Animating Gases with Hybrid Meshes

Bryan E. Feldman James F. O'Brien Bryan M. Klingner

University of California, Berkeley

Abstract

This paper presents a method for animating gases on unstructured tetrahedral meshes to efficiently model the interaction of fluids with irregularly shaped obstacles. Because our discretization scheme parallels that of the standard staggered grid mesh, we are able to combine tetrahedral cells with regular hexahedral cells in a single mesh. This hybrid mesh offers both accuracy near obstacles and efficiency in open regions.

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

Over the last few years, simulation-based methods for animating fluids have developed to a mature state where production-quality results can be reliably obtained using both commercial and proprietary systems. The majority of these systems perform computations on regular hexahedral meshes using a standard staggered-grid discretization scheme that allows high-quality results to be produced using reasonable amounts of computation. Unfortunately, with grid-based methods, boundaries typically must be represented in a voxelized fashion, and as a result animating fluids in irregularly shaped domains can be awkward.

In this paper we describe a fluid simulation method for use on unstructured tetrahedral meshes that can be made to conform to arbitrary polygonal boundaries so that fluids may easily be modeled over irregularly shaped domains. The discretization scheme we employ for tetrahedra is a variation of the staggered scheme commonly used for regular hexahedral grids, and it allows natural enforcement of essential boundary conditions. Furthermore, because our tetrahedral discretization is compatible with the standard staggered-grid scheme, we can combine unstructured tetrahedral regions and regular hexahedral grids to efficiently cover a single simulation domain. This hybrid approach allows easy conformance to polygonal boundaries while still retaining a regular grid's efficiency over large open regions.

We have implemented this method and tested its performance on a variety of scenarios, such as the one shown in Figure 1, that involve fluids completely filling an irregularly shaped simulation domain. For simple, rectilinear domains, the results generated using our hybrid method are comparable, both in visual quality and computational efficiency, to results obtained from grid-based methods, but our method can also conform to complex boundaries, and in many contexts this facility can be quite useful.

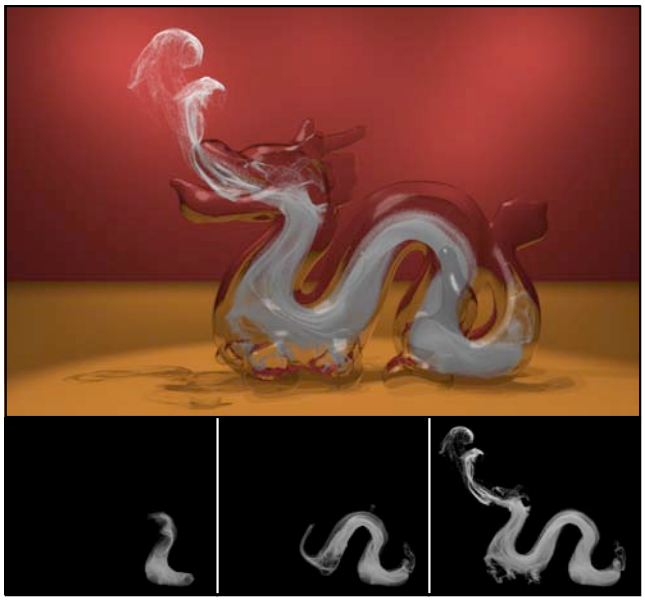

Figure 1: A smoke simulation on a tetrahedral mesh inside the Stanford Dragon model. A small region in the dragon's mouth is open so that smoke can flow out around its jaw into the surrounding volume.

2 Related Work

A substantial amount of work in the field of computer graphics has addressed the problem of realistically animating the behavior of fluids. Many of the resulting methods make use of a spatial discretization on regular hexahedral grids, and some recent examples include [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], [Goktekin et al., 2004], and [Carlson et al., 2004]. In general, these techniques have progressed to the point where some fluid phenomena can be modeled well enough that a naïve viewer may have difficulty distinguishing between real and simulated footage.

