Finite Volume Flow Simulations on Arbitrary Domains

Jeremy D. Wendt
University of North Carolina at Chapel Hill
jwendt@cs.unc.edu

William Baxter
OLM Digital, Inc.
baxter@olm.co.jp

Ipek Oguz
University of North Carolina at Chapel Hill
ipek@cs.unc.edu

Ming C. Lin
University of North Carolina at Chapel Hill
lin@cs.unc.edu

May 9, 2006

Manuscript correspondence is best sent via email to the addresses listed above. However, if postal mail or telephone communication is required, Jeremy can be reached at:

Jeremy D. Wendt
Department of Computer Science
Campus Box 3175, Sitterson Hall
UNC-Chapel Hill
Chapel Hill, NC 27599-3175 USA
Phone: (919) 962-1926

Abstract

We present a novel method for solving the incompressible Navier-Stokes equations that more accurately handles arbitrary boundary conditions and sharp geometric features in the fluid domain. It uses a space filling tetrahedral mesh, which can be created using many well known methods, to represent the fluid domain. Examples of the method's strengths are illustrated by free surface fluid simulations and smoke simulations of flows around objects with complex geometry.

Keywords: Fluid simulation, physically-based modeling, Navier-Stokes equations, Euler equations, finite volume method, stable fluids, vorticity confinement, smoke, water.

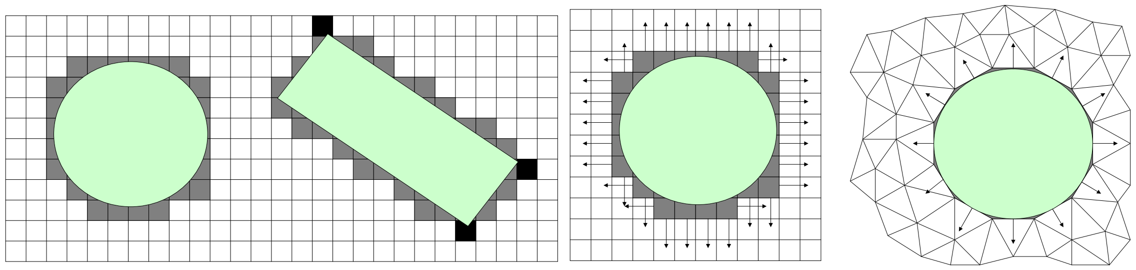

Figure 1: These images shows the boundary error introduced by sampling the actual boundary of immersed objects onto a fluid domain discretized as an axis aligned grid. The gray cells would typically serve as ghost cells; however, the darker cells are illegal and either need to be eliminated or have a neighbor added to them. In 3D, the special cases required to detect and eliminate invalid ghost cells become more numerous. This also shows the error in boundary normals introduced by a typical grid method (left) vs. ours (right).

1 Introduction

Recent advances in fluid simulation have made it possible to animate smoke and water with striking visual realism. These techniques have served as key components of both 3D animated and live action movies (e.g. *Shrek II* and *The Day After Tomorrow*). The techniques most widely presented and used in this context have been based on finite difference methods (FDMs) for solving the equations of fluid motion. However, a competing class of methods known as finite volume methods (FVMs) have also been studied extensively in computational sciences and engineering domains, and present an attractive alternative to FDMs because of their inherent ability to handle irregularly-shaped boundaries more accurately. However, previous FVM approaches have not offered the efficiency and unconditional stability demanded by visual simulation and special effects production.

FDMs work by approximating partial derivatives with differences derived from Taylor expansions on a discrete number of points in space. Grid-based FDMs have several strengths. First, implementing them is relatively straightforward, and when implemented using structured rectilinear grids, the positions and connectivity of grid cells need not be stored explicitly. This implicit connectivity and uniform grid spacing reduces memory use and simplifies computations.

However, FDMs also suffer from several problems. First, for uniform rectilinear grids, the fixed arrangement of grid cells in space implies that to resolve high complexity in one area, high resolution must be used in all areas, resulting in both increased computation and memory. Automatic mesh refinement (AMR) schemes, which have been used in engineering applications for years (see e.g. [1, 2]) do help, but at the cost of implementation complexity. Recently Losasso et al. [3] presented a refinement scheme based on an octree grid. These approaches allow adaptive refinement around areas where detail is most needed, but they incur an additional cost in memory usage over basic FDMs given a fixed number of grid cells. Furthermore, these schemes do not adequately address the next issue, non-physical grid effects.

Having all cells lie on a regular grid can lead to non-physical grid effects in the output data. A simple example is the anisotropy of space with the typical face-centered MAC grid [4, 5]. Numerical diffusion, for instance, is much lower for flows in the axis-aligned directions. This can result in spurious non-physical effects in the simulation output that depend on the orientation of the grid. This problem is reduced with the unstructured grid method we propose, since there are, in general, no directions preferred over others. Adaptive grid refinement schemes may also reduce this problem, but do not eliminate it, in part because of the next issue, accurate enforcement of boundary conditions.

Enforcing the proper no-slip or Neumann boundary condition on irregular boundaries with complex geometry is difficult with finite difference methods. In the fluid simulation methods of [5–10] the fluid domain is coarsely rasterized onto an axis aligned grid, and each cell is treated either as completely fluid or completely rigid (e.g. see Fig. 1). This $O(\Delta x)$ error in the actual boundary location can lead to noticeable artifacts, such as the fluid either penetrating the surface or "skirting around" the surface without touching it.

Although high resolution or axis-aligned refinement schemes [3] can reduce positional error to visual tolerance, the normals at the rasterized boundary still have $O(1)$ error. That is, they are only correct for axis-aligned boundaries, and changing Δx (refining the grid) does not reduce the error. Using the incorrect normal on the boundary can lead to artifacts in pressure calculations, especially when object boundaries are moving through the grid. These can lead to visual artifacts (e.g. see [11] Fig. 4)

Perhaps one of the most important motivations for this work is a step toward simulating complex interaction between fluids and other objects, either rigid or deformable, with *complex geometry, irregular boundaries, and thin features*. Some of the best known techniques for modeling deformable bodies and fractual mechanics include finite element methods (FEMs) based on tetrahedral meshes. The impetus of this research is to investigate alternative techniques that can perhaps better facilitate and more naturally handle the interface between fluids and FEM-based representations, commonly used in modeling deformable bodies, fractures, and other heterogeneous materials. Although we have not yet achieved our ultimate goal, the work presented here describes our first step in this direction.

