Fluid Simulation on Unstructured Quadrilateral Surface Meshes

Haimasree Bhattacharya
University of Utah

Joshua A. Levine
Clemson University

Adam W. Bargteil
University of Utah

Abstract

In this paper, we present a method for fluid simulation on unstructured quadrilateral surface meshes. We solve the Navier-Stokes equations by performing the traditional steps of fluid simulation, semi-Lagrangian advection and pressure projection, directly on the surface. We include level-set based front-tracking for visualizing "liquids," while we use densities to visualize "smoke." We demonstrate our method on a variety of meshes and create an assortment of visual effects.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation.

Keywords: Fluid simulation, quadrilateral meshes, physics-based animation.

1 Introduction

Fluid simulation has been one of the greatest successes of physics-based animation, generating hundreds of research papers and a great many special effects. This paper studies fluid simulation on two-manifolds embedded in three-dimensional space, a problem that has received relatively little attention from the computer animation community. While perhaps somewhat esoteric, such simulations produce compelling visual effects and are a good model of "sand pictures" and other instances of fluid-like materials sandwiched between plates of glass, which are common decorations in homes and offices.

In contrast to previous work, which performed fluid simulations on triangle meshes [Shi and Yu 2004] or highly structured subdivision surfaces [Stam 2003], this paper seeks to perform fluid simulation on arbitrary, unstructured quadrilateral surface meshes. To do so, we adapt the traditional advection and pressure projection steps to be performed directly on the surface. Unlike the regular grids typically used in three-dimensional fluid simulation, on unstructured meshes there is no inherent global parameterization. Instead, we parameterize each quadrilateral individually and convert data between parameterizations as it flows along the surface. We also provide two visualization techniques to visualize our surface flows: a level-set interface tracking method for immiscible fluids and density advection for miscible fluids or smoke.

Quadrilateral meshes offer some advantages over the more common triangle meshes. Quad meshes require fewer elements to achieve similar approximation power [Cifuentes and Kalbag 1992; Shepherd and Johnson 2008]. Moreover, quad meshes achieve reduced approximation error when the approximated function is not convex [D'Azevedo 2000]. Most importantly, however, quadrilaterals can align with geometric features and anisotropies without necessarily degrading the quality of individual elements.

Contact email: {hb123,adamb}@cs.utah.edu, levinej@clemson.edu

The cost of these advantages is that computing quadrilateral meshes is generally more difficult. Recently, however, the computer graphics community has invested much effort and made much progress in techniques for high quality quadrangulations of surfaces, with research both in mesh generation and mesh improvement techniques. Bommes et al. [2012] discuss the state-of-the-art for quad meshes. To leverage these developments and exploit the advantages of quadrilateral meshes, we present a method to simulate fluid on arbitrary, unstructured quadrilateral surface meshes, with or without boundary. Our method is entirely generic and can be applied to any quadrilateral mesh, independent of how the mesh was generated.

2 Related Work

Fluid simulation has been well studied across the fields of science since the invention of computers. In computer graphics, Foster and Metaxas [1996] performed the first fully three-dimensional fluid simulations. Timestep restrictions led Stam [1999] to advocate the *Stable Fluids* approach. However, the semi-Lagrangian advection scheme leads to significant numerical damping, which led Fedkiw and colleagues [2001] to introduce vorticity confinement. Since that time, hundreds of fluid simulation papers have been published and a complete review is beyond the scope of this paper.

Fluid simulation on surface meshes is a topic that has received relatively little attention. Stam [2003] was the first to model fluid flow on surface meshes. He used Catmull-Clark subdivision [Catmull and Clark 1978] to model surface geometry and took advantage of the natural cubic B-spline parameterization of individual surface patches. He then solved the Navier-Stokes equations in curvilinear coordinates in each patch, with special handling of patch boundaries. Our goal is to simulate on unstructured quadrilateral meshes, without being limited to Catmull-Clark surfaces.

Shi and Yu [2004] presented a novel technique to simulate inviscid incompressible fluids on triangular meshes. They perform simulation directly on the triangular mesh, avoiding parameterization distortion, and produced compelling results. Like them we seek to simulate directly on the mesh, but we target quadrilateral meshes rather than triangle meshes.