The most commonly used discretization scheme for regular hexahedral grids is the well known staggered grid method that was originally developed by [Harlow and Welch, 1965]. By storing velocity components and pressure on mutually staggered grids, it elegantly avoids the problems that plague methods which store collocated pressure and velocity val-

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

ACM 0730-0301/05/0700-0904

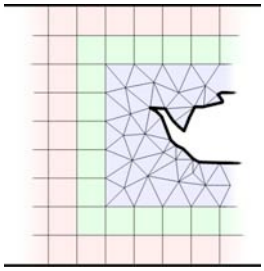

Figure 2: This two-dimensional diagram illustrates how regions of unstructured mesh can be connected to regularly gridded regions.

ues. The method also provides an easy and natural method for imposing both open and closed boundary conditions over surfaces that align with the grid-cell faces.

Recently, [Losasso et al., 2004] described an octree-based method that retains many of the advantages of regular staggered grids while also providing the additional advantage that simulation detail may be focused only in desired regions. A powerful application of this method is for tracking moving boundaries (free surfaces) at high resolution. To some extent, octrees also provide a method for dealing with the fixed boundaries of an irregular domain. However, forced octree refinement near fixed boundaries may not always be desirable.

While the above fluid methods make use of Eulerian grids on regular domains, other meshless methods use Lagrangian particles and can operate easily with irregular boundaries. Some examples of these methods include [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 methods have not yet demonstrated the same level of realism as current grid-based methods, but this situation is likely temporary as researchers continue to develop and adapt these simulation methods for graphics use.

The differential operators used in formulating fluid flow problems have been discretized for triangle and tetrahedral meshes in [Tong et al., 2003]. They used the method to perform decomposition and manipulation tasks on already computed vector fields. Building on that work, [Shi and Yu, 2004] showed that those differential operators could actually be used to simulate fluids on triangular meshes. Additionally, [Botta and Hempel, 1996] describe an equivalent triangular discretization for simulating fluids. However, they found the scheme less than ideal because of its difficulty with boundary constraints. They addressed the problem to some extent by adding extra degrees of freedom at boundary nodes.

A different discretization scheme for triangle meshes appears in [Rida et al., 1997]. Similar to the one presented in this paper, their scheme adapts the staggered method for regular grids to triangular meshes. However, in order to obtain second-order accuracy they adopted the restriction that the triangles in a mesh must contain their own circumcenter. While that restriction is somewhat reasonable for triangle meshes, it is not at all practical for tetrahedral ones because very few meshes could meet the requirement.

Concurrently with our work, [Elcott et al., 2005] have also developed a staggered tetrahedral discretization scheme nearly identical to ours. The fluid quantities are stored in the same manner, and they provide a different derivation of equivalent derivative operators. Two key differences are that they use a different interpolation method and do not discuss

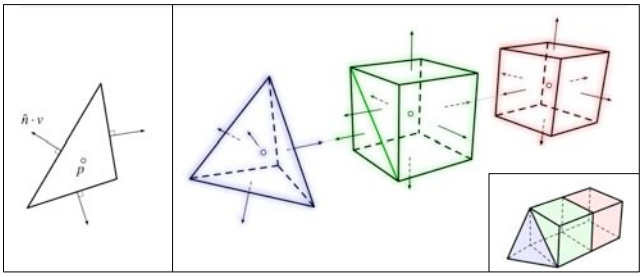

Figure 3: Storage scheme for velocity components and pressure. The 2D diagram illustrates how scalars, such as pressure, are stored in cell centers, while velocity is stored by recording only the normal component at the center of each cell face. The 3D diagram shows the scheme applied to tetrahedral (blue) and regular hexahedral (red) cells. The appropriate faces of transition (green) cells are split to allow them to interface with the triangular faces of tetrahedra. For clarity, the second tetrahedron that would adjoin the transition cell has not been shown.

combining tetrahedral and regular hexahedral girds within a single mesh. They do, however, provide an interesting circulation-based alternative to standard semi-Lagrangian advection.

