Fluid Animation with Dynamic Meshes

Bryan M. Klingner Bryan E. Feldman Nuttapong Chentanez James F. O'Brien
University of California, Berkeley

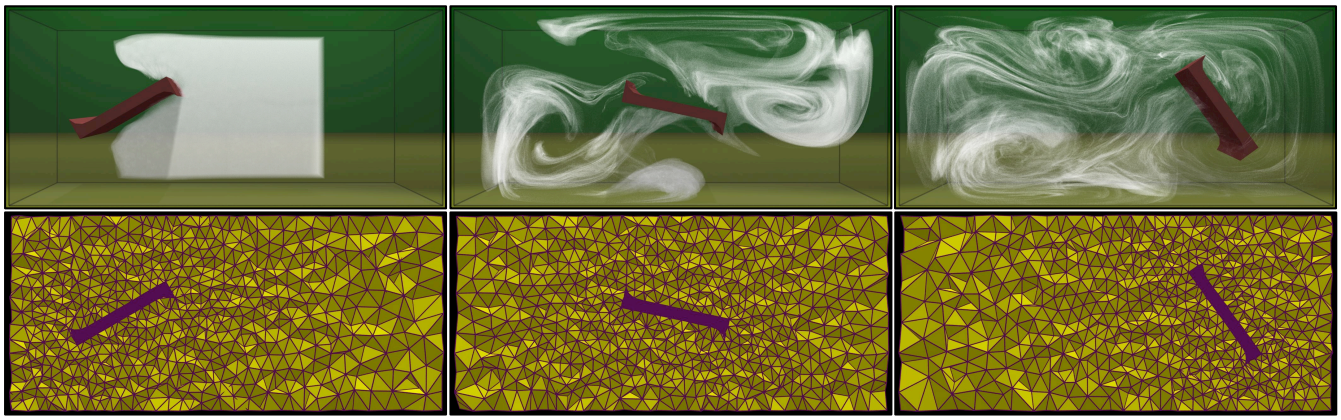

Figure 1: Top: A paddle mixes smoke in a tank. Bottom: A cross-section of the simulation meshes used for each frame.

Abstract

This paper presents a method for animating fluid using unstructured tetrahedral meshes that change at each time step. We show that meshes that conform well to changing boundaries and that focus computation in the visually important parts of the domain can be generated quickly and reliably using existing techniques. We also describe a new approach to two-way coupling of fluid and rigid bodies that, while general, benefits from remeshing. Overall, the method provides a flexible environment for creating complex scenes involving fluid animation.

Keywords: Natural phenomena, physically based animation, computational fluid dynamics.

CR Categories: 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.

1 Introduction

Although systems for physically based fluid animation have developed rapidly in recent years and can now reliably generate production-quality results, they still have some limitations. Simulation domains can change substantially from step to step because of deforming boundaries, moving obstacles, and evolving fluid motion, yet current systems based on fixed grids are not ideally suited to handle these situations.

E-mail: {klingner|feldman|nchentan|job}@eecs.berkeley.edu

We propose a method to simulate fluids with such rapidly changing domains by generating a new tetrahedral simulation mesh at each time step. When generating the mesh, we use the position and shape of boundaries as well as criteria based on the visually important parts of the fluid and velocity field to construct a sizing field that dictates the desired edge length for tetrahedra throughout the domain. We then use an efficient and reliable meshing algorithm adapted from [Alliez et al., 2005] to produce a mesh that is refined according to this field. We use unstructured tetrahedral meshes because they conform to curved and irregular boundaries better than axis-aligned grids with the same number of grid elements and allow for precise control of refinement throughout the domain.

We transfer the physical properties of the simulation from the old mesh to the new mesh using a generalization of the semi-Lagrangian velocity advection technique that introduces no additional smoothing. We then perform a mass conservation step that has been extended to allow a new, single-step solution of two-way coupling between fluid and rigid bodies.

Overall, this approach provides a flexible framework for fluid simulation that opens the door to many features. We have implemented the system and tested it in a variety of scenarios such as the one shown in Figure 1. We have found that the combination of unstructured tetrahedral domains and dynamic remeshing creates a versatile environment for the creation of complex and visually interesting fluid animations.

2 Background

The animation of fluids through physical simulation has become an important tool in the visual effects industry. One approach that has been popular in recent years makes use of a spatial discretization based on regular, fixed, hexahedral grids. Some examples of this approach can be found in [Foster and Metaxas, 1996], [Foster and Metaxas, 1997], [Stam, 1999], [Yngve et al., 2000], [Fedkiw et al., 2001], [Foster and Fedkiw, 2001], [Enright et al., 2002], [Carlson et al., 2002], [Feldman et al., 2003], and [Goktekin et al., 2004]. The most

commonly used storage scheme for these approaches is the "staggered grid" scheme. This method offsets storage of different quantities on the grid, and was first described by [Harlow and Welch, 1965]. Efforts have been made to enhance these methods to allow for better conformance to irregular boundaries such as the free surface of liquids, complex obstacles, or irregularly shaped domains. [Losasso et al., 2004] described an octree-based method that retains many of the advantages of regular grids while allowing computational effort to be focused in particular parts of the simulation domain; this enables detailed tracking of moving boundaries such as liquid surfaces. Both [Carlson et al., 2004] and [Guendelman et al., 2005] have demonstrated methods for two-way coupling of obstacles to fluid.