Recently, Hegeman and colleagues [2009] modeled fluid flow on surfaces parametrized using conformal (i.e angle preserving) geometric structures. Their GPU-based implementation was able to achieve impressive frame rates. However, they were limited to genus zero surfaces. Researchers have also considered the related problems of water drops flowing on surfaces [Wang et al. 2005; Zhang et al. 2012] and solving the shallow water equations on surfaces [Wang et al. 2007].

Our method generalizes the steps of fluid simulation to unstructured quadrilateral surface meshes. Quadrilateral mesh generation is an active area of research both in computer graphics and computational science [Alliez et al. 2005; Bommes et al. 2012]. Converting from unstructured data requires some effort. For example, in the simplest case, if a given mesh is triangular, Catmull-Clark subdivision or triangle pairing [Gurung et al. 2011; Tarini et al. 2010] could be used to trivially remesh to quadrilateral elements. However, in practice such meshes created with these techniques rarely are desirable for simulation because they can generate high numbers of ex-

traordinary vertices (those having valence other than four) as well as elements with poor alignment and quality. In the former situation, extraordinary vertices require special case numerics, reducing the simplicity (and sometimes the efficiency). The latter situation is more difficult, as poorly shaped elements can have small volumes, or worse be non-convex, leading to subsequent bad numerical stability in the simulation.

To address these concerns, more advanced techniques for quadrilateral meshing have been proposed in the past few years. These techniques tend to fall into three different categories. One class of techniques constructs global parameterizations through energy minimizations [Ray et al. 2006; Bommes et al. 2009; Bommes et al. 2013]. Since these techniques require global energy minimization techniques for computing the parameterization, they are often aided by initially cutting the base domain into a collection of disk patches, which simplifies the problem. Tong et al. [2006] provide a user-driven approach to designing the placement of singular vertices, whereas Dong et al. [2006] use a semi-automatic approach driven by scalar field analysis with the Morse-Smale complex. Direction fields are often used to guide these parameterizations, both locally and globally [Ray et al. 2008; Lai et al. 2010]. Finally, a third class of techniques relies on an energy optimization of centroid Voronoi tesselations [Liu et al. 2009] to drive quadrilateral remeshing [Lévy and Liu 2010]. While quality meshing is an important goal, a complete understanding of the interplay between mesh quality and simulation quality is beyond the scope of this work. Instead we focus herein on aspects of developing techniques for simulation that generalize to the setting of quadrilateral mesh inputs.

3 Methods

The Navier-Stokes equations model incompressible flow. Dropping the viscosity term we arrive at the Euler equations,

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

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

where $\mathbf{u}$ is the velocity, p is pressure, ρ is the density, $\mathbf{f}$ is external forces, and ∇ denotes the gradient operator.

Our goal is to generalize standard computer graphics fluid simulation techniques for solving these equations to arbitrary two-manifolds discretized with unstructured quadrilateral meshes. This goal comprises several subtasks. First, we store the mesh and associated data by enumerating each quadrilateral, edge, and vertex in the mesh and storing connectivity information as well as simulation variables. Second, to parameterize the surface, we opt for the straightforward solution and parameterize each quadrilateral individually. Third, to advect variables through the mesh we employ standard semi-Lagrangian characteristic tracing techniques, which are complicated by the fact that when a path crosses an edge in the mesh the parameterization changes. Fourth, to perform pressure projection, we use standard finite volume methods. Finally, to visualize our results, we either track an interface using a level-set method or track density values through time.

3.1 Storing Data On The Mesh

We use the standard approach of staggering pressure and velocity samples with pressures stored at quadrilateral centroids and velocity stored at edge midpoints. While the structure of regular grids allows for trivial point location and topological queries, our more general meshes require additional data structures to answer such queries. In the case of an unstructured mesh, we must explicitly enumerate and store each edge, in addition to vertices and quadrilateral faces. We additionally cache the solutions to a number of calculations (such as the parameterization of a quadrilateral) for later reuse. Specifically, we store:

For each **Quadrilateral, Face, or Cell**

- pressure
- centroid and normal of the best fitting plane
- a parameterization of the plane—two orthogonal three-dimensional vectors
- pointers to the four incident vertices, in counterclockwise order
- pointers to the four incident edges, in counterclockwise order
- pointers to the four adjacent quadrilaterals, in counterclockwise order