3 Methods

The techniques we describe here provide a method for animating the behavior of fluids that completely fill a domain tiled by a combination of tetrahedra and axis-aligned, regular hexahedra. The details of and applications for this method are largely the same as for the method described in [Fedkiw et al., 2001]. The key distinction is that by including tetrahedral cells we can easily conform to complex domain boundaries.

While simulation on tetrahedral meshes provides substantial flexibility for matching boundaries, it does so at a moderate computational cost. To avoid paying this cost unnecessarily, we continue to use regular grids over large open areas. The resulting domain meshes are hybrid ones with regular grids smoothly joined to irregular tetrahedra. A two-dimensional diagram shown in Figure 2 illustrates the general concept. However, in three dimensions, we will see that the grid cells immediately adjacent to tetrahedra require special treatment.

The fluid's motion is governed according to the standard 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 solve this system using operator splitting as described by [Stam, 1999] so that we have three distinct steps: *add forces* which computes accelerations due to buoyancy or other forces acting on the fluid, *velocity advection* which uses a semi-Lagrangian advection scheme to account for the motion of the fluid, and *pressure correction* which enforces incompressibility by solving for an appropriate pressure field.

Additionally, for rendering purposes we advect a large number of massless tracker particles.

3.1 Discretization

As shown in Figure 2, the meshes we use decompose the simulation domain into regions occupied by regular, axis-aligned hexahedral cells and regions occupied by an unstructured tetrahedral mesh. Hexahedral cells adjacent to tetrahedra are transition cells that we treat slightly differently from other hexahedral cells.

Within the hexahedral region our storage scheme exactly follows the well established standard staggered grid method of [Harlow and Welch, 1965]. Scalar values, such as pressure, are stored at the center of each cell. Velocity is stored by placing at the center of each face the normal velocity component of the fluid at that location.

For tetrahedra we use essentially the same scheme: pressure in the centers and normal velocity on faces. However, the tetrahedral faces are not axis aligned or mutually orthogonal. As a result, differential operators and velocity interpolation require special treatment.

To link tetrahedral and grid regions, we designate the hexahedral cells adjacent to tetrahedral cells as transition cells. The adjacent tetrahedra share vertices with the transitions cells, and the appropriate faces of the transition cells are split so that they are compatible with the adjacent tetrahedra. (See Figure 3.) For interpolation purposes, which we discuss in Section 3.5, hexahedral cells sharing an edge with a tetrahedral cell are also treated as transition cells.

Computing the pressure correction step requires a divergence operator that can be applied to the velocity field and a gradient operator that can be applied to the pressure field. As with regular staggered grids, the divergence values should be computed at locations collocated with pressure storage, and gradient components should be computed corresponding to where matching velocity components are stored.

We form the divergence operator for all types of cells, (hexahedral, tetrahedral, and transition) by discretizing the divergence theorem, $\int_V (\nabla \cdot \mathbf{u}) \mathrm{d}V = \int_S (\hat{\mathbf{n}} \cdot \mathbf{u}) \mathrm{d}S$, assuming that the underlying velocity field is piecewise linear on the mesh. This construction, borrowed from the Finite Volume Method, yields the equation:

$$(\nabla \cdot \mathbf{u})_j = \frac{1}{V_j} \sum_{\mathrm{face}_i \in \mathrm{cell}_j} (\hat{\mathbf{n}}_i \cdot \mathbf{u}_i) A_i \qquad (3)$$

where the V_j is the cell's volume, A_i a face's area, and the term $(\hat{\mathbf{n}}_i \cdot \mathbf{u}_i)$ is the velocity component stored at a face's center. For regular hexahedral cells, this formula reduces to exactly the one from [Harlow and Welch, 1965]. Transition cells behave as if they were regular hexahedral cells whose transition-face velocity was the average of the split values.