Unstructured tetrahedra have also been used for fluid simulation within the graphics community. Two examples of this are [Feldman et al., 2005a] and [Elcott et al., 2005]. The first method uses a velocity-based approach while the second uses a vorticity-based formulation. It is a blend of ideas from these two papers, along with a generalization of the semi-Lagrangian velocity advection technique for moving meshes described in [Feldman et al., 2005b] that forms the heart of our method.

The idea of moving meshes independent of a fixed or particle-centric coordinate system is not a new one; arbitrary Lagrangian-Eulerian (ALE) methods were designed for just this purpose. They have proven useful in the simulation of highly deformable elastic materials. ALE was first described in [Hirt et al., 1974], where it was used with finite differences to solve compressible fluid problems. [Donea et al., 1977] went on to apply ALE in a finite element setting. An excellent survey of the development of ALE methods appears in [Donea et al., 2004]. Examples within the graphics literature that feature moving meshes without remeshing include [Shah et al., 2004] and [Rasmussen et al., 2004], both of which translate the grid to follow the visually important portion of the fluid.

Another approach to handling changing domains is to dispense with the mesh altogether, instead using Lagrangian particles for simulation of fluids. A few examples of this approach are [Terzopoulos et al., 1989], [Desbrun and Cani, 1996], [Cani and Desbrun, 1997], [Stora et al., 1999], [Müller et al., 2003], [Premože et al., 2003], and [Müller et al., 2004]. These meshless methods are particularly well suited to changing domains because points can move freely without concerns about mesh quality.

Because we regenerate a new simulation mesh at each time step, the viability of our method hinges on fast, high-quality, reliable tetrahedral mesh generation. While a history of unstructured mesh generation is outside the scope of this paper, [Owen, 1998] and [Teng and Wong, 2000] provide good surveys of the field. For our mesh generator we selected the approach described in [Alliez et al., 2005]. This innovative method produces meshes which conform to domains of arbitrary topology quickly and reliably. Also, it allows for the local edge length of the tetrahedra to be specified arbitrarily throughout space, which allows us to easily perform adaptive mesh refinement from step to step. The meshes produced by this technique are Delaunay, which provides improved gradient estimation and allows us to significantly simplify some of the expressions that arise when interpolating velocity values stored on the mesh.

3 Methods

The key contribution of our method is to demonstrate the freedom granted by remeshing at each simulation time step. The core of our system is based on the simple, efficient methods for discretizing the inviscid Euler equations on tetrahedral meshes described in [Elcott et al., 2005] and [Feldman et al., 2005a]. We have made a few modifications in order to combine the best aspects of both approaches that are described below.

Once we have a good discretization, we need a way to propagate information from one mesh to the next. [Feldman et al., 2005b] details a generalization of the standard semi-Lagrangian velocity advection technique that allows simulation state to be transferred between deforming domains without incurring additional smoothing. We demonstrate that their approach can easily be applied to transfer information between two arbitrary, topologically unrelated meshes, which is required to achieve more general evolution of the simulation domain from step to step.

Finally, we need to quickly and reliably generate a new tetrahedral mesh for each time step that suits the current simulation conditions, such as conformance to boundaries and obstacles as well as any desired refinement. Although methods have long existed to mesh arbitrary domains, most are relatively slow in comparison to simulation running times or don't reliably terminate under realistic conditions. The availability of efficient, versatile meshing algorithms such as [Alliez et al., 2005] has made the generation of a new mesh at each time step practical.

Any changes that were required to make these pieces work together harmoniously are discussed below. Also, we describe a new, single-step method to achieve two-way coupling between obstacle and fluid motion.

3.1 Discretization

We use a staggered fluid state storage scheme that stores pressures at tetrahedron circumcenters and "face-normal velocities," the component of velocity in the direction of the face normal, at the face circumcenters. Similar schemes have been used in [Botta and Hempel, 1996], [Elcott et al., 2005] and [Feldman et al., 2005a]. These methods are a generalization of the staggered grid scheme originally proposed by [Harlow and Welch, 1965]. This staggered method is used to discretize the inviscid Euler equations:

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

subject to the mass conservation constraint for incompressible fluids:

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

In these equations, $\mathbf{u}$ is the fluid velocity, t time, p pressure, ρ density, and $\mathbf{f}$ any external forces. The symbol ∇ denotes the vector of differential operators $\nabla = [\partial/\partial x, \partial/\partial y, \partial/\partial z]^{\mathsf{T}}$. We account for the changes in the mesh over a time step directly during semi-Lagrangian advection (see Section 3.2).