For each **Edge**

- signed flux
- pointers to the two incident quadrilaterals, with the convention that positive flux represents flow from q_0 to q_1
- pointers to the two incident vertices, with the convention that the ordering of v_0 and v_1 is counterclockwise in q_0
- distance from the edge midpoint to each of the adjacent cell centers
- edge midpoint in three-dimensional space
- two-dimensional normals to the edge in each incident quadrilaterals coordinate system

For each **Vertex**

- density, distance values, temperature
- two-dimensional velocity (computed from the edge fluxes, see Section 3.3.2)
- centroid and normal of the best fitting plane
- a parameterization of the best fitting plane—two orthogonal three-dimensional vectors
- pointers to vertices in the one-ring, in counterclockwise order
- pointers to edges in the one-ring, in counterclockwise order

3.2 Parameterization

For a given quadrilateral, we create a two-dimensional parameterization by first computing its centroid and the "best-fitting" plane. The centroid is simply the average of the four vertices—if the positions of the quadrilateral's vertices are $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$, then the centroid is $\mathbf{c} = 0.25(\mathbf{v}_0 + \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3)$. To find the best fitting plane we apply principal component analysis to the covariance matrix of the four points and choose the plane's normal to be the direction of least variation. Specifically let $\mathbf{A}$ be the covariance matrix

$$\mathbf{A} = \begin{pmatrix} (\mathbf{v}_0 - \mathbf{c})^T \\ (\mathbf{v}_1 - \mathbf{c})^T \\ (\mathbf{v}_2 - \mathbf{c})^T \\ (\mathbf{v}_3 - \mathbf{c})^T \end{pmatrix}^T \begin{pmatrix} (\mathbf{v}_0 - \mathbf{c})^T \\ (\mathbf{v}_1 - \mathbf{c})^T \\ (\mathbf{v}_2 - \mathbf{c})^T \\ (\mathbf{v}_3 - \mathbf{c})^T \end{pmatrix}, \tag{3}$$

the normal, $\mathbf{n}$, to the best-fitting plane will be the eigenvector corresponding to the smallest eigenvalue. The origin of the local coordinate system is $\mathbf{v}_0$ and, letting $\mathbf{v}_{01} = \mathbf{v}_1 - \mathbf{v}_0$, the coordinate axes are

$$\mathbf{e}_0 = \frac{\mathbf{v}_{01} - (\mathbf{v}_{01} \cdot \mathbf{n})\,\mathbf{n}}{\|\mathbf{v}_{01} - (\mathbf{v}_{01} \cdot \mathbf{n})\,\mathbf{n}\|} \tag{4}$$

and

$$\mathbf{e}_1 = \mathbf{n} \times \mathbf{e}_0. \tag{5}$$

A point, $\mathbf{x}$, on the three-dimensional quadrilateral can be re-parameterized in this two-dimensional space as $(x_x, x_y) = ((\mathbf{x} - \mathbf{v}_0) \cdot \mathbf{e}_0, (\mathbf{x} - \mathbf{v}_0) \cdot \mathbf{e}_1)$, to allow for, e.g., bilinear interpolation. Note that (x_x, x_y) are not necessarily between 0 and 1 and that computing bilinear weights requires re-parameterization of the quadrilateral's vertices and solving a quadratic equation.

3.3 Advection

Semi-Lagrangian advection is a two-step process. First, a path is traced backwards along a characteristic of the flow field. Second, the velocity is evaluated at the end of this path, which may be at an arbitrary point on the surface.

3.3.1 Characteristic Tracing

In our approach we trace a piecewise linear path on the surface. In general, we could begin at any arbitrary point, though in practice we usually advect only quantities stored at edge midpoints and vertices of the mesh. Following Feldman and colleagues [2005], we do not apply expensive velocity fitting while tracing characteristics, opting instead for bilinear interpolation of (two-dimensional) velocities computed at vertices. Given a starting point $\mathbf{x}_0$ and a timestep Δt, we initialize a "timer" $t = \Delta t$ and begin our tracing the characteristic with the following loop:

1. $\mathbf{x}^* = \mathbf{x} - t\mathbf{u}(\mathbf{x})$
2. Compute bilinear weights for $\mathbf{x}^*$
3. If bilinear weights $\in [0, 1] \times [0, 1]$, return $\mathbf{x}^*$
4. Check for intersection along ray from $\mathbf{x}$ to $\mathbf{x}^*$ with an edge of the current quad
5. Clip the path to the nearest intersection, update the timer and current quad, and goto 1.