For pressure gradients, we do not require the entire gradient at each face center, only the component of the gradient normal to the face. We compute this component by taking the difference in pressure values for the two cells sharing a face and dividing by the distance between the two cell centers measured perpendicular to the face. For regularly shaped hexahedral or transition cell of width h, the perpendicular distance from a face to the cell's center is simply $1/2h$. For tetrahedra, the perpendicular distance between the center of tetrahedron j and its i'th face is $3V_j/4A_i$. In summary, the gradient component across face i between cells a and b is given by:

$$(\hat{\mathbf{n}} \cdot \nabla p)_i = \frac{p_b - p_a}{(k_b + k_a)} \quad k_j = \begin{cases} 1/2h, & j \text{ is hex} \\ 1/2h, & j \text{ is tran} \\ 3V_j/4A_i, & j \text{ is tet} \end{cases} . \qquad (4)$$

Note that care should be taken to maintain proper signing. The above assumes $\hat{\mathbf{n}}$ is the normal at face i directed outward from cell a.

These divergence and gradient operators have several nice properties. In particular, they are simple, have small support, are cheap to compute, and like regular staggered-grid methods, they do not lead to interleaved, decoupled solutions during pressure correction projection. They also generate symmetric projection matrices, as we describe in Section 3.4. The equivalent derivative operators used in both [Elcott et al., 2005] and [Losasso et al., 2004] share all these useful properties, and we refer the reader to those papers for alternative derivations of those operators.

3.2 Accelerations due to Body Forces

Body forces resulting from phenomena such as gravity or techniques such as vorticity confinement act over the volume of the fluid. When working with regular grids, the acceleration on a face will be proportional to the average of the forces acting on adjacent cells. However, with cells of differing sizes care should be taken to prevent non-conservative behavior where small cells neighbor large ones. Luckily, the appropriate integrals reduce to simple volume-weighted averaging. The normal acceleration across face i between cells a and b is given by:

$$(\hat{\mathbf{n}} \cdot \mathbf{a})_i = \hat{\mathbf{n}} \cdot \frac{V_a \mathbf{f}_a + V_b \mathbf{f}_b}{\rho(V_a + V_b)} . \qquad (5)$$

3.3 Semi-Lagrangian Integration

A detailed description of semi-Lagrangian integration appears in [Stam, 1999]. Briefly, one determines the velocity at a point $\mathbf{x}$ at time $t + \Delta t$ by tracing backwards from $\mathbf{x}$ in the current flow field to find a point $\mathbf{x}'$. The velocity of $\mathbf{x}$ at $t + \Delta t$ is then set to the value found at $\mathbf{x}'$. An important consideration is that the method used to determine the velocity at $\mathbf{x}'$ must not introduce unnecessary smoothing or else, over the course of multiple integration steps, the simulated result will suffer from undesirable numerical diffusion. We address this issue in Section 3.5.

3.4 Mass Conservation

The intermediate velocity field, $\mathbf{u}^*$, produced by the advection step, in general, will not be mass conserving. To make it so, we enforce a volume-scaled divergence-free condition, $V_j (\nabla \cdot \mathbf{u})_j = 0$ for each cell by computing a correcting pressure field.

Let $\mathbf{z}$ be the system-length vector containing the normal velocities at each face in the mesh, and let $\mathbf{D}$ be the matrix that encodes Equation (3) such that $\mathbf{Dz}$ yields a vector containing the resulting divergence in each cell. However, because we require volume-scaled divergence, we modify $\mathbf{D}$ by omitting the $1/V_i$ scale factor from Equation (3). Similarly, let $\mathbf{p}$ be the vector of pressures in each cell, and $\mathbf{G}$ encodes Equation (4) such that $\mathbf{Gp}$ yields a vector containing the normal gradient component at each face. Combining Equations (2) and (1) with the above matrix operators gives us the discrete form of Poisson's Equation that we use to solve for $\mathbf{p}$:

$$\mathbf{DGp} = \frac{\rho}{\Delta t} \mathbf{Dz}^* \qquad (6)$$

where $\mathbf{z}^*$ is the vector of uncorrected face-normal velocities, and the corrected velocities are given by $\mathbf{z} = \mathbf{z}^* - (\Delta t/\rho)\mathbf{Gp}$. Equation (6) is a symmetric, positive-definite linear system that can be efficiently solved for $\mathbf{p}$ using a preconditioned conjugate-gradient solver.