3.1.1 Discrete Derivative Operators

Divergence and gradient operators are needed as part of the mass conservation step. We make discrete estimates of these derivatives following the formulation presented in [Losasso et al., 2004] and [Elcott et al., 2005]. The divergence of a tetrahedron is computed as an area weighted sum of the tetrahedron's face normal velocities. The gradient at a face circumcenter in the direction of the face's normal is computed using finite differences. The difference in circumcenter pressures adjacent to a face is divided by the distance between these circumcenters. In Delaunay meshes, the line connecting adjacent tetrahedra circumcenters passes through the circumcenter of the face between them and is in the direction of that face's normal. This property of Delaunay meshes motivates our storage scheme at circumcenters because the gradient estimate is equivalent to the gradient of a piecewise linear function that interpolates the circumcenter values.

3.1.2 Velocity Interpolation

The staggered scheme stores only the component of velocity in the face normal direction. For both the semi-Lagrangian step and to advect smoke particles for rendering, a full velocity vector must be found at arbitrary positions in the mesh. We interpolate velocity vectors from face normal velocities using the two-step method developed in [Elcott et al., 2005]. First, a velocity vector, $\mathbf{u}_t$, is computed at each tetrahedron circumcenter, then we interpolate within Voronoi cells using $\mathbf{u}_t$ values at the cell vertices.

Velocity $\mathbf{u}_t$ for tetrahedron t is found by solving the small linear system

$$\mathbf{N}_t \mathbf{u}_t = \mathbf{z}_t$$

where $\mathbf{N}_t$ is a matrix containing 4 rows of the face normals of t and $\mathbf{z}_t$ is a vector of the 4 face normal velocities associated with t. For a divergence-free field, this solution has the remarkable property that interpolating back to the face circumcenters exactly recovers the original face-normal velocities. Thus interpolating the $\mathbf{u}_t$ velocities also exactly interpolates the face-normal velocity components, and does not incur the error one would otherwise expect from a two-step interpolation method.

To find a velocity at an arbitrary point we interpolate within the Voronoi cell using the tetrahedra velocities associated with the cell. This interpolation is based on the method of [Warren et al., 2004], which presents a way to interpolate within a general convex polytope. They interpolate the value at the point $\mathbf{x}$ as a weighted sum of the polytope's node values where node t's unnormalized weight is computed as

$$w_t(\mathbf{x}) = \frac{|\mathcal{N}_t|}{\prod_{f \in \sigma_t} \mathbf{n}_f \cdot \mathbf{x} + d_f}. \tag{3}$$

Here, σ_t is the set of polytope faces that intersect at node t. The denominator is the product of distances from $\mathbf{x}$ to the faces in σ_t computed using the face normals,$\mathbf{n}_f$, and plane offsets, d_f. $|\mathcal{N}_t|$ is the determinant of a matrix of face normals in σ. Weights from all nodes are normalized to sum to 1 before use in the weighted sum. To simplify this computation we take advantage of two properties: 1) in a Delaunay mesh, edges are in the direction of the Voronoi cell's face normals and 2) the volume of tetrahedron t is $1/6|\mathbf{E}_t|$ where $\mathbf{E}_t$ is a matrix formed from the three vectors of edges emanating from a common node of t. After some manipulation, which is omitted for brevity, Equation (3) applied to node weights within a Voronoi cell can be simplified to

$$w_t(\mathbf{x}) = \frac{6\mathrm{Vol}(t)}{\prod_{i=1}^{3} (\mathbf{p}_i - \mathbf{p}_v) \cdot (\mathbf{c}_t - \mathbf{x})} \tag{4}$$

where $w_t(\mathbf{x})$ is the weight associated with the node at tetrahedra t's circumcenter, $\mathrm{Vol}(t)$ is the volume of tetrahedron t, $\mathbf{p}_v$ is the position of the node associated with the Voronoi cell, $\mathbf{p}_i$ are positions of the other nodes of t, $\mathbf{c}_t$ the circumcenter of t, and $\mathbf{x}$ the interpolation position. A similar observation appears in [Ju et al., 2005], and we find that with it the velocity interpolation is quite efficient. All quantities appearing in Equation (4) are already stored for use in other parts of the timestep, saving the need to compute the terms in Equation (3). When advecting large numbers of particles, velocities at nodes of tetrahedra can be first be found using Equation (4) and then quickly interpolated in a linear fasion over the tetrahedra to advect the particles.