Despite this relatively simple description, implementation is actually quite tedious. In step 1, we compute the position we would expect the point to end up if the current quadrilateral were infinite. In step 2, we compute bilinear weights for $\mathbf{x}^*$. If the weights are valid for this quad, then we return $\mathbf{x}^*$ in step 3. If the weights are outside $[0, 1] \times [0, 1]$ then the path will leave this quadrilateral. In step 4, we perform edge-edge intersection test with each of the four edges of the quadrilateral and record the closest intersection. In step 5, we update the current quad with the appropriate neighbor and reduce the timer by the amount of time spent traversing this quadrilateral,

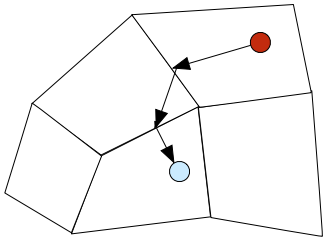

Figure 1: *Process of backward tracing a characteristic during semi-Lagrangian advection. We intersect the ray from the current position (red) to the previous position with all four edges of the quadrilateral. After finding the nearest intersection we repeat the procedure in the next quadrilateral. This process continues until we have used up all the time.*

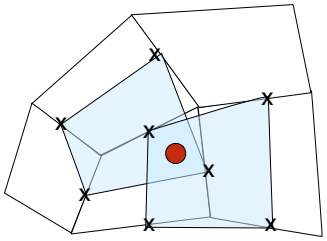

Figure 2: *To fit a velocity at the red point we use the two shifted quads shown in light blue. Edges marked with an "X" are included in the least-squares fit.*

which is easily computed by dividing the distance traveled by the velocity. Finally, we compute the new velocity in the new quadrilateral and continue the tracing process.

One issue that arises in practice is that approximation errors may lead the velocity in the new quadrilateral to point into the previous quadrilateral. In this event, we simply travel along the edge between the two quadrilaterals until we arrive at a vertex.

3.3.2 Velocity Fitting

Because the path traced following the characteristic can end at an arbitrary location of the mesh, we must be able to evaluate the velocity at an arbitrary point, $\mathbf{x}$, in quadrilateral Q. Moreover, because we are going to copy this velocity to a new location, we do not want to incur the additional smoothing. Consequently, we perform a least squares fit to nearby fluxes. Specifically, we use the fluxes stored on the eight edges of two shifted quadrilaterals (see Figure 2). The choice of shifted quadrilaterals depends on the quadrant in which $\mathbf{x}$ lies. We first transform the two-dimensional edge normals into the coordinate system of Q. Then we solve the system

$$\begin{bmatrix} \hat{\mathbf{n}}_0^T \\ \hat{\mathbf{n}}_1^T \\ \cdot \\ \cdot \\ \cdot \\ \hat{\mathbf{n}}_7^T \end{bmatrix} \mathbf{u} = \begin{bmatrix} (\hat{\mathbf{n}} \cdot \mathbf{u})_0 \\ (\hat{\mathbf{n}} \cdot \mathbf{u})_1 \\ \cdot \\ \cdot \\ \cdot \\ (\hat{\mathbf{n}} \cdot \mathbf{u})_7 \end{bmatrix}, \tag{6}$$

for $\mathbf{u}$, where $\hat{\mathbf{n}}_i$ are unit-magnitude normals to the edges and $(\hat{\mathbf{n}} \cdot \mathbf{u})_i$ are the fluxes stored on the edges. Essentially, we seek the velocity that minimizes the least-squares error between predicted (by $\mathbf{u}$) and actual fluxes. We solve this system by forming the normal equations and inverting the resulting 2×2 matrix. This matrix is invertible if the normals, $\hat{\mathbf{n}}_i$, are not identical.

Velocity Fitting at Vertices As noted above, as we trace the path backward along the characteristic we wish to avoid the cost of fitting a velocity many times. Instead, we bilinearly interpolate velocities stored at vertices. These velocities are computed the same as at an arbitrary point, except that instead of using edges determined by shifted quads, we use the edges incident to the vertex in the fit (see Figure 3).