One of the attractive features of the discretization we use is that boundary conditions are particularly easy to enforce. To make a face behave as an open boundary, the gradient operator (Equation (4)) uses an ambient pressure of 0 on the side of the boundary face that is outside the mesh and places it at a distance equal to that of the distance to the pressure that is in the mesh. Closed boundary conditions are enforced by setting the pressure outside the mesh, opposite the closed face, to be equal to the pressure of the tetrahedron that is in the mesh.

3.5 Interpolation

Although the storage scheme we use produces nice, simple formulae for computing divergence and gradients, interpolating the velocity field turns out to be somewhat awkward. Other researchers who have used similar staggered schemes on triangles, for example [Rida et al., 1997], have not addressed the issue because they had no need to interpolate the velocity field. However, the semi-Lagrangian integration scheme we use does require interpolated velocities. Additionally, we use many tracker particles for rendering purposes, and advecting these particles also requires interpolated velocities.

We make use of three different interpolation methods. For regular hexahedral cells we always use standard tri-linear interpolation. However, for tetrahedral and transition cells we use different methods depending on what the interpolated values will be used for. For computing the interpolated velocity values that semi-Lagrangian advection will carry forward for the next time step, we use a variation of moving least-squares interpolation with linear basis functions that is accurate but somewhat costly. For advecting particles and for backward tracing during the semi-Lagrangian step, we use a faster, less accurate method. The faster method first uses moving least-squares interpolation with constant basis functions to compute velocities at the mesh vertices, and then does standard linear interpolation within tetrahedra.

The moving least-squares interpolation we use is part of a family of interpolation methods commonly used for certain "meshless" simulation techniques. A comprehensive discussion can be found in [Belytschko et al., 1996], and the particular method we use has has also been used in graphics for implicit surface construction. (For example, see [Ohtake et al., 2003] and particularly [Shen et al., 2004].)

In our case, we only have velocity normal components stored at each face, so a constant least-squares velocity fit would take the form

$$\begin{bmatrix} \hat{\mathbf{n}}_1^\mathsf{T} \\ \hat{\mathbf{n}}_2^\mathsf{T} \\ \vdots \\ \hat{\mathbf{n}}_k^\mathsf{T} \end{bmatrix} \mathbf{u} = \begin{bmatrix} (\hat{\mathbf{n}} \cdot \mathbf{u})_1 \\ (\hat{\mathbf{n}} \cdot \mathbf{u})_2 \\ \vdots \\ (\hat{\mathbf{n}} \cdot \mathbf{u})_k \end{bmatrix} \tag{7}$$

where $\hat{\mathbf{n}}_i$ is the face normal at face i and $(\hat{\mathbf{n}} \cdot \mathbf{u})_i$ the corresponding normal velocity component. If we name the matrix used above, we can instead write more compactly:

$$\mathbf{N}\mathbf{u} = \mathbf{z} \ . \tag{8}$$

Equation (8) can be transformed into an interpolation method by introducing the diagonal moving-weight matrix, $\mathbf{W}(\mathbf{x})$, defined by

$$w_{ii}(\mathbf{x}) = \frac{1}{r_i^2 + \epsilon^2} \tag{9}$$

where r_i is the distance between $\mathbf{x}$ and the center of face i. We can then interpolate the velocity at an arbitrary point $\mathbf{x}$

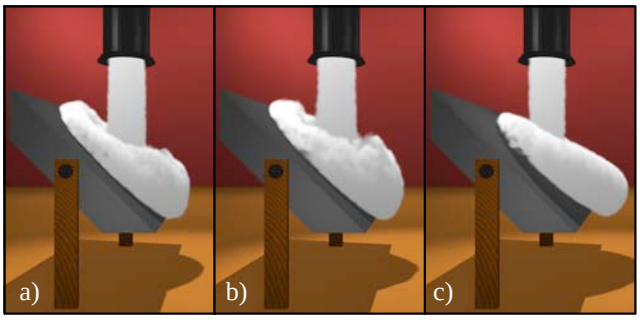

Figure 4: A comparison of a jet shooting downward onto an inclined ramp a) using an all-tetrahedral mesh b) using a hybrid mesh c) using all regular hexahedral cells.

