraction) for comparison to previous methods and real fluids, and to demonstrate how the method can be used to generate realistic results. Our colored and textured examples illustrate how easily a variety of properties may be attached to the surface. In practice, we believe that advected properties could be used effectively with standard shading techniques to generate a wide range of interesting effects.

11. CONCLUSIONS

Semi-Lagrangian contouring offers an elegant and effective means for surface tracking and has a number of advantages over competing methods. First, we have an explicit representation. In addition to enabling exact evaluation, this explicit representation also allows us to leverage 30 years of computer graphics technology which has been optimized for polygonal meshes. Rendering, texture mapping, and a variety of other applications are all very straightforward. Second, we have an implicit representation. This implicit representation allows us to update the surface without explicitly addressing any of the difficult topological issues which plague other approaches. Third, semi-Lagrangian advection gives us a mapping between surfaces at adjacent timesteps. This mapping allows us to accurately track surface properties on the actual surface at negligible complexity and cost. Fourth, our method does not have any ad hoc rules or parameters to tune. In fact, the only parameters to our system are the upper and lower corners of the domain, the maximum depth of the octree (a resolution parameter), and some resolution tolerances. Finally, and most importantly, we are able to produce detailed, flicker-free animations of complex fluid motions.

ACKNOWLEDGMENTS

We thank the other members of the Berkeley Graphics Group for their helpful criticism and comments. In particular, we thank Ravi Kolluri, Okan Arikan, and Bryan Feldman for many fruitful discussions. We also thank the anonymous reviewers for their many helpful comments and suggestions.

REFERENCES

ARIKAN, O. 2005. Pixie: Photorealistic renderer. Go online to `http:// www.cs.utexas.edu/~okan/Pixie/pixie.htm`.

BÆRENTZEN, J. A. AND AANÆS, H. 2002. Computing discrete signed distance fields from triangle meshes. Tech. rep. Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

BÆRENTZEN, J. A. AND CHRISTENSEN, N. J. 2002. Interactive modelling of shapes using the level-set method. *Int. J. Shape Model. 8*, 2, 79–97.

BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*. Academic Press Professional, Inc., San Diego, CA, 324–349.

BOISSONNAT, J. D. AND OUDOT, S. 2003. Provably good surface sampling and approximation. In *SGP '03: Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. Eurographics Association, Aire-la-Ville, Switzerland, 9–18.

CANI, M.-P. AND DESBRUN, M. 1997. Animation of deformable models using implicit surfaces. *IEEE Trans. Visual. Comput. Graph. 3*, 1 (Jan.), 39–50.

CARLSON, M., MUCHA, P. J., R. BROOKS VAN HORN, I., AND TURK, G. 2002. Melting and flowing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM Press, New York, NY, 167–174.

CARLSON, M., MUCHA, P. J., AND TURK, G. 2004. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. 23*, 3, 377–384.

COURANT, R., ISAACSON, E., AND REES, M. 1952. On the solution of nonlinear hyperbolic differential equations by finite differences. *Comm. Pure Appl. Math 5*, 243–249.

DESBRUN, M. AND CANI, M.-P. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of Computer Animation and Simulation 1996*. 61–76.

DESBRUN, M. AND GASCUEL, M.-P. 1995. Animating soft substances with implicit surfaces. In *the Proceedings of SIGGRAPH 95*, 287–290.

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

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2005. A fast and accurate semi-Lagrangian particle level set method. *Comput. Struct. 83*, 479–490.

ENRIGHT, D. P., MARSCHNER, S. R., AND FEDKIW, R. P. 2002b. Animation and rendering of complex water surfaces. In *Proceedings of ACM SIGGRAPH 2002*. 736–744.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *the Proceedings of ACM SIGGRAPH 2001*. 15–22.

FOSTER, N. AND FEDKIW, R. 2001. Practical animation of liquids. In *the Proceedings of ACM SIGGRAPH 2001*. 23–30.

FOSTER, N. AND METAXAS, D. 1996. Realistic animation of liquids. In *Proceedings of Graphics Interface 1996*. 204–212.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY/Addison-Wesley Publishing Co., Reading, MA, 249–254.

GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. 2004. A method for animating viscoelastic fluids. In *Proceedings of ACM SIGGRAPH 2004*. 463–468.

GREEN, D. AND HATCH, D. 1995. Fast polygon-cube intersection testing. In *Graphics Gems V*. Academic Press Professional, Inc., San Diego, CA, 375–379.

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.