3.3.3 Transforming Variables

At a number of points it is necessary to transform variables from the coordinate system in one quadrilateral to another. These transformations are applied to fit velocities that are to be copied to a dif-

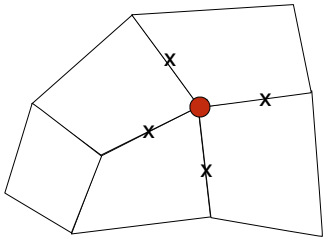

Figure 3: *For a vertex, the incident edges, marked with an "X," are used in the least-squares fit.*

ferent quad and to edge normals to perform the fit. Each quadrilateral is parameterized by two (three-dimensional) vectors (see Section 3.2). Let the source quadrilateral be parameterized by $(\mathbf{e}_0, \mathbf{e}_1)$ and the destination quadrilateral by $(\mathbf{e}'_0, \mathbf{e}'_1)$ then the transformation matrix that takes a vector from the source parameterization to the destination parameterization is

$$T = \begin{bmatrix} \mathbf{e}_0 \cdot \mathbf{e}'_0 & \mathbf{e}_0 \cdot \mathbf{e}'_1 \\ \mathbf{e}_1 \cdot \mathbf{e}'_0 & \mathbf{e}_1 \cdot \mathbf{e}'_1 \end{bmatrix}. \tag{7}$$

Note that this transformation will not generally preserve lengths. To address this issue, we renormalize all transformed vectors to have the same magnitude as before the transformation.

3.4 Pressure Projection

After advection we apply pressure projection to obtain an incompressible velocity field. For this step we require two operators - the divergence operator and the gradient operator. Applying the divergence theorem, we can compute the divergence over a computational cell of a vector field, u, stored on edges of the mesh as

$$(\nabla \cdot \mathbf{u}) = \sum_{e \in edges} (\hat{\mathbf{n}}_e \cdot \mathbf{u}_e) l_e, \tag{8}$$

where $(\hat{\mathbf{n}}_e \cdot \mathbf{u}_e)$ is the flux stored at the edge midpoints and l_e is the length of the edge. Essentially we sum the fluxes along the edges weighted by edge length. This approach gives us an area-weighted divergence estimate. We use the same operator for taking the divergence of the pressure gradient to compute the Laplacian.

For the gradient operator, which is applied to the pressure field, we only require the component of the gradient normal to an edge. For any edge, we divide the difference in pressure values in the adjacent quadrilaterals by the sum of the distances from the edge midpoint to the two quadrilateral centroids. Specifically,

$$(\hat{\mathbf{n}} \cdot \nabla p) = \frac{p_1 - p_0}{\|c_1 - m\| + \|c_0 - m\|}, \tag{9}$$

where p_1 is the pressure in the quadrilateral in the direction of positive flux (see Section 3.1), p_0 is the pressure in the other quadrilateral, c_1 and c_0 are the quadrilateral's respective centroids, and m is the edge midpoint.

The Laplace operator is constructed by composing the divergence and gradient operators allowing us to formulate and solve the well-known Poisson equation

$$\nabla \cdot \nabla p = \nabla \cdot u. \tag{10}$$

We then subtract the gradient of this pressure field from the edge fluxes to arrive at a divergence free velocity field.

3.4.1 Visualization

We provide two mechanisms for visualizing the velocity fields that result from our simulation: level-set interface tracking and density field advection.

Level-set Interface Tracking Perhaps the most compelling and novel of our visualization techniques is level-set interface tracking, which has not previously been applied to fluid simulations on surfaces. We use semi-Lagrangian advection to solve the level-set equation,

$$\phi_t = -\mathbf{u} \cdot \nabla \phi, \tag{11}$$

where ϕ is the level-set function and $\mathbf{u}$ is the velocity field. Our semi-Lagrangian approach closely resembles that for velocity advection, however, because level-set values are stored at vertices of the mesh we can perform bilinear interpolation instead of a least-squares fit. Additionally, periodic redistancing is accomplished using a fast sweeping method and the taxi-cab metric (distance along mesh edges) rather than a more accurate geodesic distance. The taxi-cab metric will always over-estimate distance values. We have not found this overestimation to be problematic in practice.

Density Field Advection Similar to our level-set approach we can also track density values on the mesh, by storing the values at vertices and applying identical semi-Lagrangian advection. In contrast, this approach allows density values to blur during bilinear interpolation. Essentially this approach models fluids that can mix, as opposed to the immiscible fluids better suited to the level-set technique.