3.2 Generalized Semi-Lagrangian Step

The simple and stable semi-Lagrangian method has become the standard tool for advection of the velocity field for graphical applications [Stam, 1999]. The basic idea of the method is that we can find a velocity that will advect to a point by tracing back from that point and interpolating the old velocity field. This method does not rely on velocities being stored at any particular place, as long as the velocity can be interpolated throughout space. We can extend this technique naturally to meshes which change arbitrarily at each time step as in [Feldman et al., 2005b]. This extension does not incur any additional smoothing compared to using semi-Lagrangian advection with static meshes.

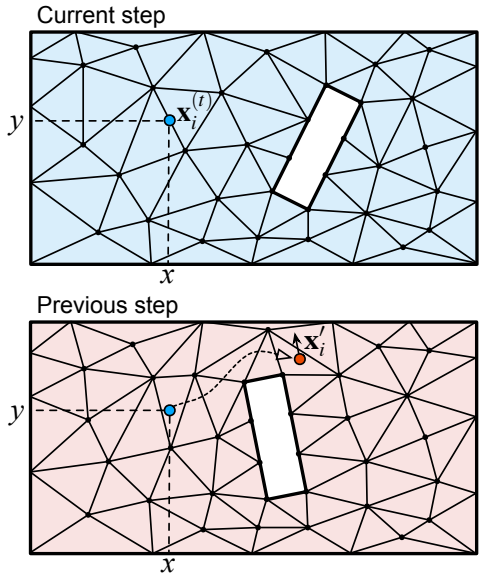

Figure 2: A two-dimensional representation of the generalized semi-Lagrangian advection step. We trace back from the position where a velocity is stored in the new mesh, $\mathbf{x}_i = (x, y)$, interpolate the velocity using the old mesh and velocity field, and update the velocity in the new mesh.

Suppose at time t velocities are stored at locations $\mathbf{x}^{(t)}$ (in our case, the face circumcenters), and we want to find the velocity at a particular face location $\mathbf{x}_i^{(t)}$. We trace back from $\mathbf{x}_i^{(t)}$ through the velocity field of the previous time step to a point $\mathbf{x}_i'$, which has no necessary correspondence to any feature of the old mesh. Then, we update the velocity at $\mathbf{x}_i^{(t)}$ to the value interpolated from the old velocity field at $\mathbf{x}_i'$. Because the velocities from the previous step are stored on a different mesh, we have to trace back and interpolate using this previous mesh (see Figure 2).

3.3 Remeshing

The domain boundaries, obstacles, and smoke are free to move and change from step to step of the simulation. By regenerating the mesh at each time step we can ensure that our domain conforms well to boundaries and is refined in visually important areas. We accomplish this by using the variational tetrahedral meshing algorithm presented in [Alliez et al., 2005]. This method allows for generation of tetrahedral meshes that conform well to an arbitrary input surface mesh, have no restrictions on topology (*i.e.*, allow nested voids), and allow for sizing of tetrahedra throughout the domain based on arbitrary criteria.

Our implementation differs from the original algorithm in a couple of details. As in the original method, refinement of the mesh is controlled by a sizing function $\mu(\mathbf{x})$ that, for any point $\mathbf{x}$ in the simulation domain, returns the desired local edge length of the tetrahedra. While the original algorithm builds this sizing function by finding the minimum combination of local feature size and distance to a boundary point

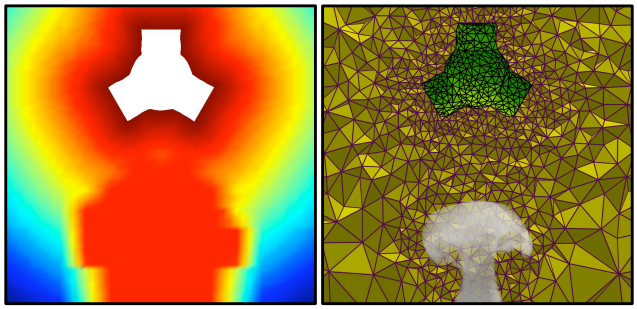

Figure 3: Left: a visualization of the sizing field for a rectangular domain with an irregular obstacle at the top and a plume of smoke at the bottom. Right: the resulting simulation mesh. Obstacle faces are colored green.

from $\mathbf{x}$, we instead formulate it as follows:

$$\mu(\mathbf{x}) = k_0 + \min\left(k_d d(\mathbf{x}), k_s\left(1 - s(\mathbf{x})\right), k_\omega\left(1 - \omega(\mathbf{x})\right)\right) \quad (5)$$

In this equation, k_0 is an offset value that controls the minimum value of the sizing field, and hence the minimum local edge length of tetrahedra. $d(\mathbf{x})$ is the distance to the closest obstacle or boundary which demands refinement, $s(\mathbf{x})$ is a function of the density of smoke particles, and $\omega(\mathbf{x})$ is a function of the vorticity of the velocity field. The parameters k_d, k_s, and k_ω respectively control the weight each of these functions has on the sizing field. These three factors are the same as those used for octree refinement in [Losasso et al., 2004]. The overall goal of the sizing field is to focus computational effort in the most visually important parts of the scene, that is, near closed boundaries, where the velocity field varies most, and where smoke is visible. Figure 3 shows an example of a sizing field and the resulting mesh. Figure 4 demonstrates the benefits of refinement near areas of high vorticity and smoke density.