HILTON, A., STODDART, A. J., ILLINGWORTH, J., AND WINDEATT, T. 1996. Marching triangles: Range image fusion for complex object modelling. In *Proceedings of the International Conference on Image Processing*. 381–384.

HIRT, C. W. AND NICHOLS, B. D. 1981. Volume of fluid (VOF) method for the dynamics of free boundaries. *J. Computat. Phys. 39*, 201–225.

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

HOUSTON, B., NIELSEN, M. B., BATTY, C., NILSSON, O., AND MUSETH, K. 2006. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Trans. Graph. 25*, 1, xx–xx.

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, 339–346.

LEVEQUE, R. J. 1990. *Numerical Methods for Conservation Laws*. Birkhauser-Verlag, Basel, Switzerland.

LORENSEN, W. E. AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, 163–169.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. In *Proceedings of ACM SIGGRAPH 2004*. 457–462.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *ACM SIGGRAPH 2003 Symposium on Computer Animation*. 154–159.

MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*.

NIELSEN, M. B. AND MUSETH, K. 2006. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *J. Sci. Comput. 26*, 1, 1–39.

OSHER, S. AND FEDKIW, R. 2003. *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York, NY.

OSHER, S. AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J Computat. Phys. 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.

POSTON, T., WONG, T.-T., AND HENG, P.-A. 1998. Multiresolution isosurface extraction with adaptive skeleton climbing. *Comput. Graph. For. 17*, 3 (Sept.), 137–148.

PREMOŽE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle-based simulation of fluids. *Comput. Graph. For. 22*, 3 (Sept.), 401–410.

RASMUSSEN, N., ENRIGHT, D., NGUYEN, D., MARINO, S., SUMNER, N., GEIGER, W., HOON, S., AND FEDKIW, R. 2004. Directable photorealistic liquids. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*. ACM Press, New York, NY, 193–202.

SAMET, H. 1990. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Longman Publishing Co., Inc., Reading, MA.

SCHNEIDER, P. J. AND EBERLY, D. H. 2002. *Geometric Tools for Computer Graphics*, 1st ed. Morgan Kaufmann, San Francisco, CA.

SETHIAN, J. A. 1996. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci. USA 93*, 4 (Feb.), 1591–1595.

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

SHEKHAR, R., FAYYAD, E., YAGEL, R., AND CORNHILL, J. F. 1996. Octree-based decimation of marching cubes surfaces. In *VIS '96: Proceedings of the 7th conference on Visualization '96*. IEEE Computer Society Press, Los Alamitos, CA, 335ff.

SHU, R., CHEN, Z., AND KANKANHALLI, M. S. 1995. Adaptive marching cubes. *Vis. Comput. 11*, 202–217.

STAM, J. 1999. Stable fluids. In *the Proceedings of ACM SIGGRAPH 99*. 121–128.

STORA, D., AGLIATI, P.-O., CANI, M.-P., NEYRET, F., AND GASCUEL, J.-D. 1999. Animating lava flows. In *Proceedings of Graphics Interface 99*. 203–210.

STRAIN, J. A. 1999a. Fast tree-based redistancing for level set computations. *J. Computat. Phys. 152*, 2 (July), 648–666.

STRAIN, J. A. 1999b. Semi-Lagrangian methods for level set equations. *J. Computat. Phys. 151*, 2 (May), 498–533.

STRAIN, J. A. 1999c. Tree methods for moving interfaces. *J. Computat. Phys. 151*, 2 (May), 616–648.

STRAIN, J. A. 2000. A fast modular semi-Lagrangian method for moving interfaces. *J. Computat. Phys. 161*, 2 (July), 512–536.

STRAIN, J. A. 2001. A fast semi-Lagrangian contouring method for moving interfaces. *J. Computat. Phys. 169*, 1 (May), 1–22.

SUSSMAN, M. AND PUCKETT, E. G. 2000. A coupled level set and volume-of-fluid method for computing 3D and axisymmetric incompressible two-phase flows. *J. Comput. Physat. 162*, 2, 301–337.

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

Turk, G. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. In *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, 289–298.

Wang, H., Mucha, P. J., and Turk, G. 2005. Water drops on surfaces. *ACM Trans. Graph. 24*, 3, 921–929.

Witkin, A. and Kass, M. 1991. Reaction-diffusion textures. In *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, 299–308.

Wyvill, G., McPheeters, C., and Wyvill, B. 1986. Data structure for *soft* objects. *Vis. Comput. 2*, 4, 227–234.

Zhu, Y. and Bridson, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 3, 965–972.

Received September 2005; accepted December 2005