4 Results and Discussion

We have run our simulation system with a variety of models, visualization techniques, and forcing functions. In our first example (see Figure 4), a heavy blue fluid, with density 100 kg/m^3, is initialized above a lighter red fluid, with a density of zero, on a "bumpy torus" mesh. As the fluids flow past each the interpolation in semi-Lagrangian advection allows the fluids to mix. In our next example (see Figure 5), a heat source is applied to the bottom of cat mesh. A bouancy force is applied and the temperature advects and diffuses throughout the mesh. When a vertex reaches a threshold temperature it becomes visible in the video. In our next example (see Figure 6), we demonstrate our level-set visualization. We initialize a heavy dark blue fluid above a lighter cyan one, creating a Rayleigh-Taylor instability. As the fluids flow past each other they create the "fingers" typical of the phenomena. Our next example is similar (see Figure 7), but the motion is induced by rotating the mesh. Our final example (see Figure 8) demonstrates our methods ability to handle manifolds with boundaries. The boundaries are treated with free-slip conditions. Timing information is listed in Table 1.

Limitations and Future Work Our current level-set implementation uses the taxi-cab metric, whereas measuring geodesics would improve accuracy. Additionally, the color of each vertex is chosen in a binary fashion based on the sign of the level-set function. This choice leads to aliasing type artifacts. A better approach would be to blend the colors along the interface. Also, it would be nice to decouple the resolution of the level-set or density field from the resolution of the underlying mesh. While this will lead to some artifacts [Losasso et al. 2004], higher resolution level-set or density fields will lead to significantly improved visual results, including less mixing of the density fields.

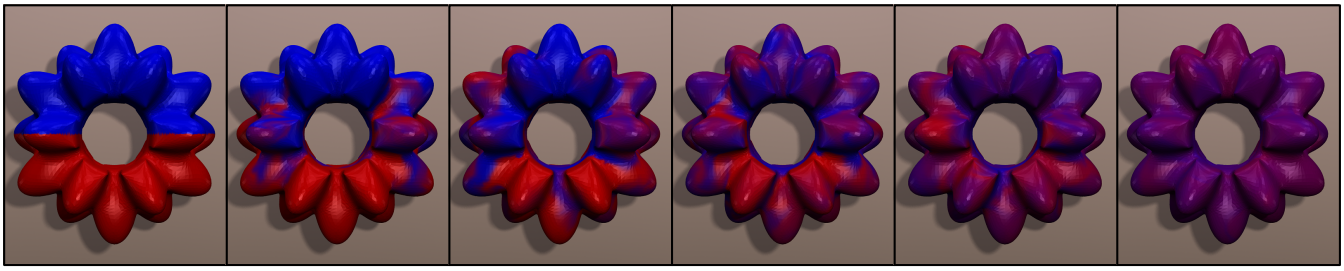

Figure 4: *In this example, a heavy blue fluid begins above lighter red fluid. The fluids mix as they flow past each other.*

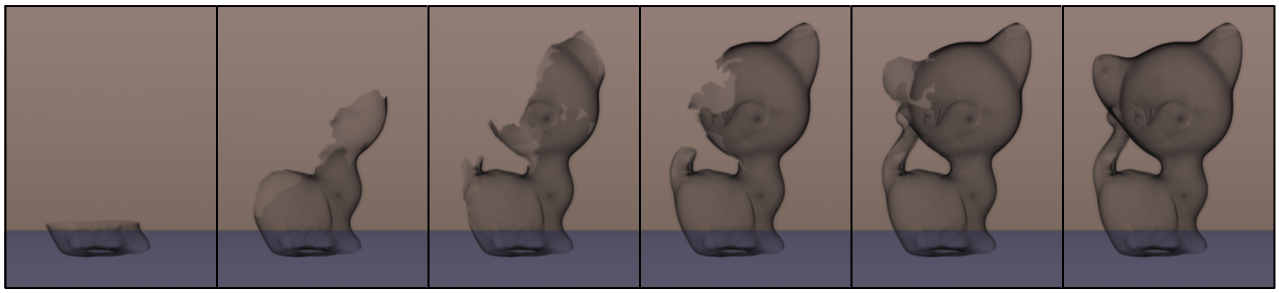

Figure 5: *In this example, a heat source at the bottom of the cat creates a buoyancy force that drives the smoke throughout the cat model.*

