

Inviscid and Incompressible Fluid Simulation on Triangle Meshes

Lin Shi

Yizhou Yu

Computer Science, University of Illinois at Urbana-Champaign

201 North Goodwin Avenue, Urbana, IL 61801, USA

email: {linshi,yyz}@uiuc.edu

Simulating fluid motion on manifold surfaces is an interesting but rarely explored area because of the difficulty of establishing plausible physical models. In this paper, we introduce a novel method for inviscid fluid simulation over meshes. It can enforce incompressibility on closed surfaces by utilizing a discrete vector field decomposition algorithm. It also includes effective implementations of semi-Lagrangian tracing and velocity interpolation schemes. Different from previous work, our method performs simulations directly on triangle meshes and thus eliminates parametrization distortions. Our implementation can produce convincing fluid motion on surfaces and has interactive performance for meshes with tens of thousands of faces.

Keywords: Advection, The Poisson Equation, Velocity Interpolation

Introduction

Fluid simulation on manifolds is a challenging problem with many applications. There has been previous work in the physical sciences which deals with fluid flow on a sphere, which can serve as a model for our atmosphere. In graphics, such simulations provide a means to confine fluid flows to a geometric shape and can be used for creating interesting special visual effects, such as objects with elegant dynamic appearances and soap bubbles with swirling diffraction patterns.

Existing physics-based equations for fluid dynamics work

in 3D spaces. Simulating flows on flat surfaces can be considered as a special case of 3D simulation. However, due to the geometric properties of curved surfaces, physically accurate fluid simulations on surfaces are still being investigated by researchers [1]. Our goal in graphics is to develop techniques that can produce visually convincing, physically plausible results. Stam [2] proposed such a solution for Catmull-Clark subdivision surfaces [3]. His method performs fluid simulation in the 2D parametrization space of a curved surface.

In this paper, we provide a solution for triangle meshes which are widely used for representing free-form objects and can serve as the control polyhedra of Loop subdivision surfaces [4]. Our method directly simulates an incompressible and inviscid fluid on a mesh instead of its parametrization space. Because a mesh is a discretization of an underlying smooth surface, our strategy is to use this same discretization or its subdivisions for fluid simulation as well and adapt every step of a previous numerical solution for regular 3D grids to this irregular surface discretization. During such adaptation, we maintain important physical properties of the fluid. We demonstrate that our method not only eliminates distortions and guarantees the zero divergence property, but also produces visually convincing results.

Related Work

The modeling of fluid phenomena has received much attention from the computer graphics community over the last two decades. Recent work in this direction simulates the equations of fluid dynamics. Foster and Metaxas [5] used relatively coarse grids to produce nice smoke motion in three-dimensions. Their simulations are stable only when the time step is sufficiently small. To alleviate this problem and make the simulations more efficient, Stam introduced a new model which is unconditionally stable and could be run at any speed [6]. This was achieved using a combination of a semi-Lagrangian advection scheme [7] and implicit solvers. Fedkiw *et. al.* [8] introduced vorticity confinement and a higher-order interpolation technique. As a result, the simulations can keep finer details on relatively coarse grids. Wei *et. al.* [9] simulated the motion and deformation of lightweight objects in a wind generated by the Lattice-Boltzmann Model.

The focus of this paper is surface flows while the aforementioned fluid simulation techniques are all concerned with volumetric fluids. Thus, the most related previous work in graphics is presented in [2] which introduces a nice technique to solve this problem for Catmull-Clark subdivision surfaces. This method does simulations in the surface parametrization space and alleviates parametrization distortions by incorporating the metric tensor. Though the amount of distortion was reduced significantly, it still exists and is noticeable. There has not been any techniques to completely eliminate such parametrization distortions as far as we know. In addition to distortions, it is also hard to simulate incompressible flows in the parametrization space by enforcing the zero divergence property. On the contrary, the method presented in this paper directly simulates fluids on surface meshes. Our method eliminates distortions and guarantees the zero divergence property.

Background

The Poisson Equation

Originally emerging from Isaac Newton’s law of gravitation, the Poisson equation with Dirichlet boundary condition is formulated as

$$\nabla^2 f = \nabla \cdot \mathbf{w}, \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}, \quad (1)$$

where f is an unknown scalar function, \mathbf{w} is a *guidance* vector field, f^* provides the desirable values on the boundary of Ω , $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ is the Laplacian operator, $\nabla \cdot \mathbf{w} = \frac{\partial w_x}{\partial x} + \frac{\partial w_y}{\partial y} + \frac{\partial w_z}{\partial z}$ is the divergence of $\mathbf{w} = (w_x, w_y, w_z)$. Since the Poisson equation defines the Laplacian of an unknown *scalar* function, solving the equation is actually the process of reconstructing the scalar function from its local differential property.