This meshing method is iterative, so the mesh from the previous simulation time step can be used as an initial guess for the node placement in the mesh at the next simulation time step. Because there is, in general, strong temporal coherence between steps of the simulation, the sizing field does not change too much and so the nodes from the previous step are often a good initialization. Before the algorithm proceeds, the initial node placement is corrected to match the sizing field of the current step.

One other modification we made to the algorithm is that, when optimizing the node positions, we move nodes to the average of the barycenters of the surrounding tetrahedra instead of the circumcenters. We have found that while this tends to slightly decrease the average quality of tetrahedra in the mesh, it often leads to substantial improvements in the quality of the worst elements of the mesh, which are of more concern for numerical simulation.

Of course, remeshing takes time, so it is important to consider the impact it has on overall simulation performance. The time spent generating meshes for each simulation step varies, but generally accounts for less than a quarter of the overall simulation time. In Section 4 we show timing information for several examples.

3.4 Two-way Coupling and Mass Conservation

The motion of fluid and rigid bodies that mutually effect each other can be complex and visually appealing. The interaction occurs as a consequence of the conditions that:

1. The velocities in the normal direction are the same at the interface of the fluid and the rigid body surface.
2. The fluid velocity is divergence free and the rigid body velocity is rigid.
3. The linear and angular momentum of the combined system is conserved.

In [Carlson et al., 2004] these conditions are enforced sequentially. While for many cases this produces results that look very good, under some situations artifacts can be created because enforcing one of the conditions in general will break a previously enforced one. Examples of such artifacts might be fluid leaking through solid boundaries or poor performance in piston-like situations. Our implementation differs from [Carlson et al., 2004] in a couple of ways, but most significantly we enforce these conditions simultaneously within the mass conservation step.

In general, the mass conservation step solves for pressures that accelerate the velocity field to be divergence free. In previous works, including those with two-way coupling, the mass conservation step treats faces to behave as fluid or explicitly prescribes their velocities. For fluid faces, the pressure accelerates the velocity proportional to the gradient of the pressure while for prescribed faces, the pressure does not effect the fluid. For a more complete discussion of fluid/prescribed-velocity mass conservation see [Fedkiw et al., 2001].

We extend mass conservation to include a dynamic, rigid body. To do so, we solve for acceleration of the fluid and the rigid body, ignoring pressure for both. We then solve for a pressure term that satisfies boundary and incompressibility constraints to find the final accelerations. The rigid body accelerations can be computed by creating a matrix $\mathbf{R}$ that is multiplied by a vector of the pressures that surround a rigid body. $\mathbf{R}$ can be formed by a series of matrix multiplications:

$$\mathbf{R} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_k \end{bmatrix} \begin{bmatrix} \mathbf{M}^{-1} & 0 \\ 0 & \mathbf{I}^{-1} \end{bmatrix} \begin{bmatrix} A_1\mathbf{b}_1^\mathsf{T} & \cdots & A_k\mathbf{b}_k^\mathsf{T} \end{bmatrix} \quad (6)$$

where $\mathbf{b}_i = \left[\mathbf{n}_i^\mathsf{T} \mid (\mathbf{r}_i \times \mathbf{n}_i)^\mathsf{T}\right]$, $\mathbf{n}_i$ is the normal of the ith face, $\mathbf{r}_i$ is the vector from the rigid objects center of mass to position of the ith face, and A_i is the area of that face. The rightmost matrix finds the net force-torque couple acting on a rigid body by summing up the contribution due to pressure forces acting on rigid body mesh faces. The force-torque couple is converted to a linear and angular acceleration of

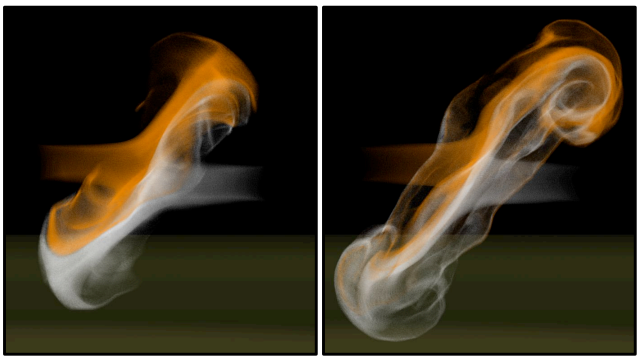

Figure 4: A comparison between uniform and selectively refined simulation meshes. Left: a frame from a simulation using approximately 43000 uniformly sized tetrahedra. Right: the same frame using approximately 32000 tetrahedra refined near areas of high vorticity and smoke density. The refined mesh preserves the fine detail in the velocity field and near the visible smoke, enhancing vortex action and natural movement. The runtimes of the two are equivalent.