Main Results: In this paper, we propose using a novel *finite volume* based technique for fluid simulation. Our technique alleviates the problems just discussed that arise when using grid-based methods. And, it also *differs* from existing work on FVM in applied mathematics and computational physics, since our focus is on fast and plausible visual simulation, not higher-order convergence regardless of the computational costs. Our approach offers the following advantages over existing grid-based methods:

- Efficiently handles boundary conditions on complex irregular domains without resorting to numerous special cases;

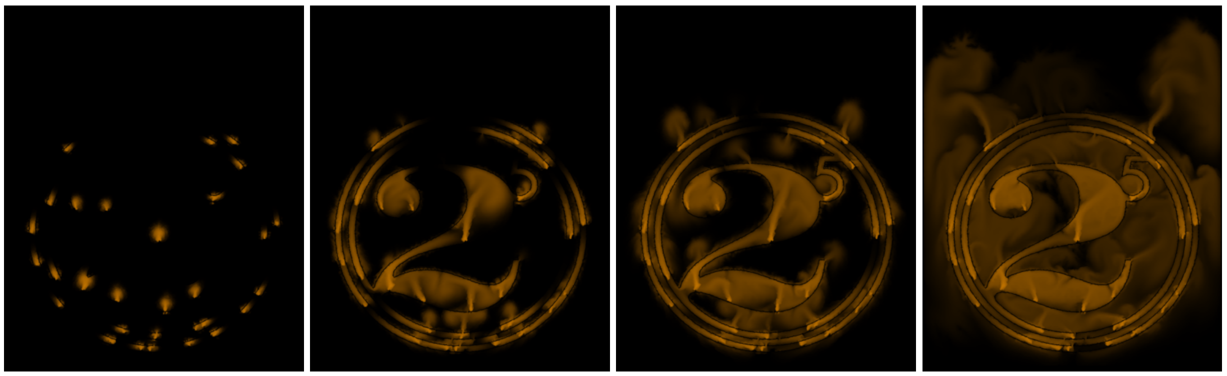

Figure 2: In this simulation, 2D smoke is controlled completely and easily using our thin boundary conditions. A semi-permeable membrane is simulated by passing a small percentage of the smoke difference across the boundary at each time step causing the smoke to "bleed". This simulation sequence appeared in the SIGGRAPH 2005 Animation Festival.

- Enables the use of polygonal models directly as flow boundaries without rasterization onto a fluid grid;
- Provides a natural way to control fluid flows.

This paper presents a technique for simulating fluid flows on arbitrarily shaped domains with regions of heterogeneous size and widely varying topology. We also present a method to control the fluid simulation using the inherent capabilities of our approach (see Figure 2). We demonstrate our implementation on challenging scenes of smoke and free-surface fluid flows around geometry with holes and fine geometric detail, which are difficult to simulate correctly with traditional grid-based methods.

Organization: The rest of the paper is organized as follows. In section 2, we briefly review the related work. We give a brief overview of finite volume methods in section 3 and our solution procedure in section 4. In section 5, we describe our implementation and demonstrate the results we achieve. Finally we conclude and suggest directions for future research in section 6.

2 Previous Work

Numerical methods for computational fluid dynamics have been actively investigated for decades. Recently there has been a growing interest in adapting some of these techniques for simulated fluid flows in computer graphics. We briefly survey some of the related work in this section and refer the readers for more detail to surveys in recent journal articles [12, 13].

Finite Difference Methods (FDMs): In computer graphics, Foster and Metaxas [6] were among the first to present the use of the full 3D Navier-Stokes differential equations for generating fluid animations. Their FDM tracks the free surface using the marker-and-cell approach [4]. The “stable fluids” method of Stam [7] introduced stable semi-Lagrangian advection combined with an implicit viscosity solver to arrive at a completely stable method, more amenable to use in animation. This method supported simple periodic, or axis-aligned external boundary conditions. Fedkiw et. al. [14] presented a method for simulating smoke with Euler equations, and also introduced vorticity confinement to reduce the spurious numerical damping of vorticity. Foster and Fedkiw [5] combined the surface-tracking particles [6] with level sets to more accurately track the surface of the fluid. Enright et al. [9] enhanced free surface tracking by developing a method they referred to as “the particle level set method”. Their modification is to place particles on both sides of the interface instead of just the interior to further reduce erosion of small surface features caused by advection techniques.

Losasso et al. [3] introduced octree refinement to the previously cited methods to allow simulations with effectively high resolutions at lower computational cost. Carlson et al. [10] presented techniques based on distributed Lagrange multipliers to ensure two-way coupling to animate the interplay between rigid bodies and fluid. Mihalef et. al. [15] also demonstrated a coupling of 3D fluid simulation with 2D fluid ”slices” in order to simulate thin features of breaking waves.

Each of these methods have used finite difference methods on axis-aligned grids to solve the Navier-Stokes equations. Consequently, all of the above methods had to rely on high-error approximations in order to capture the effect of a boundary that cuts through the grid at an angle.

Higher-Order Convergence with FDMs: Several methods have been developed to achieve higher order handling of boundaries with FDMs. A good example is the GENSMAC method [16]. The grid-based technique is based on the original MAC method [4], but using modified finite difference stencils on cut boundary cells to better approximate the actual derivatives on the boundary. Determining which stencil to apply for each possible cut configuration leads to numerous special cases, especially in 3D, making the code tedious to write and difficult to debug. The method is only demonstrated in 2D.

Mesh-free Methods: Although most recent applications of fluid simulation for animation have relied on FDMs, mesh-free methods have also been explored. Stam and Fiume [17] presented simulations based on a smoothed particle formulation of gas dynamics, and Desbrun and Gascuel [18] followed with a modified version of “Smoothed Particle Hydrodynamics” (SPH), developed originally for cosmological simulations. Stora et. al. [19] added temperature transport to their SPH in order to simulate and render realistic lava flows. Müller et al. [20] demonstrated 3D SPH simulations. Though particle methods have their advantages, boundary conditions are even more difficult to enforce properly using these techniques than with FDMs. Good overviews of SPH and other mesh-free methods can be found in [21, 22].

Flows on 2D Meshes: Stam [23], and Shi and Yu [24] simulated Navier-Stokes flows on 2D meshes. Stam's method requires the surface to be a regular quadrilateral mesh, while Shi and Yu's technique works on any triangulated mesh. Both focused on the goal of generating plausible 2D flows on surfaces embedded in 3D space. Both methods are modified finite difference techniques, and both are limited to 2D flows. In contrast, we present techniques for simulating planar 2D flows and volumetric 3D flows, but on irregular, non-uniform triangular or tetrahedral tessellations of the 2D or 3D space.

Finite Volume Formulations: In the computational physics literature many methods have also been proposed based on finite volume formulations. Finite volume techniques naturally lead to conservative methods that can preserve mass and other properties of fluids exactly to within machine precision, which has proven very valuable in the study of compressible fluids. LeVeque [25,26] discusses many such techniques for solution of hyperbolic PDEs on axis-aligned grids. Other researchers in fluid dynamics have presented finite volume methods for simulation of incompressible flows on unstructured grids composed of triangles or quadrilaterals in 2D [27–29], or tetrahedra and hexahedra in 3D [1]. These methods have focused on high order convergence regardless of the computational cost, rather than visually plausible results at costs acceptable for animation production, which is the goal of this paper.