Vector Field Decomposition

Fluid simulation is closely related to Helmholtz-Hodge vector field decomposition [10] which uniquely exists for a smooth 3D vector field \mathbf{w} defined in a region Ω :

$$\mathbf{w} = \nabla\phi + \nabla \times v + \mathbf{h}, \quad (2)$$

where ϕ is a scalar potential field with $\nabla \times (\nabla\phi) = 0$, v is a vector potential field with $\nabla \cdot (\nabla \times v) = 0$, and \mathbf{h} is a field that is both divergence and curl free. The uniqueness of this decomposition requires proper boundary conditions. The scalar potential field ϕ from this decomposition happens to be the solution of the following least-squares minimization

$$\min_{\phi} \int_{\Omega} \|\nabla\phi - \mathbf{w}\|^2 dA, \quad (3)$$

whose solution can also be obtained by solving the Poisson equation, $\nabla^2\phi = \nabla \cdot \mathbf{w}$.

Discrete Fields and Decomposition

One prerequisite of solving differential equations over a triangle mesh is to overcome its irregular connectivity in comparison to a regular image or voxel grid. One recent approach to circumvent this difficulty is to approximate smooth fields with discrete fields first and then redefine the differential properties and equations for the discrete fields [11, 12]. A discrete vector field on a triangle mesh is defined to be piecewise constant with a constant vector within each triangle. A discrete potential field is defined to be a piecewise linear function, $\phi(\mathbf{x}) = \sum_i B_i(\mathbf{x})\phi_i$, with B_i being the piecewise-linear basis function valued 1 at vertex \mathbf{v}_i and 0 at all other vertices, and ϕ_i being the value of ϕ at \mathbf{v}_i .

For a discrete vector field \mathbf{w} on a mesh, its divergence at vertex \mathbf{v}_i can be defined to be

$$(\text{Div}\mathbf{w})(\mathbf{v}_i) = \sum_{T_k \in N(i)} \nabla B_{ik} \cdot \mathbf{w}|T_k| \quad (4)$$

where $N(i)$ is the set of triangles sharing the vertex \mathbf{v}_i , $|T_k|$ is the area of triangle T_k , and ∇B_{ik} is the gradient vector of B_i within T_k . Note that this divergence is dependent on the geometry and 1-ring structures of the underlying mesh.

Given the definitions of discrete fields and their divergence, the discrete Poisson equation [12] can be expressed as

$$\text{Div}(\nabla\phi) = \text{Div}\mathbf{w}, \quad (5)$$

which is actually a sparse linear system,

$$\mathbf{A}\mathbf{f} = \mathbf{b}, \quad (6)$$

that can be solved numerically using the conjugate gradient method. We still call (5) the Poisson equation for convenience.

Overview

Our method simulates inviscid and incompressible fluids on meshes that are manifold surfaces. The basic equations for such fluids in a 3D space are:

$$\nabla \cdot \mathbf{u} = 0 \quad (7)$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla P + \mathbf{f} \quad (8)$$

A numerical implementation solving the above incompressible scheme consists of the following basic steps [8]:

1. Compute an intermediate fluid velocity field, $\{\mathbf{u}^*\}$, from (8) ignoring the pressure term by first adding external force times the time step, and then solving the advection part $(\mathbf{u} \cdot \nabla)\mathbf{u}$ by using semi-Lagrangian tracing [7];
2. Obtain the pressure P by solving the Poisson equation,

$$\nabla^2 P = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*; \quad (9)$$

where Δt is the size of the time step;

3. Obtain the divergence free component of $\{\mathbf{u}^*\}$ by subtracting the gradient of P from the intermediate velocity,

$$\mathbf{u} = \mathbf{u}^* - \Delta t \nabla P. \quad (10)$$

4. Advection the density field ρ using semi-Lagrangian tracing again.

We adapt these steps from regular 3D grids to meshes with irregular connectivity. Before revising these steps, we need to clarify the discretization scheme on meshes first. Following the definition of discrete fields on meshes, we define a velocity field as a piecewise constant vector field and a pressure or density field as a piecewise linear scalar field. The constant velocity vector within each triangle is defined at the center of the triangle while pressure or density values are defined at the mesh vertices. The pressure or density at a point inside a triangle is linearly interpolated from the values at the three vertices of the triangle using the barycentric coordinates of the point as interpolation coefficients.