Grid Type	Number Tetrahedra	Number Hexahedra	Simulation time per frame (mean)
a) All Tet.	85720	0	48.22 sec
b) Hybrid	34839	85155	20.97 sec
c) All Hex.	0	97707	12.68 sec

Table 1: A comparison of different meshes for the inclined ramp example.

by solving for $\mathbf{u}$ with the method of normal equations using the system:

$$\mathbf{W}(\mathbf{x})\mathbf{N}\mathbf{u} = \mathbf{W}(\mathbf{x})\mathbf{z} \ . \tag{10}$$

We construct a higher order moving least-squares interpolant by including linear basis functions. The unweighted least-squares fit in Equation (7) then takes the form:

$$\begin{bmatrix} \hat{\mathbf{n}}_1^\mathsf{T} & x_1\hat{\mathbf{n}}_1^\mathsf{T} & y_1\hat{\mathbf{n}}_1^\mathsf{T} & z_1\hat{\mathbf{n}}_1^\mathsf{T} \\ \hat{\mathbf{n}}_2^\mathsf{T} & x_2\hat{\mathbf{n}}_2^\mathsf{T} & y_2\hat{\mathbf{n}}_2^\mathsf{T} & z_2\hat{\mathbf{n}}_2^\mathsf{T} \\ & & \vdots & \\ \hat{\mathbf{n}}_k^\mathsf{T} & x_k\hat{\mathbf{n}}_k^\mathsf{T} & y_k\hat{\mathbf{n}}_k^\mathsf{T} & z_k\hat{\mathbf{n}}_k^\mathsf{T} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{bmatrix} = \begin{bmatrix} (\hat{\mathbf{n}} \cdot \mathbf{u})_1 \\ (\hat{\mathbf{n}} \cdot \mathbf{u})_2 \\ \vdots \\ (\hat{\mathbf{n}} \cdot \mathbf{u})_k \end{bmatrix} \tag{11}$$

where the scalars x_i, y_i, and z_i denote the position coordinates of the i'th face center relative to the evaluation point $\mathbf{x}$, and the $\mathbf{u}_0$, $\mathbf{u}_x$, $\mathbf{u}_y$, and $\mathbf{u}_z$, are the coefficients of the linear polynomial fit. The procedure for applying the moving weights to Equation (11) follows as described above. Because the evaluation point, $\mathbf{x}$, is used as the origin during the linear fit, we have $\mathbf{u}(\mathbf{x}) = \mathbf{u}_0$.

With either constant or linear fits some care must be taken when selecting ϵ. Large values introduce undesirable smoothing, while too small values can cause conditioning problems. We have found that adjusting ϵ so that it is 0.03 times the average edge-length of the cell containing the evaluation point works well. Further discussion of moving least-squares interpolation and fast evaluation techniques can be found in [Shen et al., 2004].

At each time step, prior to semi-Lagrangian advection, we use the constant-basis moving least-squares interpolation to estimate velocities at all vertices of the tetrahedral and hybrid portions of the mesh. Only the faces adjacent to a given vertex are used during the fit. Once we have vertex velocities we can quickly compute a slightly smoothed velocity estimate anywhere using linear interpolation. This interpolated velocity field is used for the backward tracing part of semi-Lagrangian advection. However, once the backward tracing has determined a point $\mathbf{x}'$ whose interpolated velocity will be carried forward and projected onto the mesh for the next timestep, we compute the velocity at that point using linear-basis moving least-squares interpolation of the original face velocity components in a two-ring neighborhood of $\mathbf{x}'$. The