Concurrent Work: Other researchers have explored finite volume methods for graphics applications concurrently with the research presented in this paper. Feldman et. al. [11, 30] present a method for creating and simulating on "hybrid" meshes. Their meshes use finite difference axis-aligned grids in open areas, and tetrahedra near boundaries. However, they only presented non-viscous Eulerian flows. Extending their work to handle viscous Navier-Stokes flows would be non-trivial. Elcott et al. also investigated a slightly different technique which aims at preserving circulation in fluid flows [31]. At publication time of this work, their work is still under submission, but a preprint is available on their website.

3 Overview

In this section, we describe the basic formulation of finite volume methods (FVMs).

3.1 Introduction to Finite Volume Methods

A finite volume method relies on the integral form of a PDE, instead of the differential form. Thus, instead of using finite difference stencils on a set of points, integrals are approximated over a space-filling set of volume primitives. Using finite differences, one solves the Navier-Stokes equations by directly discretizing the differential operators. In contrast, in a finite volume method one solves:

$$\frac{d}{dt}\int_{\Omega} \mathbf{u}\, d\Omega = \int_{\Omega} \left(-(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} - \nabla p + \mathbf{F}\right)\, d\Omega \tag{1}$$

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

by discretizing and approximating the integrals numerically. In the above, Ω is a space filling element (area in 2D, volume in 3D). The principle mathematical tool used to make this formulation practical is Stokes' theorem, which allows one to convert volume integrals into surface integrals. In particular, the divergence theorem and the curl theorem (special cases of Stokes' theorem) are useful.

The divergence theorem is used in both our 2D and 3D solvers:

$$\int_{\Omega} \nabla\cdot\mathbf{f}\, d\Omega \equiv \oint_{\Gamma} \mathbf{f}\cdot\mathbf{n}\, d\Gamma \tag{3}$$

$$\int_{\Omega} \nabla^2 f\, d\Omega \equiv \oint_{\Gamma} \nabla f\cdot\mathbf{n}\, d\Gamma \tag{4}$$

where $\mathbf{f}$ is an arbitrary vector field, f is a scalar field, Ω is defined as above, Γ is a closed boundary around Ω (a closed curve in 2D, a closed surface in 3D), $\oint$ is a boundary integral, $\mathbf{n}$ is a vector normal to the boundary curve or surface, and $d\Gamma$ is a surface element on the boundary.

For integrals of curls (required to implement vorticity confinement [14] in a finite volume context), we use the curl theorem in 2D and a variation of the divergence theorem in 3D:

$$\int_{\Omega} (\nabla\times\mathbf{f})\cdot d\Omega \equiv \oint_{\Gamma} \mathbf{f}\cdot d\Gamma \tag{5}$$

$$\int_{\Omega} (\nabla\times\mathbf{f})\, d\Omega \equiv \oint_{\Gamma} d\Gamma\times\mathbf{f}, \tag{6}$$

where Eq. 5 is used in 2D and Eq. 6 is used in 3D. Eq. 6 can be derived from the basic divergence theorem (Eq. 3) by substituting $\mathbf{c}\times\mathbf{f}$ in for $\mathbf{f}$, where $\mathbf{c}$ is a constant vector.

3.2 Grid Primitives

We define our unstructured finite volume grids using simplexes: triangles in 2D and tetrahedra in 3D. Both of these primitives are well supported by current graphics tools and applications, and are simple to work with. Unstructured grids can be created from other volume elements, or by combinations of several types of elements as in [29], but this adds to the complexity of the code and leads to more special cases.

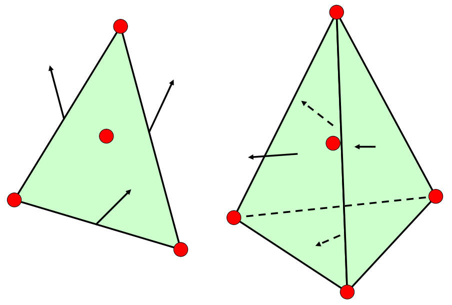

Figure 3: Our grid primitives: triangles in 2D, tetrahedra in 3D. Note that full velocity vectors are stored at each edge/face center. Also note that pressure values (denoted by dots) are stored at both cell centers and vertices.

We store pressure values at cell centers and vertices, while full velocity vectors are stored at edge centers in 2D, and face centers in 3D. Storing an unrestrained velocity vector (i.e. not restricted the the face normal direction), we are able to add diffusion to our simulator in a straighforward way, and remove some of the difficulties posed by advection discussed in [11]. Our standard grid cells are shown in Fig. 3. Scalar values such as density and temperature are stored at vertices in both 2D and 3D. When needed, a continuous scalar field is derived from these vertex values by linear interpolation.

Finite volume methods allow the freedom of placing grid geometry where desired, enabling non-axis-aligned boundaries that are better approximations of the underlying geometry. Furthermore, if the desired boundaries *are* polygonal models, as is often the case in visual applications, then the boundaries enforced can match those specified exactly. This is in contrast to existing FDM-based methods where the boundary is discretized using axis-aligned grids, leading to spurious interactions at the boundary.

In order to make use of the surface integral formulations obtained by application of Eqs. 3–6, the domain is partitioned into a set of control volumes. How these are defined depends upon which form of Stokes' theorem is being used and where the data values are stored (e.g. cell-centers vs. face-centers). Once the control volumes are defined, then the surface integrals can be discretized over the boundaries of those control volumes. For example, on a 2D triangle mesh with values $\mathbf{f}_i$ stored on triangle edges, the left hand side of Eq. 3 can be discretized as a summation over the triangle edges:

$$\sum_{i=1}^{3} (\mathbf{f}_i \cdot \mathbf{n}_i)\, \Delta\Gamma_i, \tag{7}$$

where $\Delta\Gamma_i$ is the length of the ith edge. In this case, the result is an approximation of the integral of the divergence of $\mathbf{f}$ over the triangle. If divided by the area of the triangle, this gives the triangle's average divergence per-unit area. This is in contrast to an FDM approach, which would give the divergence at a single point in space. Further descriptions of the control volumes used in our algorithm are given in Fig. 4.