With such a discretization scheme on meshes, we summarize the revisions of the above numerical simulation as follows.

- In the first step, we add the acceleration onto the velocity field and then solve velocity advection by implementing semi-Lagrangian tracing on the mesh surface. Details will be introduced in the next section.
- In the second step, to solve the Poisson equation for pressure on meshes, we replace the original equation with the discrete Poisson equation in (5). In the current context, the unknowns in the equation should be pressure values at the mesh vertices. To be consistent, we also multiply the right hand side of (5) with $\frac{1}{\Delta t}$. The solution to this equation is a piecewise linear pressure function defined over the mesh surface.
- In the third step, the original scheme can be followed without any change since there is a constant intermediate velocity vector within each triangle and the gradient of the pressure function is also a constant vector within each triangle. This step enforces incompressibility.
- In the last step, we also need to adapt semi-Lagrangian tracing for density advection, which will also be discussed in the next section.

These revisions produce a working numerical solution for visual fluid simulation on meshes. We will discuss in detail how to perform the advection of velocity and density fields in the next section.

Advection on Meshes

Semi-Lagrangian tracing is the key component of the method we use to perform the advection of both velocity fields and density fields. In a 3D space, if we would like to obtain the advected velocity at a point \mathbf{r} with the reversed velocity direction \mathbf{u}^r , semi-Lagrangian tracing is simply carried out by transporting the velocity at $\mathbf{r} + \mathbf{u}^r dt$ to \mathbf{r} , where dt is the time step.

Performing semi-Lagrangian tracing on meshes is more complicated. Since the velocity of a triangle is defined at its center, velocity tracing always starts from the center of a triangle. On the other hand, since density (scalar) values are defined on vertices, we have to start tracing from there. However, the reversed velocity direction at a vertex is not well defined without a careful interpolation scheme. In this section, we first describe a method to advect velocities, and then generalize it to solve the advection of density (scalar) fields.

Advection of Velocity Fields

In a 3D space, the destination of semi-Lagrangian tracing is still in the same space. However, on a curved mesh surface, $\mathbf{r} + \mathbf{u}^r dt$ is not necessarily on the same surface any more. To overcome this difficulty, the key point is to generate a curved trajectory on the mesh by wrapping the tracing direction around the surface and treating the shared edge of two adjacent faces properly when the trajectory crosses it from one of the faces. It is natural to require that the trajectory continue in the adjacent face from its intersection with the shared edge.

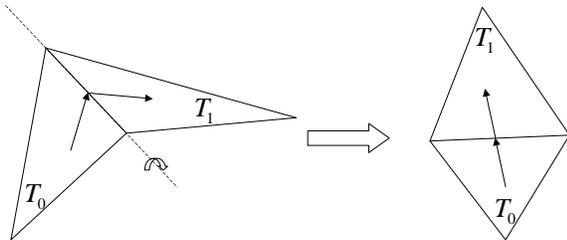


Figure 1: Flattening faces T_0 and T_1 : rotate face T_0 to the same plane as T_1 , and rotate the velocity vector using the same matrix operation.

How can we define the tracing direction on the adjacent face to guarantee continuity? The situation is shown in Fig-

ure 1. We first need to define the continuity of vector fields across faces. The vector field on two adjacent faces is considered to be continuous at their shared edge if the resulting vector field from the following operation is continuous at the shared edge: the operation simply rotate one of the faces, as well as the vectors on it, along the shared edge onto the plane where the other face resides. According to this definition, it is straightforward to define the tracing direction on the adjacent face. Note that there are two possible choices in the above rotation. More reasonably we use the one that makes two faces end up on different sides of the edge. This guarantees that our algorithm degenerates into the original fluid algorithm for 2D spaces when the mesh is actually representing a flat surface. And this is an important issue when adapting the original algorithm for meshes. We call it degeneracy concern.