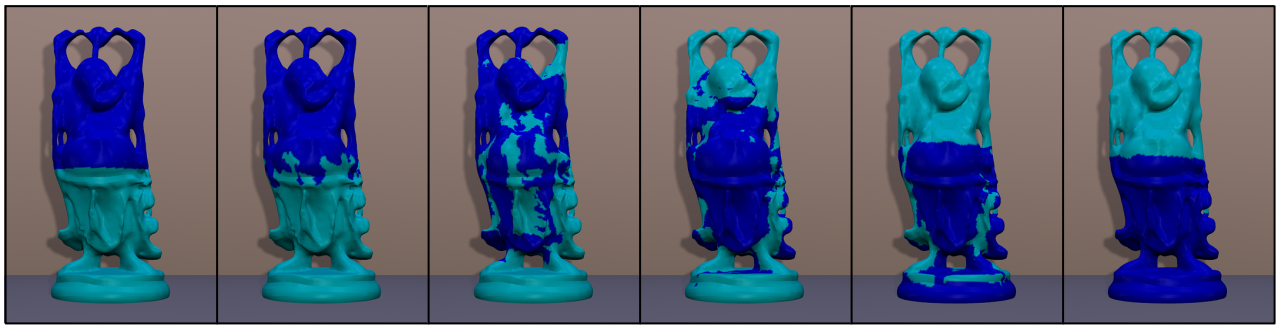

Figure 6: *In this example, a heavy blue fluid is placed above a lighter cyan fluid. Using the level-set interface tracking prevents the fluids from mixing.*

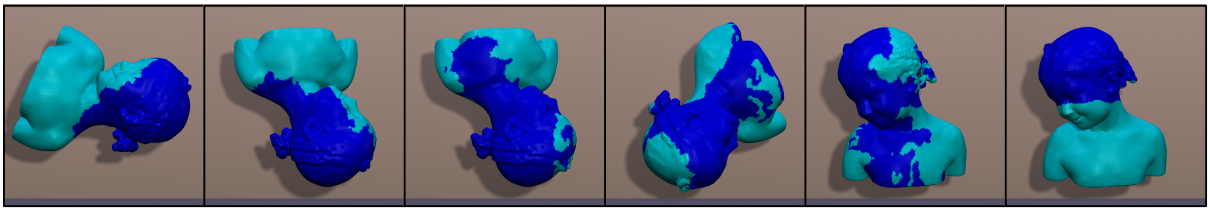

Figure 7: *In this example, the cyan fluid is heavier than the blue fluid. Rotation of the model induces the flow.*

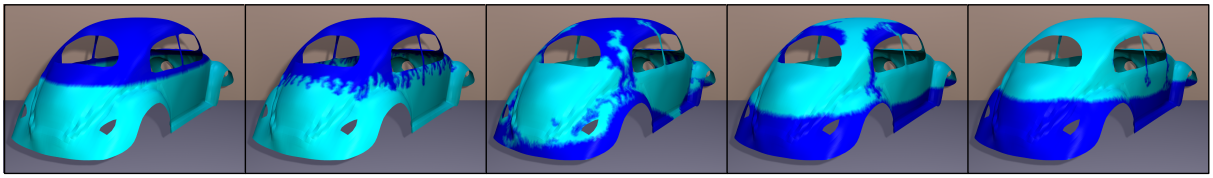

Figure 8: *This example demonstrates our ability to handle manifolds with boundaries.*

Example	# Faces	Steps/Frame	Time/Step	Advection	Pressure
Figure 4	95256	6	11.1698	4.28228	5.4037
Figure 5	115296	2	9.15138	1.70077	6.05998
Figure 6	140608	6	19.3425	3.84929	13.1703
Figure 7	62842	1	4.73444	0.779622	1.87488
Figure 8	94475	1	13.4323	3.50538	8.32156

Table 1: *Timing results (in seconds) for various meshes recorded on a single-core of an Intel Core(TM) i7 CPU processor at 1.67 GHz with 6 GB of memory.*

Finally, we note that one of the main goals of our future work is to better understand the relationship between a simulation and the underlying mesh. Such understanding will guide both simulation and meshing research and foster closer integration of the two. While our experiments have made clear that the underlying mesh affects the simulation we have only just begun to study this question.

Acknowledgments The authors wish to thank the anonymous reviewer for their helpful comments. This work was supported in part by a gift from Adobe Systems Incorporated and National Science Foundation Awards CNS-0855167, IIS-1045032, and IIS-1314896.