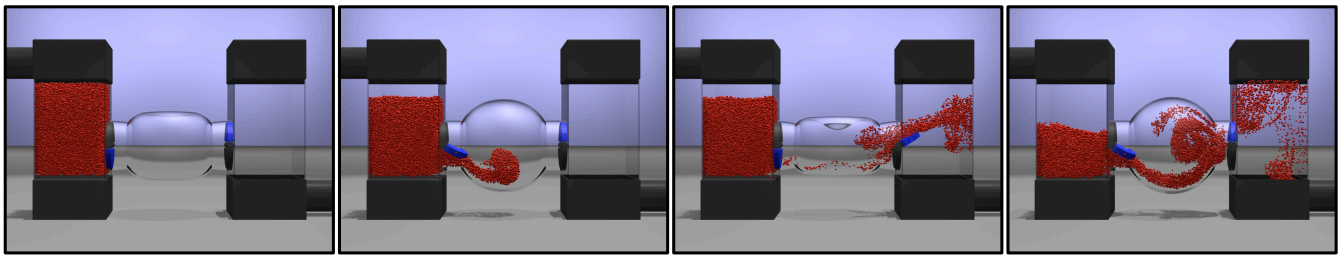

Figure 5: Red particles are transfered from the left tank to the right by squeezing and releasing the central bulb. The blue valves are coupled to the fluid simulation and prevent backflow.

the body by the middle (6×6) block matrix. $\mathbf{M}$ is a diagonal matrix with the mass of the rigid body on the diagonals and $\mathbf{I}$ is the inertia matrix. The leftmost matrix in the multiplication returns the acceleration of the fluid-rigid faces in the direction of the face normal due to the linear and angular acceleration of the rigid body. By construction, accelerations generated by this matrix behave rigidly.

Computing pressure accelerations of both the fluid and fluid-rigid faces can be expressed as a matrix $\mathbf{A}$ multiplied by a vector of all the pressures. A row of $\mathbf{A}$ that corresponds to a face with fluid on both sides contains the same entries as the standard gradient matrix multiplied by $-1/\rho$. A row of $\mathbf{A}$ that belongs to a face at the fluid-rigid interface has element values obtained from the corresponding row of $\mathbf{R}$. The elements of this row are placed at columns corresponding to the pressures that surround the rigid-body. With $\mathbf{A}$ built, mass conservation including two way coupling proceeds much in the same way as in the all-fluid case, with $\mathbf{A}$ replacing the role of the discrete gradient matrix. For a given vector of pressures, $\mathbf{p}$, the intermediate velocity field, $\mathbf{z}^*$, is accelerated to the end-of-step velocity, $\mathbf{z}$, by $\mathbf{z} = \mathbf{z}^* + \Delta t\mathbf{Ap}$. For the fluid faces, $\mathbf{z}^*$ is found by applying all terms of Equation (1) except the pressure term. For the fluid-rigid faces, $\mathbf{z}^*$ is found using a rigid body simulator without pressure forces applied. To find a particular pressure that accelerates $\mathbf{z}^*$ such that $\mathbf{z}$ is divergence free we solve the linear system

$$\Delta t\mathbf{DAp} = -\mathbf{Dz}^*. \tag{7}$$

This linear system can be solved efficiently using PCG since the the matrix $\mathbf{DA}$, which replaces the discrete Laplacian from the all fluid case, is also a positive-definite symmetric matrix.

Using the same machinery, we can also interact with constrained rigid bodies. This simply requires finding an $\mathbf{R}$ matrix that correctly computes face accelerations due to pressure. For example, one could easily alter $\mathbf{R}$ such that the body was constrained to just rotate about the origin by replacing $\mathbf{b}_i$ in Equation (6) with $\mathbf{b}'_i = \left[(\mathbf{r}_i \times \mathbf{n}_i)^\mathsf{T}\right]$ and using only the $\mathbf{I}^{-1}$ block for the center matrix. This idea could be extended further to include even articulated bodies.

4 Results and Discussion

We implemented the method described above in MATLAB[1] and C, making use of Pyramid [Jonathan Shewchuck, personal communication] for Delaunay triangulation and PIXIE[2] for all renderings. Typical simulation times for meshes with 100,000 tetrahedra were about 1 minute per frame. Table 1 compares remeshing and simulation times for several of the examples presented in this paper.

