Title
Fluid animation with dynamic meshes

Permalink
https://escholarship.org/uc/item/0359d8gz

Journal
ACM Transactions on Graphics, 25(3)

ISSN
0730-0301

Authors
Klingner, Bryan M
Feldman, Bryan E
Chentanez, Nuttapong
et al.

Publication Date
2006-07-01

DOI
10.1145/1141911.1141961

Supplemental Material
https://escholarship.org/uc/item/0359d8gz#supplemental

Peer reviewed
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.


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.

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 [Eldrott 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.

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., 1987], [Desbrun and Cani, 1996], [Cani and Desbrun, 1997], [Stora et al., 1999], [Müller et al., 2003], [Fremoze 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 [Eldrott 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], [Eldrott 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:

\[
\frac{\partial \mathbf{u}}{\partial t} = - (\mathbf{u} \cdot \nabla) \mathbf{u} - \frac{\nabla p}{\rho} + \frac{\mathbf{f}}{\rho}
\]

subject to the mass conservation constraint for incompressible fluids:

\[
\nabla \cdot \mathbf{u} = 0
\]

In these equations, \( \mathbf{u} \) is the fluid velocity, \( t \) time, \( p \) pressure, \( \rho \) density, and \( \mathbf{f} \) any external forces. The symbol \( \nabla \) denotes the vector of differential operators \( \nabla = \left[ \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right]^T \).

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 [Eldrott 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, \( u_t \), is computed at each tetrahedron circumcenter, then we interpolate within Voronoi cells using \( u_t \) values at the cell vertices. Velocity \( u_t \) for tetrahedron \( t \) is found by solving the small linear system

\[
N_t u_t = z_t
\]

where \( N_t \) is a matrix containing 4 rows of the face normals of \( t \) and \( z_t \) 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 \( u_t \) 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 \( x \) as a weighted sum of the polytope’s node values where node \( t \)'s unnormalized weight is computed as

\[
w_t(x) = \frac{|N_t|}{\prod_{f \in \sigma_t} |n_f \cdot x + d_f|},
\]

Here, \( \sigma_t \) is the set of polytope faces that intersect at node \( t \). The denominator is the product of distances from \( x \) to the faces in \( \sigma_t \) computed using the face normals \( n_f \) and plane offsets \( d_f \). \( |N_t| \) is the determinant of a matrix of face normals in \( \sigma \). 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 \( \frac{1}{6} |E_t| \) where \( E_t \) 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

\[
w_t(x) = \frac{6\text{Vol}(t)}{\prod_{i=1}^{t} (p_i - p_v) \cdot (c_i - x)}
\]

where \( w_t(x) \) is the weight associated with the node at tetrahedron \( t \)'s circumcenter, \( \text{Vol}(t) \) is the volume of tetrahedron \( t \), \( p_v \) is the position of the node associated with the Voronoi cell, \( p_i \) are positions of the other nodes of \( t \), \( c_i \) the circumcenter of \( t \), and \( x \) 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 fashion 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.

Suppose at time \( t \) velocities are stored at locations \( x^{(i)} \) (in our case, the face circumcenters), and we want to find the velocity at a particular face location \( x' \). We trace back from \( x^{(i)} \) through the velocity field of the previous time step to a point \( x'' \), which has no necessary correspondence to any feature of the old mesh. Then, we update the velocity at \( x'' \) to the value interpolated from the old velocity field at \( x' \). 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 \( \mu(x) \) that, for any point \( x \) 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...
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 \( R \) that is multiplied by a vector of the pressures that surround a rigid body. \( R \) can be formed by a series of matrix multiplications:

\[
R = \begin{bmatrix} b_1 \vdots b_k \end{bmatrix} \begin{bmatrix} M^{-1} & 0 \\ 0 & I^{-1} \end{bmatrix} \begin{bmatrix} A_1 b_1^T \\ \vdots \\ A_k b_k^T \end{bmatrix}
\]  

where \( b_i = [n_i^T \ (r_i \times n_i)^T] \), \( n_i \) is the normal of the \( i \)th face, \( r_i \) is the vector from the rigid objects center of mass to position of the \( i \)th 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

\[
\mu(x) = k_0 + \min(k_d d(x), k_s (1 - s(x)), k_\omega (1 - \omega(x)))
\]  

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 \( R \) that is multiplied by a vector of the pressures that surround a rigid body. \( R \) can be formed by a series of matrix multiplications:

\[
R = \begin{bmatrix} b_1 \vdots b_k \end{bmatrix} \begin{bmatrix} M^{-1} & 0 \\ 0 & I^{-1} \end{bmatrix} \begin{bmatrix} A_1 b_1^T \\ \vdots \\ A_k b_k^T \end{bmatrix}
\]  

where \( b_i = [n_i^T \ (r_i \times n_i)^T] \), \( n_i \) is the normal of the \( i \)th face, \( r_i \) is the vector from the rigid objects center of mass to position of the \( i \)th 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

\[
\mu(x) = k_0 + \min(k_d d(x), k_s (1 - s(x)), k_\omega (1 - \omega(x)))
\]  

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.
the body by the middle ($6 \times 6$) block matrix. $M$ is a diagonal matrix with the mass of the rigid body on the diagonals and $I$ 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 $A$ multiplied by a vector of all the pressures. A row of $A$ 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 $A$ that belongs to a face at the fluid-rigid interface has element values obtained from the corresponding row of $R$.

The elements of this row are placed at columns corresponding to the pressures that surround the rigid-body. With $A$ built, mass conservation including two way coupling proceeds much in the same way as in the all-fluid case, with $A$ replacing the role of the discrete gradient matrix. For a given vector of pressures, $p$, the intermediate velocity field, $z^*$, is accelerated to the end-of-step velocity, $z$, by $z = z^* + \Delta t A p$.

For the fluid faces, $z^*$ is found by applying all terms of Equation (1) except the pressure term. For the fluid-rigid faces, $z^*$ is found using a rigid body simulator without pressure forces applied. To find a particular pressure that accelerates $z^*$ such that $z$ is divergence free we solve the linear system

$$\Delta t DA p = -Dz^*.$$  \hspace{1cm} (7)

This linear system can be solved efficiently using PCG since the the matrix $DA$, 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 $R$ matrix that correctly computes face accelerations due to pressure. For example, one could easily alter $R$ such that the body was constrained to just rotate about the origin by replacing $b_i$ in Equation (6) with $b'_i = [(r_i \times n_j)T]$ and using only the $I^{-1}$ 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 Pyramidal [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.

![Figure 1](image1.png)

**Figure 1:** Red particles are transferred 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.

<table>
<thead>
<tr>
<th></th>
<th>Remeshing time per frame (mean)</th>
<th>Total time per frame (mean)</th>
<th>Percent remeshing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>13.2 sec</td>
<td>64.8 sec</td>
<td>20.3%</td>
</tr>
<tr>
<td>Figure 5</td>
<td>8.33 sec</td>
<td>44.5 sec</td>
<td>18.7%</td>
</tr>
<tr>
<td>Figure 6</td>
<td>5.76 sec</td>
<td>35.8 sec</td>
<td>16.1%</td>
</tr>
<tr>
<td>Figure 7</td>
<td>313 sec</td>
<td>796 sec</td>
<td>39.3%</td>
</tr>
</tbody>
</table>

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 tetraedra. 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


\(^1\)http://www.mathworks.com

\(^2\)http://sourceforge.net/projects/pixie
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.