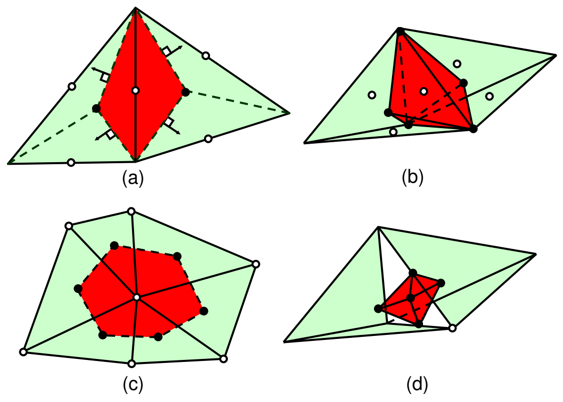

Figure 4: The control volumes associated with different data locations. Control volumes define the boundary over which the finite volume integrals will be evaluated. In different steps of the algorithm, control volumes are needed for cell-centered data, face-centered data, and vertex-centered data. The control volume for cell-centered data is simply the primitive itself. In defining other control volumes, dual graphs are used. (a) and (b) show the control volume for the boundary-centered velocities used in the diffusion step (Sec. 4.3, Eq. 8) in 2D and 3D respectively. These volumes are formed by connecting edge/face vertices to the barycenters of the neighboring triangles/tetrahedra. (c) and (d) show the control volumes for the per-vertex pressure solution (Sec. 4.4). The 2D version is formed by connecting the barycenters of all triangles around a vertex. In 3D, an enclosed mesh is formed around each vertex by combining all tetrahedral barycenters, face barycenters and edge centers. The 3D image above shows the 4 triangles created from one vertex face pair. Each triangle is made up of one tetrahedron barycenter, the face barycenter and one edge midpoint.

4 Solution Procedure

In order to solve the Navier-Stokes equations we follow the general solution procedure of "Stable Fluids" [7]. This procedure breaks Eqs. 1–2 down into several simpler steps which are solved sequentially. These steps are (in order) velocity self-advection, application of external forces, velocity diffusion, and divergence correction via pressure. Each of these four steps is detailed in order in the following subsections. The output values of each step are used as the input to the next.

4.1 Advection

This step solves the $(\mathbf{u} \cdot \nabla)\mathbf{u}$ portion of the momentum equation (Eq. 1). For this step only, we depart from the finite volume approach, and use an unconditionally stable semi-Lagrangian technique [7], similar to the method used in [24] for triangle meshes, but we have extended it to tetrahedral meshes for 3D simulations, as well. Semi-Lagrangian advection involves solving ordinary differential equations to trace characteristics backwards through the velocity field. Simply stated, in order to solve for the advected velocity at some point in space ($\mathbf{x}_i$), one traces the current velocity $\mathbf{u}_i$ backwards $-\Delta t$ to a new point in space. The backtracing can also be performed using a higher-order scheme like Runge-Kutta 2 or Runge-Kutta 4. The velocity at the new point is interpolated from neighboring values, and this

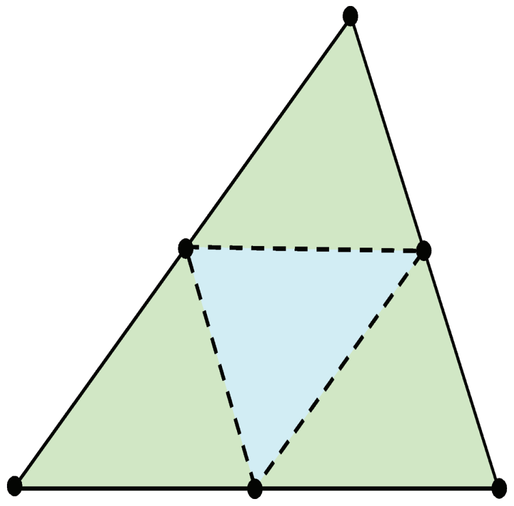

Figure 5: Velocity interpolation in 2D. Vertex velocities are first interpolated from neighboring edge velocities. Using only these values to then interpolate a velocity field over the entire triangle results in unacceptable numerical diffusion. This can be avoided by dividing the triangles into four regions. The velocity at a point in the center region (diagonal lines) is interpolated purely from edge-centered velocities. Velocities at points in three outer regions are interpolated from a combination of 2 edge-centered velocities and 1 vertex (averaged) velocity.

velocity is then used as the advected velocity at $\mathbf{x}_i$.

When back-tracing the velocity vectors at each edge/face center, we perform the computations using barycentric coordinates. This simplifies determination of whether a point is within a particular volume element, and, if not, gives an indication of which boundary the trajectory crossed over, leading to a natural algorithm for walking through the mesh data structure to find the current triangle/tetrahedron. Care must be taken to check that the vector does not pass through a boundary with a prescribed boundary value (whether it be the edge of the domain or an internal boundary). If the vector hits such a boundary, the velocity of the boundary is returned as the result of the trace.

The first step in performing interpolation is obtaining the velocity at vertices. This is obtained by computing an unweighted mean of all of the velocities on boundaries in the 1-neighborhood of that vertex, i.e. in 2d, all the edges cut by dotted lines in Fig. 4(c). We can now interpolate the velocity using barycentric coordinates. In order to avoid dissipation, we only use the vertex velocities when necessary. As is described in Figs. 5 and 6, this splits the triangle or tetrahedron into smaller triangles and tetrahedra.

Although finite volume methods may also be used to solve for the advection of the velocity field, by using semi-Lagrangian tracing, advection can be performed stably with a large time step [7]. Finite volume methods for advection are subject to the CFL condition, which imposes that $\mathbf{u}\Delta t$ everywhere must be less than the local size of the grid cells. Avoiding the CFL restriction is especially important on an unstructured mesh where having a few cells much smaller than the others would force the time step for the entire grid to be reduced.

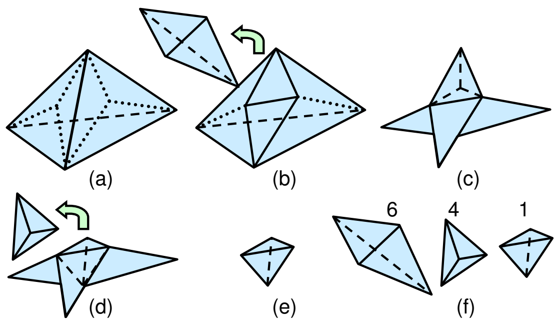

Figure 6: Velocity interpolation in 3D. Similar to the 2D interpolation (Fig. 5), a tetrahedron is divided into different interpolation regions, 11 in all. (a) shows the initial tetrahedron and the cuts created by cutting along each face to the barycenter. In (b), one of those pieces is removed. (c) shows the resulting structure after all 6 of these pieces are removed. In (d) a piece defined by one vertex and 3 face centers is removed. (e) shows the center tetrahedron created by the barycenters of the four faces of the original tetrahedron. In (f) we see that there are 6 tetrahedra corresponding to two original tetrahedron vertices and 2 face centers, 4 tetrahedra corresponding to 1 original tetrahedron vertex and 3 face centers and 1 tetrahedron formed by the face centers. When performing semi-Lagrangian advection, we determine in which of these 11 volumes the final point lies and interpolate the velocity from the corresponding 4 velocity values. Barycentric coordinates determine which volume contains the point as well as the linear weights used in interpolation.

4.2 Adding Forces

Adding body forces, such as gravity (indicated by $\mathbf{F}$ in Eq. 1), is very straightforward. For each location in the grid where velocity is stored, the velocity is updated by $\mathbf{u} := \mathbf{u} + \Delta t \mathbf{F}$.