References

ALLIEZ, P., UCELLI, G., GOTSMAN, C., AND ATTENE, M. 2005. Recent advances in remeshing of surfaces. research report. *AIM@SHAPE Network of Excellence.*.

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph. 28*, 3.

BOMMES, D., LÉVY, B., PIETRONI, N., PUPPO, E., SILVA, C., TARINI, M., AND ZORIN, D. 2012. State of the art in quad meshing. In *Eurographics STARS*.

BOMMES, D., CAMPEN, M., EBKE, H.-C., ALLIEZ, P., AND KOBBELT, L. 2013. Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics 32*, 4 (July).

CATMULL, E., AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (Nov.), 350–355.

CIFUENTES, A. O., AND KALBAG, A. 1992. A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis. *Finite Elem. Anal. Des. 12*, 3-4, 313–318.

D'AZEVEDO, E. F. 2000. Are bilinear quadrilaterals better than linear triangles? *SIAM Journal on Scientific Computing 22*, 1, 198–217.

DONG, S., BREMER, P.-T., GARLAND, M., PASCUCCI, V., AND HART, J. C. 2006. Spectral surface quadrangulation. *ACM Trans. Graph. 25*, 3, 1057–1066.

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

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. *ACM Trans. Graph. 24*, 3 (July), 904–909.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graphical Models and Image Processing*, 471–483.

GURUNG, T., LANEY, D. E., LINDSTROM, P., AND ROSSIGNAC, J. 2011. SQuad: Compact representation for triangle meshes. *Comput. Graph. Forum 30*, 2, 355–364.

HEGEMAN, K., ASHIKHMIN, M., WANG, H., QIN, H., AND GU, X. 2009. Gpu-based conformal flow on surfaces. *Communication in Information and Systems 9*, 197–212.

LAI, Y.-K., JIN, M., XIE, X., HE, Y., PALACIOS, J., ZHANG, E., HU, S.-M., AND GU, X. 2010. Metric-driven RoSy field design and remeshing. *IEEE Trans. Vis. Comput. Graph. 16*, 1, 95–108.

LÉVY, B., AND LIU, Y. 2010. L_p centroidal Voronoi tessellation and its applications. *ACM Trans. Graph. 29*, 4.

LIU, Y., WANG, W., LÉVY, B., SUN, F., YAN, D.-M., LU, L., AND YANG, C. 2009. On centroidal Voronoi tessellation - energy smoothness and fast computation. *ACM Trans. Graph. 28*, 4.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3, 457–462.

RAY, N., LI, W.-C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic global parameterization. *ACM Trans. Graph. 25*, 4, 1460–1485.

RAY, N., VALLET, B., LI, W.-C., AND LÉVY, B. 2008. N-symmetry direction field design. *ACM Trans. Graph. 27*, 2 (May), 10:1–10:13.

SHEPHERD, J. F., AND JOHNSON, C. R. 2008. Hexahedral mesh generation constraints. *Engineering with Computers 24*, 3, 195–213.

SHI, L., AND YU, Y. 2004. Inviscid and incompressible fluid simulation on triangle meshes. *Computer Animation and Virtual Worlds 15*, 3–4 (July), 173–181.

STAM, J. 1999. Stable fluids. In *The Proceedings of SIGGRAPH*, 121–128.

STAM, J. 2003. Flows on surfaces of arbitrary topology. *ACM Trans. Graph. 22*, 3, 724–731.

TARINI, M., PIETRONI, N., CIGNONI, P., PANOZZO, D., AND PUPPO, E. 2010. Practical quad mesh simplification. *Comput. Graph. Forum 29*, 2, 407–418.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing quadrangulations with discrete harmonic forms. In *Symposium on Geometry Processing*, 201–210.

WANG, H., MUCHA, P. J., AND TURK, G. 2005. Water drops on surfaces. *ACM Trans. Graph. 24*, 3, 921–929.

WANG, H., MILLER, G., AND TURK, G. 2007. Solving general shallow wave equations on surfaces. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer animation*, 229–238.

ZHANG, Y., WANG, H., WANG, S., TONG, Y., AND ZHOU, K. 2012. A deformable surface model for real-time water drop animation. *IEEE Trans. on Vis. and Comp. Graph. 18*, 8 (Aug.), 1281–1289.