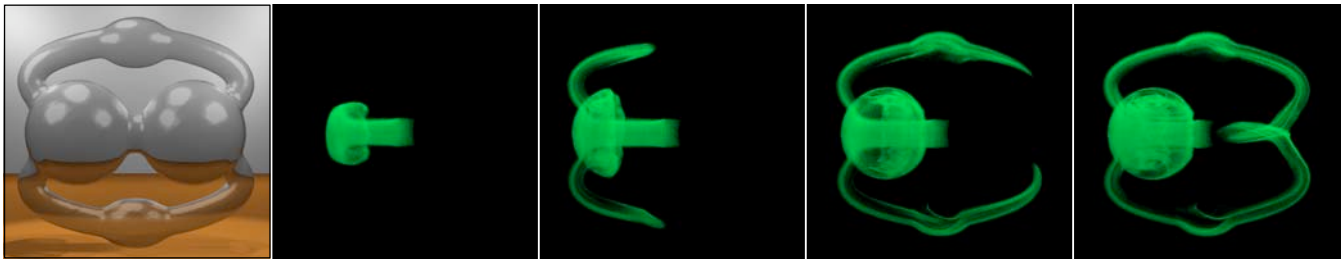

Figure 5: The leftmost image shows a hollow glass object. The remaining images show the progression of green smoke as it is injected into the center.

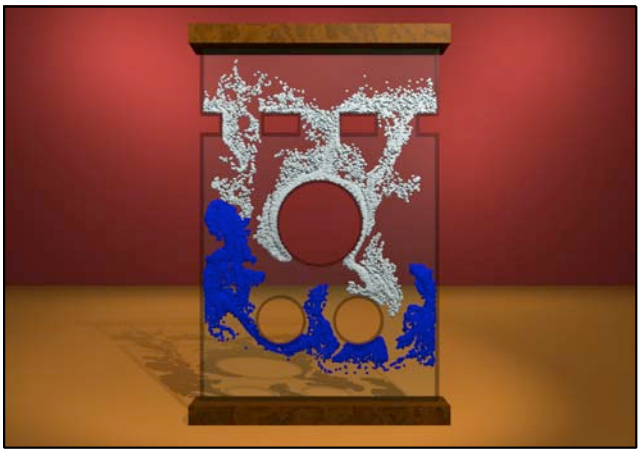

Figure 6: A desk toy filled with high-density fluid at the top and low-density fluid at the bottom. The fluid between is neutrally dense. Massless particles are passively advected in the flow.

process of computing vertex velocities using constant-basis moving least-squares interpolation is repeated prior to particle advection, and the particles are then advected using linearly interpolated vertex velocities. Comparisons of results rendered using our multiple interpolation method approach exhibit no visual difference from results computed using exclusively linear-basis moving least-squares for all interpolation.

4 Results and Discussion

We have implemented the method described above in MATLAB[1] and used it to generate several example animations. The tetrahedral portions of the meshes were generated using NETGEN[2], and our final images were rendered using PIXIE[3]. Simulation times varied from approximately four hours for the dragon example shown in Figure 1 to less than an hour for the corkscrew in Figure 7. Each of the animations appear in the accompanying video.

In Figure 4 we compare three animations of the same scenario that were computed on different meshes. In each, a downward jet exhausts onto an inclined ramp. Figure 4a) shows a frame from the animation that uses a mesh consisting of all tetrahedra. Figure 4b) uses a hybrid mesh with tetrahedra near the ramp and regular cells filling the remaining open space. Finally, Figure 4c) uses a regular grid everywhere. The number of cells used and runtimes appear in Table 1. The behaviors of the tetrahedral and hybrid systems are quite similar. In both, the flow spreads naturally up the ramp. The behavior with the regular grid is less natural because the ramp must be represented as axis-aligned faces. Grid-based simulation in general suffers from poor behavior at curved or angled boundaries while tetrahedral meshes are naturally suited to these situations.