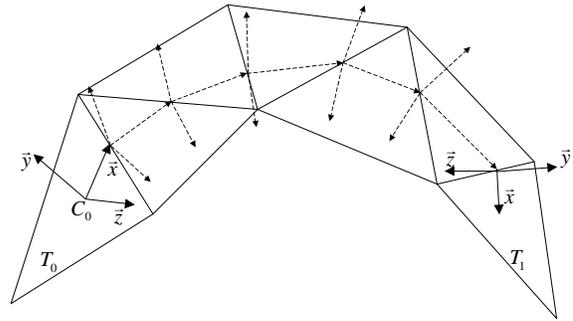


Figure 2: The velocity vector at the end of a traced trajectory in face T_1 is transported to the beginning of the trajectory at the center of face T_0 . The x -axis of the local frame in T_0 is the starting tracing direction.

Once the end of the trajectory has been reached, the velocity there needs to be transported to the point where we started tracing. In a 3D space, velocity vectors can be simply translated. However, on a mesh surface, a translated vector will not necessarily belong to the same surface any more. In addition, a curved surface change its orientation constantly. Translated vectors cannot reflect this orientation change. Our solution to this problem is based on local frames defined in the starting face and the terminating face of the trajectory. It is illustrated in Figure 2. We define a local frame inside a triangle face with respect to a tracing direction as follows. The unit vector along the specific tracing direction in the face is the x -axis. The normal vector of the face is the y -axis. The

z -axis is simply the cross product between the x - and y -axes. A mapping between vectors in the two faces can be defined. A vector in the second local frame is mapped to the vector in the first frame with the same local coordinates. We use this mapping to transport desirable velocity vectors from the terminating face of the trajectory to the starting face. Note that both the local frames and mapping change when the tracing direction changes. This scheme also degenerates into vector translation in 2D when the mesh is actually representing a flat surface.

Velocity Interpolation

Note that in this paper, a velocity field is represented as a piecewise constant vector field. The velocities at triangle edges are not continuous. Therefore, the velocity at the end of the traced trajectory should be first smoothly interpolated before being transported to avoid visual discontinuities and other artifacts. Before introducing our general interpolation scheme, let us first look at how to interpolate the velocity at a vertex.

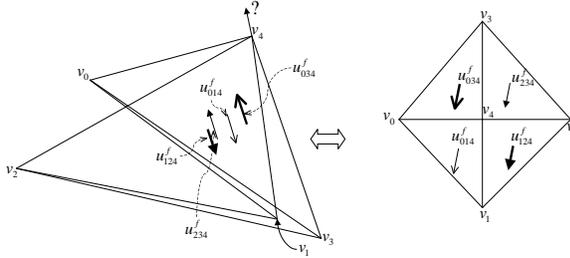


Figure 3: Velocity interpolation at a vertex whose neighboring faces are formed by folding a flat surface.

The first choice we tried is to perform vector interpolation in the 3D space. Unfortunately, we found it to be problematic. Consider the situation in Figure 3, where we would like to interpolate the velocity at v_4 . The result from simple 3D interpolation should be close to zero. But this conflicts with the fact that neighboring faces of v_4 form a flat surface when being unfolded and in that plane all velocities are the same. Naturally, the velocity at v_4 is expected to be the same, too.

To correctly interpolate the velocity at a vertex, what we really need is to flatten its 1-ring neighborhood first. In this paper, local flattening is achieved with affine mappings (parametrizations). In Figure 4, we show how vector f is

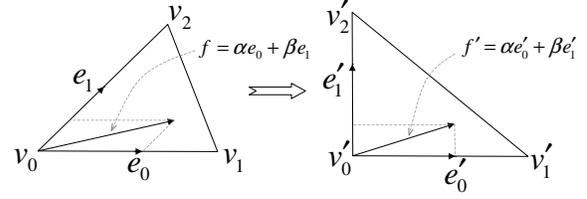


Figure 4: A local affine transformation defined by mapping two basis vectors e_0 and e_1 .

mapped when the triangle it lies on is transformed to a different shape. Suppose unit vectors e_0 and e_1 collinear with two edges of the triangle shown in the figure are transformed to e'_0 and e'_1 . If we consider e_0 and e_1 as the basis vectors of an affine frame, the transformed vectors e'_0 and e'_1 define a new affine frame. An affine transform between the original and new affine frames can be easily derived. This affine transform can be applied to any vector defined in the original frame to obtain its corresponding new vector in the new frame. Meanwhile, the inverse affine transform is also well-defined so a vector can be mapped from the new frame back to the original frame.