4.3 Diffusion

Next we solve for the diffusion of the velocity, using an implicit finite volume formulation:

$$\int_\Omega (\mathbf{u} - \mathbf{u}^*)\, d\Omega = \Delta t \int_\Omega \nu \nabla^2 \mathbf{u}\, d\Omega$$

where $\mathbf{u}^*$ is the velocity before diffusion occurs, and $\mathbf{u}$ is the post-diffusion velocity. After application of Eq. 4 we obtain

$$\int_\Omega \mathbf{u}\, d\Omega - \Delta t \oint_\Gamma \nu \nabla \mathbf{u} \cdot \mathbf{n} d\Gamma = \int_\Omega \mathbf{u}^*\, d\Omega, \tag{8}$$

where it should be understood that this one vector equation represents three separate scalar equations, one for each component of the velocity. Each equation is solved independently by discretizing the integrals and the spatial derivatives. While, Ω and Γ are defined similarly as in Eq. 1– 5, ν, $\mathbf{u}$, Δt and $\mathbf{n}$ are the fluid viscosity, velocity, timestep, and boundary normals respectively. In this case, the control volume, faces, normals, and velocities used are the *dual* representations specified in Fig 4 (a) and (b).

The surface integral in Eq. 8, $\Delta t \oint_{\Gamma} \nu \nabla \mathbf{u} \cdot \mathbf{n} d\Gamma$, requires a control volume centered on the existing edge/face velocity values, and it requires the gradient of velocity (for each component) to be evaluated on the boundary of that control volume. The volumes satisfying these properties are shown in Fig. 4, (a) and (b). The computation of gradients like $\nabla \mathbf{u}$ on the control volume edge/face is discussed in Appendix A.

This matrix, as well as all others discussed in this paper, is sparse and symmetric positive definite. We use the conjugate gradient method [32] in order to solve the system of equations. In practice, with a conjugate gradient solver it is not necessary to explicitly assemble the system matrix. One just needs a routine which is able to compute the result of multiplying the matrix times a given vector (in this case a vector of grid velocities). The following pseudocode computes this matrix-vector product for the diffusion step (Eq. 8). In the pseudocode, `boundary` refers to the edge or face containing a velocity value, and `dualboundary` refers to an edge or face of the control volume around that velocity value. The `u` variable holds the input velocity value to the solver, while `output` holds the result of the matrix-vector multiplication:

```
1  for each boundary i do
2     output[i] = 0
3     for each dualboundary j around boundary[i] do
4        neighbor = edge center across dualboundary[j]
5        output[i] += dot(∇u(neighbor, boundary[i]), normal[j]) * ΔΓ[dualboundary[j]]
6     end
7     output[i] *= Δt * ν / ΔΩ
8     output[i] = u[boundary[i]] - output[i]
9  end
```

The `normal` used is the outward facing normal for each `dualboundary`. The above represents the evaluation of the left hand side of Eq. 8 for a $\mathbf{u}$ vector supplied by the conjugate gradient solver. The meaning of `∇u(a,b)` on line 5 is explained in Appendix A, where we discuss the computation of gradients on the unstructured mesh. The inner for loop solves the boundary integral for each boundary component of the control volume, and the last two lines place that partial result in the context of the left hand side.

4.4 Pressure Correction

The first three steps solve for the advection, external body forces, and diffusion. The resulting velocity field will not, in general, be divergence free, as required by the continuity equation (Eq. 2). The final step, then, is to enforce the continuity equation by solving a Poisson problem for pressure, p:

$$\int_\Omega \nabla^2 p \, d\Omega = \int_\Omega \nabla \cdot \mathbf{u} \, d\Omega. \tag{9}$$

This step is a combination of the ∇p portion of the momentum equation (Eq. 1) and the continuity equation (Eq. 2). By applying Eqs. 3 and 4, Eq. 9 becomes

$$\oint_\Gamma \nabla p \cdot \mathbf{n} \, d\Gamma = \oint_\Gamma \mathbf{u} \cdot \mathbf{n} \, d\Gamma. \tag{10}$$

Again, Ω and Γ are defined similarly as in Eq. 1– 5. p, $\mathbf{u}$, and $\mathbf{n}$ are the pressure, velocity, and boundary normals, respectively.

Since we use full velocity vectors on our boundaries, there are two components to the divergence in our system: that which is perpendicular to the boundary (cell divergence), and that which is tangential to the boundary (vertex divergence). We must solve Eq. 10 for both portions.

We first solve for p at cell centers to eliminate the divergence perpendicular to the boundary. Creating a numerical solver for this equation is relatively straightforward. The vector for the right hand side of the equation can be approximated just as in Eq. 7, with $\mathbf{f} \equiv \mathbf{u}$.

The operation of the matrix on the left side of the above equation can be created in the following way:

```
for each volume i do
   output[i] = 0
   center[i] = barycenter of simplex[i]
   for each boundary j of simplex[i] do
      neighbor = simplex barycenter across boundary[j]
      output[i] += dot(∇p(neighbor, center[i]), normal[j]) * ΔΓ[boundary[j]]
   end
end
```

The normal used here is the outward facing normal for each edge or face. The term “simplex” refers to a triangle in 2d and a tetrahedron in 3d. The variable p is the input to the solver while output stores the result.

In order to solve for the tangential component of divergence (the vertex divergence), we locate another set of pressure values at the vertices. The control volumes for the vertices are defined by the dual graph of the mesh, as shown by the dashed lines in Fig. 4, (c) and (d).

Let $\mathbf{u}_{\|}$ denote the velocity component tangential to the cell boundary. Then the resulting system of equations can be solved much the same as with the cell centered pressure, using Eq. 7, but this time with $\mathbf{f} \equiv \mathbf{u}_{\|}$, and with $\mathbf{n}_i$ and $\Delta\Gamma_i$ indicating normals and edges of the appropriate control volume. Code to apply the stencil matrix is also very similar:

```
for each vertex i do
   output[i] = 0
   for each boundary j of dual graph vertex[i] do
      output[i] += dot(∇p, normal[j]) * ΔΓ[dualboundary[j]]
   end
end
```

Again, the normal used here is the outward facing normal for each `dualboundary`. In 2D, ∇p is the directional derivative along an edge emanating from the vertex, and, in 3D, it is the in-face derivative on the face emanating from the vertex, coincident with the dual boundary (see Fig. 4). Refer to Appendix A for more detail on the computation of gradients on edges and faces.

4.5 Other Steps

Vorticity confinement was presented by Fedkiw et. al. [14] as a way of reintroducing the swirling motion of fluid simulations lost due to semi-Lagrangian advection and coarse grid calculations. Using the finite volume method, the vorticity at cell centers can be easily calculated by using Eq. 5–6, with $\mathbf{f} \equiv \mathbf{u}$. Then the gradient of the vorticity can be calculated on cell edges (see Appendix A). The vorticity confinement force applied to edges is a simple function of this gradient.

Fedkiw et. al. [14] also employed temperature and density values for the smoke in order to create forces. We place temperature and density at vertices and average them to create forces at edge/face centers.