The images in Figure 1 show the results of an all-tetrahedral simulation where smoke is generated in the tail of the Stanford dragon model and forced out of an opening in its mouth. The smoke twists and rolls as it moves through the body due to the dragon's undulating cylindrical shape. A similar example appears in Figure 5. Smoke inside the closed shape circulates due to the action of a jet at the center. The smoke accelerates as it moves into the arms of the object and disperses where the flows from the two arms collide.

Figure 6 shows a simulation of an executive desk toy. The simulation is initialized with a region of high density at the top and a region of low density at the bottom. These regions are filled with massless tracker particles. The particles are passively advected as regions of high and low density mix in the neutrally dense fluid that fills the rest of the toy.

In Figure 7 a jet has been placed below a cylinder containing a corkscrew shape. The region inside and immediately around the cylinder contains tetrahedra, while the remainder of the scene is covered by a regular grid. The flow conforms well to the corkscrew's spiral and then disperses and rolls as it rises through the free space above.

We have presented a method for animating fluids using hybrid meshes containing both tetrahedral and regularly gridded regions. Using the technique we can efficiently model the behavior of fluids as they interact with irregular boundaries by tetrahedralizing the regions surrounding the boundaries. The computational expense incurred by a single tetrahedral cell is larger than that of a grid cell, but our hybrid approach allows us to retain the overall efficiency of grids while still benefiting from the flexibility of a tetrahedral mesh. Although our current implementation is limited to entirely fluid-filled domains by the lack of a free surface tracker, contouring methods such as [Bargteil et al., 2005] are quite flexible and should be easily adaptable for use with our technique.

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 03-067 and 04-066, and by generous support from Apple Computer, Pixar Animation Studios, Intel Corporation, Sony Computer Entertainment America, the Hellman Family Fund, and the Alfred P. Sloan Foundation.

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

[2] http://www.hpfem.jku.at/netgen

[3] http://sourceforge.net/projects/pixie

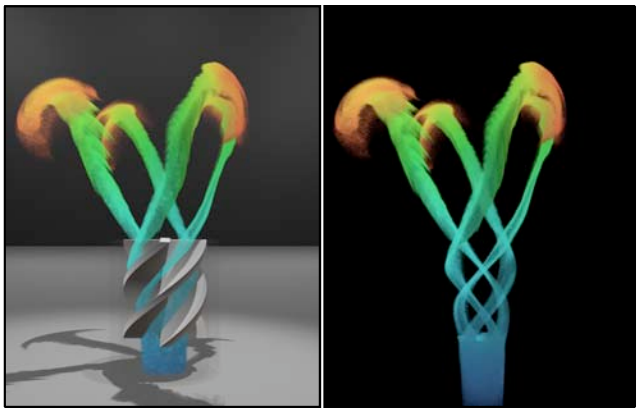

Figure 7: These images show the result of shooting colored smoke through a corkscrew cylinder. The image on the right shows the smoke with the rest of the scene removed.

References

Bargteil, A. W., Goktekin, T. G., O'Brien, J. F., and Strain, J. A. 2005. A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*. To appear.

Belytschko, T., Krongauz, Y., Organ, D., Fleming, M., and Krysl, P. 1996. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering 139*, 3–47. Special issue on meshless methods.

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.

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.

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.

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.

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.

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.

Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. 2003. Multi-level partition of unity implicits. In *the Proceedings of ACM SIGGRAPH 2003*, 463–470.

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.

Rida, S., McKenty, F., Meng, F., and Reggio, M. 1997. A straggered control volume scheme for unstructured triangular grids. *International Journal for Numerical Methods in Fluids 25*, 697–771.

Shen, C., O'Brien, J. F., and Shewchuk, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. In *the Proceedings of ACM SIGGRAPH 2004*, 896–904.

Shi, L., and Yu, Y. 2004. Inviscid and incompressible fluid simulation on triangle meshes. *Computer Animation and Virtual Worlds 15*, 3-4, 173–181.

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.

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

Tong, Y., Lombeyda, S., Hirani, A. N., and Desbrun, M. 2003. Discrete multiscale vector field decomposition. In *the Proceedings of ACM SIGGRAPH 2003*, 445–451.

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