The images in Figure 1 show smoke in a tank mixed by the scripted motion of a paddle. Refinement of the simulation mesh near the paddle ensures good conformance to its curved surfaces that produce interesting vortex effects in the smoke.

[1] `http://www.mathworks.com`

[2] `http://sourceforge.net/projects/pixie`

	Remeshing time per frame (mean)	Total time per frame (mean)	Percent remeshing
Figure 1	13.2 sec	64.8 sec	20.3%
Figure 5	8.33 sec	44.5 sec	18.7%
Figure 6	5.76 sec	35.8 sec	16.1%
Figure 7	313 sec	796 sec	39.3%

Table 1: A comparison of remeshing and simulation time for selected examples.

In Figure 5, a pump transfers particles from the left tank to the right tank as the bulb in the middle is squeezed and released. The blue valves on either side of the bulb prevent backflow. The motion of these valves is not scripted. Instead, they are modeled as rigid bodies constrained to rotate about an axis and their motion is caused by two-way interaction with the fluid.

Figure 6 demonstrates the two-way interaction of the Stanford bunny with smoke cannons. On the left is a lighter bunny which is tossed about by the force of the cannons and also affects the motion of the smoke. On the right is a heavier bunny that drops quickly to the ground.

In Figure 7, smoke moves through an array of obstacles in a higher resolution mesh of over 500,000 tetradra. Although quality of the mesh elements does not suffer at this level of refinement, the proportion of time spent meshing increases to 39.3%. The motion of the smoke at the higher resolution is more lively and exhibits more fine-scale detail. A vorticity enhancement method, such as those in [Fedkiw et al., 2001] and [Selle et al., 2005] could be used to further enhance the fluid motion but we do not find such enhancement necessary and so have not implemented it.

We have presented a system for performing fluid animation using unstructured tetrahedral domains that can change arbitrarily at each time step. Although our current implementation models completely fluid-filled domains, we believe it would be well-suited for use with surface tracking techniques for liquid simulation.

Acknowledgments

We thank the other members of the Berkeley Graphics Group for their helpful criticism and comments. This work was supported in part by California MICRO 04-066 and 05-044, and by generous support from Apple Computer, Pixar Animation Studios, Autodesk, Intel Corporation, Sony Computer Entertainment America, and the Alfred P. Sloan Foundation. Klingner and Feldman were supported by NSF Graduate Fellowships.

References

Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M. 2005. Variational tetrahedral meshing. In *the Proceedings of ACM SIGGRAPH 2005*, 617–625.

Botta, N., and Hempel, D. 1996. A finite volume projection method for the numerical solution of the incompressible navier-stokes equations on triangular grids. *First International Symposium on Finite Volumes for Complex Applications*, 15–18 (July), 355–363.

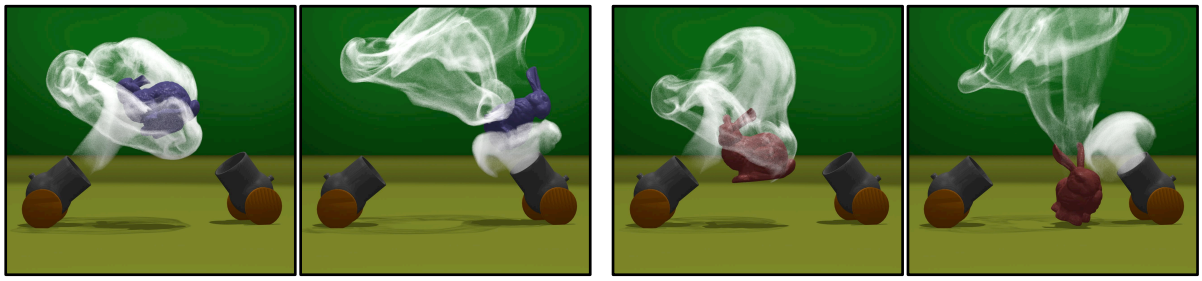

Figure 6: Cannons fire smoke at a light (left) and heavy (right) bunny.

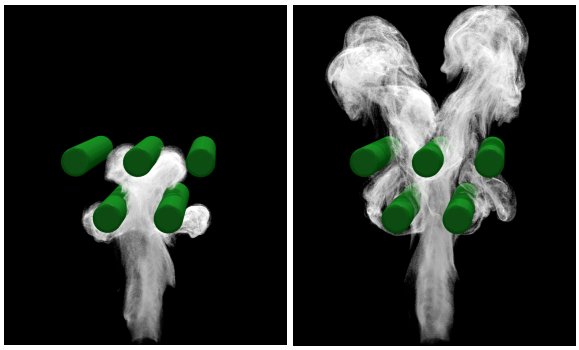

Figure 7: A high-resolution simulation of a jet of smoke moving through a set of obstacles.