5 Results and Discussions

5.1 Implementation

To implement our proposed method it is necessary to efficiently store and manage the simulation grid mesh data. In 2D, we use a simplified doubly-connected edge list as described in [33] and a relatively straightforward extension of the same technique in 3D. This data structure allows us to efficiently retrieve cell adjacency information and other cell data.

Several methods exist for creating good tetrahedral meshes for numerical simulation. Mesh generators are available both publicly and for charge (see [34] for a survey). In all of our 3D demos, we used either Müeller's mesh generation technique [35] or files created by Cutler [36].

5.2 Demonstrations

Next, we demonstrate the results of our method with several challenging scenarios. All simulations were run on a Pentium-4 PC with a 3.2 GHz processor. See the color plate for images rendered using the output of our simulator.

As mentioned earlier, intricate boundaries on the fluid are a challenge for previous methods. Particularly, a typical implementaion using ghost cells requires all boundaries to be at least two cells thick and forces the elimination of all sharp corners. However, Fig. 2 shows that we are able to control the path of smoke as desired simply by setting appropriate boundary conditions. Our boundaries can be infinitessimally thin (an edge in 2D, face in 3D), and this allows us to easily simulate the effect of a semi-permeable membrane by transferring smoke density in proportion to the density gradient across the boundary times a permeability constant, $0 \leq \pi \leq 1$. This step occurs every time step of simulation.

We also show that smoke is able to interface directly with complex surfaces (see Fig. 7). Note that, despite the relatively low resolution of the simulation mesh used, when we remove the obstacle from the rendering, its outline can be clearly seen, demonstrating that our method is accurately enforcing the boundary conditions imposed by the object. The smoke neither penetrates nor gives the obstacle a wide girth as would be the case with some of the previous methods. The smoke also properly flows through the holes in the middle of the obstacle.

We also demonstrate several free surface fluid interactions. First, we demonstrate two altered forms of a well-known test scenario: the "broken dam problem" (Fig. 8). The first shows a free surface interacting with the intricated details of the MIT Gargoyle Model. Note that the thin features (i.e. mouth and wall mounting) are able to properly fill and drain with a very low resolution. The second follows a similar simulation with the Stanford Bunny. Note how smoothly the fluid is able to follow the curve of the bunny's tail and hind legs.

This system can also easily impose moving boundary conditions (Fig. 9). We simulated the effect of sliding a cone shaped glass across a flat surface, followed by a hard stop. This simulation shows that our work easily handles the smooth, non-axis-aligned boundaries that fluid frequently encounters in daily experience.

All of our simulations are run on rather small meshes. They range from 50,000 tetrahedra to 135,000 tetrahedra. These are approximately equivalent to FDM meshes with approximately 35x35x35 to 50x50x50 resolution. These sizes are considerably smaller than mesh sizes preferred when using previous techniques. However, our improved boundary conditions are able to still provide visually pleasing results. Please see our webpage and supplementary video clips available at http://gamma.cs.unc.edu/FVM/.

5.3 Comparison

We have shown that the finite volume method can reproduce the effect of the existing state-of-the-art finite difference fluid simulators, as well as solving several problems caused by irregular internal boundaries. Since triangles and tetrahedra may be distributed as desired throughout the 3D space, resolution may be refined in key areas and reduced in others. Also, since any cell edge may be considered a fixed boundary, this approach allows accurate representation of the irregular boundaries. Such a capability is particularly useful since many 3D graphics data sets are non-manifold triangle sets (such as cloth, etc).

Also, gridding effects in the fluid flow are reduced by our use of full velocity vectors on all face centers. In comparison, the typical staggered grid finite difference method stores only one component of velocity on each face, which coincides with the face normal. With these methods it is more difficult to obtain vorticity (the swirling motion of fluids) around a single cell, for instance. This is because the vorticity around one cell runs completely tangential to the faces of the cell. By storing the complete vector, we can represent such flows.

Due to non-regular location and connectivity information extra memory must be used to store these values. The additional memory accesses during simulation reduce runtime performance as compared to the finite difference methods. However, since boundaries can be handled with more accuracy by finite volume methods, lower resolution meshes can give comparable results. The simulations in this paper ran between .3 to 4 frames per minute, although our code has not been optimized.

5.4 Resolution

The quest for higher resolution fluid simulations is driven by a number of factors. Among these is eliminating aliasing artifacts in the free surface of fluid, better approximating boundary conditions and better handling of thin fluid features. Our proposed technique is able to resolve boundaries at lower resolutions than previous finite difference based techniques. While some aliasing artifacts still exist in our free surfaces, we believe the lack of regularity in the mesh makes the artifacts less noticeable (i.e. irregular surface imperfections vs. regular stair steps). However, our technique would still require high resolution to resolve thin fluid features (as seen in splashes).

6 Conclusion and Future Work

We have presented a novel method for 2D and 3D flows on domains of arbitrary topology using a finite volume formulation. We showed that this approach can generate similar results to the existing state-of-the-art techniques based on finite difference methods, while alleviating regular grid effects and better capturing fine features arising from irregular internal boundary conditions.

This method leads to several areas of future research. There remain some numerical issues, for instance. Domains of unusual topology (such as the bunny and gargoyle) can result in ill-conditioned systems depending upon how they are discretized. Specifically, the several narrow passages (i.e. bunny ears, gargoyle mouth, etc.), can lead to a stiffer system of equations for the diffusion and the projection steps, requiring a greater number of iterations for our conjugate gradient solver to converge. Also, widely varying element volume sizes can potentially lead to numerical problems. Our simulator is able to handle volumes with up to three orders of magnitude discrepancy. However, the stiffness of the system scales as the ratio of largest to smallest volume. Future work can focus on further improving numerical stability under these circumstances. Mesh refinement could also be added so that the simulator can automatically detect key areas of interest and refine the underlying mesh.

Furthermore, we plan to investigate issues arising from coupling multiple dynamical systems, e.g. fluid interacting with both deformable and rigid bodies that can tear or fracture. Adaptive remeshing will likely to be required for computational efficiency and desired visual simulation. Fortunately, fast techniques for tetrahedral meshing have been investigated by many researchers [37–39]. These methods could be used to create an adaptive remeshing system.

A Computing Gradients

Since the nodes on our grids can be located anywhere, we must be able to approximate ∇f between any two points in space. This can be accomplished using a directional derivative. Given two points in space ($\mathbf{x}_i$ and $\mathbf{x}_j$) with some scalar f defined at each point,

$$\nabla f(\mathbf{x}_i, \mathbf{x}_j) \approx \frac{f(\mathbf{x}_i) - f(\mathbf{x}_j)}{\|\mathbf{x}_i - \mathbf{x}_j\|} * \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}. \tag{11}$$

In 3D, our solution technique requires the evaluation of ∇f on the faces of tetrahedra. If a triangle $\mathbf{T}$ is defined by the three points $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{x}_k$, let $\mathbf{e}_1$ be $(\mathbf{x}_j - \mathbf{x}_i)$ and $\mathbf{e}_2$ be $(\mathbf{x}_k - \mathbf{x}_i)$. Then we can write the equation for f on that face as a function of two of the coordinates, e.g. $f(x,z) = Ax + Bz + C$, with coefficients A, B and C to be determined. By plugging the three known values for f and the known vertex locations we have 3 equations allowing us to solve for the 3 unknowns, A, B, and C. Then we have simply that $\partial f/\partial x = A$ and $\partial f/\partial z = B$. It just remains to find the remaining

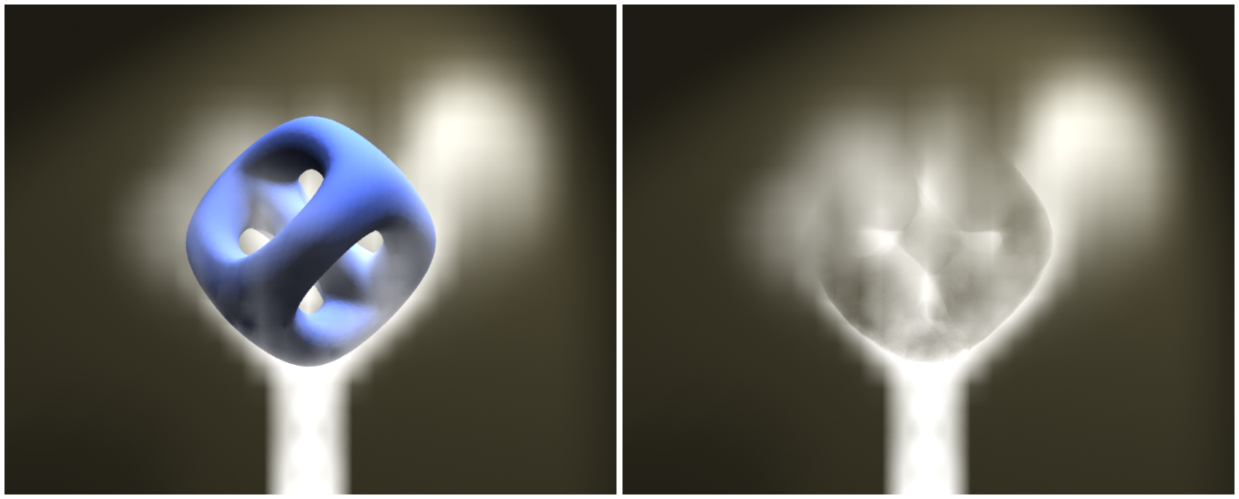

Figure 7: Here we see 3D smoke interacting with novel boundary conditions [40, 41]. The second image has the obstacle removed from the rendering. This shows how cleanly our simulation is able to approximate curved boundaries, as well as holes, even with relatively coarse resolutions.

component of the gradient by projecting the known components of the gradient onto the plane of the triangle.

References

[1] A. J. Chen and Y. Kallinderis. Adaptive hybrid (prismatic-tetrahedral) grid for incompressible flows. *Intl. Journal of of Numerical Methods for Fluids*, 26:1085, 1998.

[2] F.E. Ham, F.S. Lien, and A.B. Strong. A fully conservative second-order fintie difference scheme for incompressible flow on nonuniform grids. *Journal of Computational Physics*, 177:117–133, 2002.

[3] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph.*, 23(3):457–462, 2004.

[4] Francis. H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189, December 1965.

[5] Nick Foster and Ronald Fedkiw. Practical animations of liquids. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 23–30. ACM Press / ACM SIGGRAPH, 2001.

[6] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing: GMIP*, 58(5):471–483, 1996.

[7] Jos Stam. Stable fluids. In Alyn Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 121–128, Los Angeles, 1999. Addison Wesley Longman.

[8] Mark Carlson, Peter J. Mucha, III R. Brooks Van Horn, and Greg Turk. Melting and flowing. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*, pages 167–174. ACM Press, 2002.

[9] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744. ACM Press, 2002.

[10] M. Carlson, P. Mucha, and G. Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. In *Proc. of the ACM SIGGRAPH*. ACM Press, 2004.

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

[12] S. Lin and Y. Yu. Controllable smoke animation with guiding objects. *ACM Transactions on Graphics*, 24(1):140–164, 2005.

[13] Oh-Young Song, Hyuncheol Shin, and Hyeong-Seok Ko. Stable but nondissipative water. *ACM Transactions on Graphics*, 24(1), 2005.

[14] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 15–22. ACM Press / ACM SIGGRAPH, 2001.

[15] Viorel Mihalef, Dimitris Metaxas, and Mark Sussman. Natural phenomena: Animation and control of breaking waves. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on computer animation*, 2004.

[16] Murilo F. Tome and Sean McKee. GENSMAC: A computational marker and cell method for free surface flows in general domains. *J. Comp. Phys.*, 110:171–186, 1994.

[17] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 129–136. ACM Press, 1995.

[18] M. Desbrun and M. P. Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96 (Proceedings of EG Workshop on Animation and Simulation)*, pages 61–76. Springer-Verlag, August 1996.

[19] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabric Nayret, and Jean-Dominique Gascuel. Animating lava flows. In *Graphics Interface (GI'99) Proceedings*, pages 203–210, June 1999.

[20] M. Muller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[21] G. R. Liu. *Mesh Free Methods: Moving Beyond the Finite Element Method*. CRC Press, 1st edition, 2002.

[22] G. R. Liu and M. B. Liu. *Smoothed Particle Hydrodynamics: A Meshfree Particle Method*. World Scientific Pub. Co., Inc., 2003.

[23] J. Stam. Flows on surfaces of arbitrary topology. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 2003.

[24] Lin Shi and Yizhou Yu. Inviscid and incompressible fluid simulation on triangle meshes. *Journal of Computer Animation and Virtual Worlds*, 15(3–4):173–181, 2004.

[25] R. J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhauser Verlag, 1992.

[26] R. J. LeVeque and D. G. Crighton. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[27] J. J. H. Miller and S. Wang. An exponentially fitted finite volume method for the numerical solution of 2D unsteady incompressible flow problems. *Journal of Computational Physics*, 115:56, 1994.

[28] D. Pan, C.-H. Lu, and J.-C. Cheng. Incompressible flow solution on unstructured triangular meshes. *Numerical Heat Transfer Part B*, 26:207, 1994.

[29] Dongjoo Kim and Haecheon Choi. A second-order time-accurate finite volume method for unsteady incompresible flow on hybrid unstructured grids. *Journal of Computational Physics*, 162:411–428, 2000.

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

[31] S. Elcott, Y. Tong, E. Kanso, P. Schroder, and M. Desbrun. Stable, circulation-preserving, simplicial fluids. 2006. http://www.geometry.caltech.edu/pubs.html.

[32] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMUCS-TR-94-125, Carnegie Mellon University, 1994. (See also http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.ps.).