Cani, M.-P., and Desbrun, M. 1997. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics 3*, 1 (Jan.), 39–50.

Carlson, M., Mucha, P. J., Van Horn III, R. B., and Turk, G. 2002. Melting and flowing. In *the ACM SIGGRAPH 2002 Symposium on Computer Animation*, 167–174.

Carlson, M., Mucha, P. J., and Turk, G. 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. In *the Proceedings of ACM SIGGRAPH 2004*, 377–384.

Desbrun, M., and Cani, M.-P. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation 1996*, 61–76.

Donea, J., Fasoli-Stella, P., and Giuliani, S. 1977. Lagrangian and eulerian finite element techniques for transient fluid-structure interaction problems. In *Trans. 4th SMIRT Conf.*

Donea, J., Huerta, A., Ponthot, J.-P., and Rodríguez-Ferran, A. 2004. *The Encyclopedia of Computational Mechanics*. John Wiley & Sons Inc., New York.

Elcott, S., Tong, Y., Kanso, E., Schröder, P., and Desbrun, M. 2005. Discrete, circulation-preserving, and stable simplicial fluids. Preprint, Caltech.

Enright, D. P., Marschner, S. R., and Fedkiw, R. P. 2002. Animation and rendering of complex water surfaces. In *the 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.

Feldman, B. E., O'Brien, J. F., and Arikan, O. 2003. Animating suspended particle explosions. In *the Proceedings of ACM SIGGRAPH 2003*, 708–715.

Feldman, B. E., O'Brien, J. F., and Klingner, B. M. 2005. Animating gases with hybrid meshes. In *Proceedings of ACM SIGGRAPH 2005*.

Feldman, B. E., O'Brien, J. F., Klingner, B. M., and Goktekin, T. G. 2005. Fluids in deforming meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005*.

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 *Graphics Interface 1996*, 204–212.

Foster, N., and Metaxas, D. 1997. Modeling the motion of a hot, turbulent gas. In *the Proceedings of ACM SIGGRAPH 97*, 181–188.

Goktekin, T. G., Bargteil, A. W., and O'Brien, J. F. 2004. A method for animating viscoelastic fluids. In *the Proceedings of ACM SIGGRAPH 2004*, 463–468.

Guendelman, E., Selle, A., Losasso, F., and Fedkiw, R. 2005. Coupling water and smoke to thin deformable and rigid shells. In *the Proceedings of ACM SIGGRAPH 2005*, 973–981.

Harlow, F., and Welch, J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids 8*, 2182–2189.

Hirt, C., Amsden, A., and Cook, J. 1974. An arbitrary lagrangian-eulerian computing method for all flow speeds. *Journal of Computational Physics 14*, 227–253.

Ju, T., Schaefer, S., Warren, J., and Desbrun, M. 2005. A geometric construction of coordinates for convex polyhedra using polar duals. In *Eurographics Symposium on Geometry Processing 2005*, 181 – 186.

Losasso, F., Gibou, F., and Fedkiw, R. 2004. Simulating water and smoke with an octree data structure. In *the Proceedings of ACM SIGGRAPH 2004*, 457–462.

Müller, M., Charypar, D., and Gross, M. 2003. Particle-based fluid simulation for interactive applications. In *the 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 *the ACM SIGGRAPH 2004 Symposium on Computer Animation*, 141–151.

Owen, S. J. 1998. A survey of unstructured mesh generation technology. In *the 7th International Meshing Roundtable*, 239–267.

Premože, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R. 2003. Particle-based simulation of fluids. *Computer Graphics Forum 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 *the ACM SIGGRAPH 2004 Symposium on Computer Animation*, 193–202.

Selle, A., Rasmussen, N., and Fedkiw, R. 2005. A vortex particle method for smoke, water, and explosions. In *the Proceedings of ACM SIGGRAPH 2005*, 910–914.

Shah, M., Cohen, J., Patel, S., Lee, P., and Pighin, F. 2004. Extended galilean invariance for adaptive fluid simulation. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 13 – 221.

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 *Graphics Interface 99*, 203–210.

Teng, S.-H., and Wong, C. W. 2000. Unstructured mesh generation: Theory, practice, and perspectives. *International journal of computational geometry applications 10*, 3, 227–266.

Terzopoulos, D., Platt, J., and Fleischer, K. 1989. Heating and melting deformable models (from goop to glop). In *Graphics Interface 1989*, 219–226.

Warren, J., Schaefer, S., Hirani, A. N., and Desbrun, M. 2004. Barycentric coordinates for convex sets. To appear in Advances in Computational and Applied Mathematics.

Yngve, G. D., O'Brien, J. F., and Hodgins, J. K. 2000. Animating explosions. In *the Proceedings of ACM SIGGRAPH 2000*, 29–36.