[33] M. de Berg, M. Overmars, M. van Kreveld, and O. Schwarzkopf. *Introduction to Computational Geometry: Theory and Applications*. Springer, 2nd edition, 2000.

[34] Robert Schneiders. Mesh generation: Software, 2004. http://www-users.informatik.rwth-aachen.de/ roberts/software.html.

[35] Matthias Mueller and M. Teschner. Volumetric meshes for real-time medical simulations. In *Proceedings of BVM (Bildverarbeitung fur die Medizin 2003)*, pages 279–283, 2003.

[36] Barbara Cutler, Julie Dorsey, and Leonard McMillan. Simplification and improvement of tetrahedral models for simulation. *Proceedings of the Eurographics Symposium on Geometry Processing 2004*, 2004.

[37] Yuanxin Liu and Jack Snoeyink. A comparison of five implementations of 3d delaunay tessellation. In *Combinatorial and Computational Geometry, MSRI series*, 2005.

[38] Robert Bridson, Joseph Teran, Neil Molino, and Ronald Fedkiw. Adaptive physics based tetrahedral mesh generation using level sets. In *Engineering with Computers*, (in press).

[39] Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable*, pages 103–114, 2003.

[40] C. H. Sequin and L. Xiao. K12 and the genus-6 tiffany lamp. In *ISAMA*, pages 17–19, June 2004.

[41] C. H. Sequin. Cad tools for aesthetic engineering. In *JCAD Vol. 37 No. 7*, pages 737–750, June 2005.

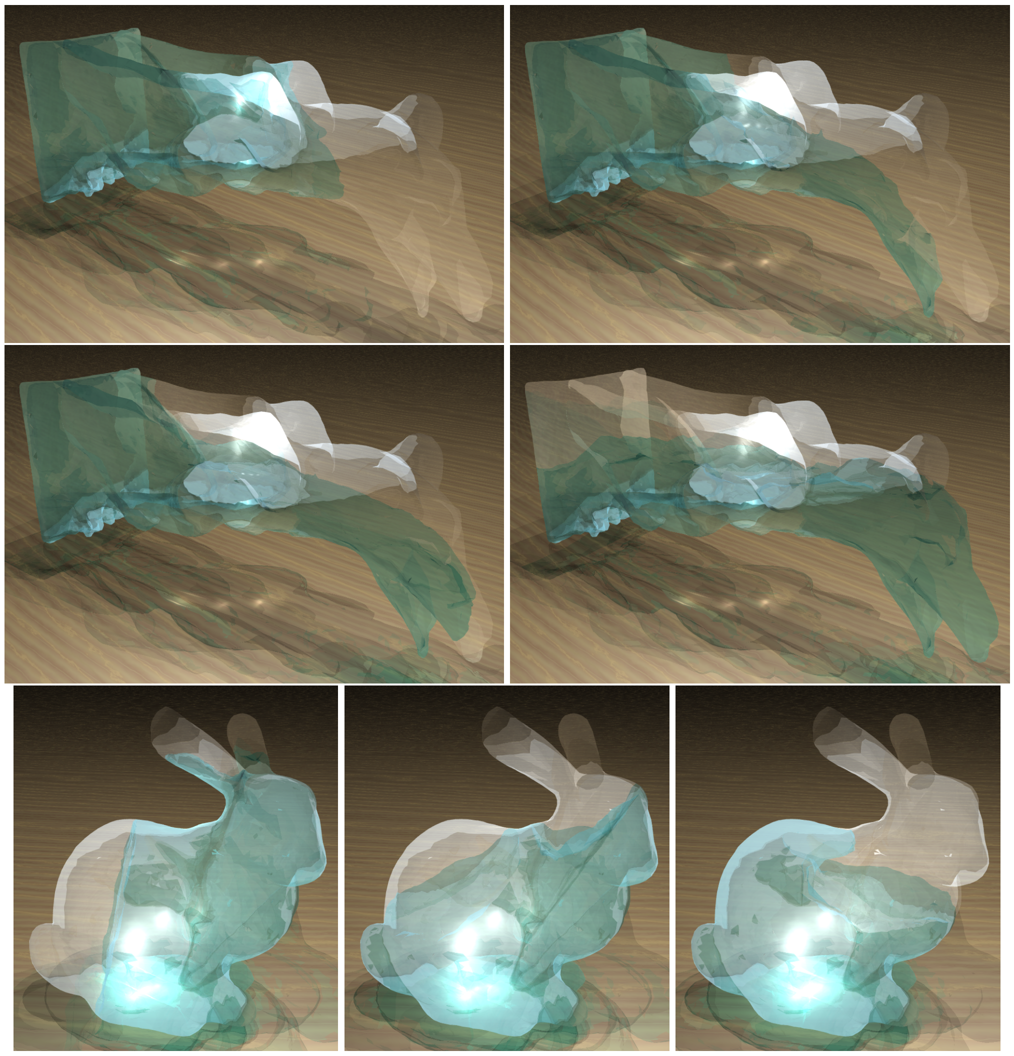

Figure 8: This sequence of images taken from our simulation shows how our system can easily handle complex domains for fluid simulations. It also shows how we are able to achieve convincing results with low resolution: 50,000 tetrahedra, roughly equivalent to a 37x37x37 resolution cube!

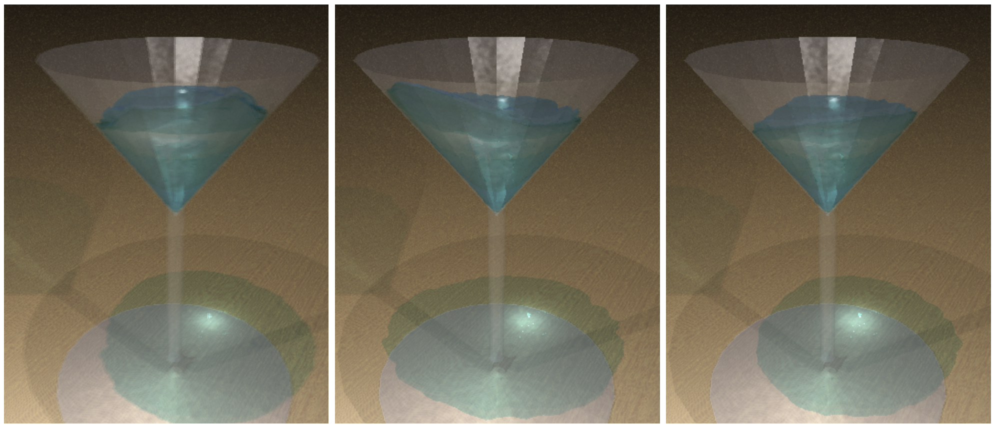

Figure 9: In this demonstration, we show a “real world case” requiring good boundary conditions for smooth, non-axis-aligned boundaries as well as a sharp feature. The simulation begins with the cup pushed to the left, followed by an abrupt stop. The simulation continues as the water splashes back and forth in the cup.