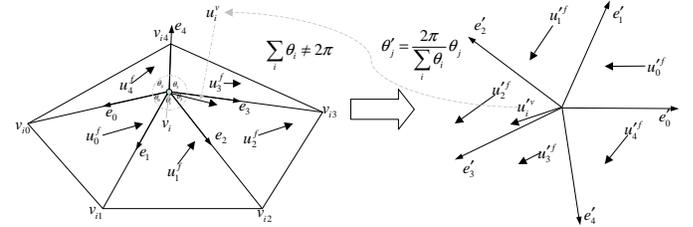


Figure 5: Vertex velocity interpolation in a local parametrization plane.

With these local affine mappings, velocity interpolation at a vertex proceeds as follows. We perform the unfolding process as in Figure 5. We map the unit vectors pointing from v_i towards its neighboring vertices onto a unit circle lying on a parametrization plane such that the relative proportion of their angles is preserved. A local affine transform can be derived for each face adjacent to v_i using these mapped unit vectors. Then the velocity vector at each face center is mapped to a new vector on the parametrization plane using the derived affine transform for the face. The weighted average of the transformed velocity vectors is the average velocity at vertex v_i in the parametrization plane. The angle of a face at this vertex is used as the weight. Note that there is a dis-

tinct affine transform for each face, the average velocity in the parametrization plane cannot be uniquely mapped back onto the mesh.

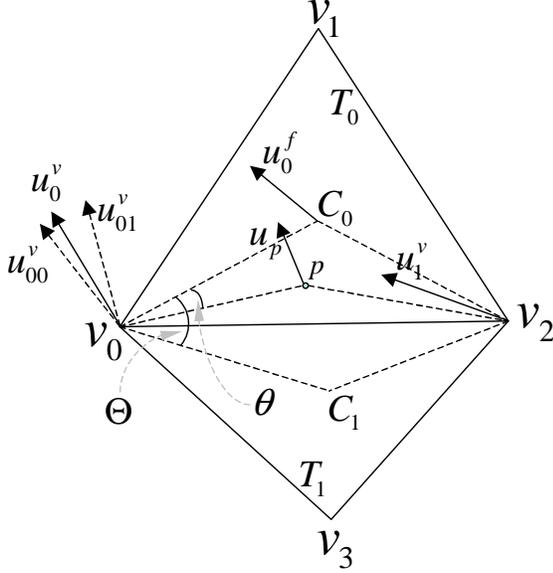


Figure 6: Velocity interpolation within a triangle.

Our general velocity interpolation scheme is illustrated in Figure 6. Assume we are interpolating the velocity at point \mathbf{p} on face T_0 whose vertices are \mathbf{v}_0 , \mathbf{v}_1 , and \mathbf{v}_2 . Also assume \mathbf{p} is within the triangle formed by \mathbf{v}_0 , \mathbf{v}_2 and \mathbf{c}_0 which is the center of face T_0 . We use the interpolated velocity of \mathbf{v}_0 , \mathbf{v}_1 (mapped back onto face T_0) and the face velocity of T_0 at its center to interpolate that of \mathbf{p} . Naturally, the barycentric coordinates of \mathbf{p} with respect to this smaller triangle are used as the averaging weights. To ensure the continuity of interpolated velocity across the shared edge of two adjacent faces, this scheme needs to be enhanced.

Suppose T_0 and T_1 are two adjacent faces, and \mathbf{v}_0 and \mathbf{v}_2 are the vertices of their shared edge. If we flatten the 1-ring structure at \mathbf{v}_0 , there are two distinct affine mappings associated with T_0 and T_1 , respectively. Vertex \mathbf{v}_0 has an average velocity in its local parametrization plane. The inverse affine transform associated with T_0 maps this average velocity back to a vector \mathbf{u}_{00}^v coplanar with T_0 . A vector \mathbf{u}_{01}^v coplanar with T_1 can also be generated similarly. In general, \mathbf{u}_{00}^v and \mathbf{u}_{01}^v are not the same even if we flatten T_0 and T_1 into a plane along their shared edge. We decided to linearly interpolate between these two vectors to achieve continuous interpolated velocity at the shared edge. Suppose the center of T_0 is \mathbf{c}_0 and

the center of T_1 is \mathbf{c}_1 . Let θ be the angle between lines $\mathbf{v}_0\mathbf{p}$ and $\mathbf{v}_0\mathbf{c}_0$, Θ be the angle between lines $\mathbf{v}_0\mathbf{c}_1$ and $\mathbf{v}_0\mathbf{c}_0$ after flattening. The interpolation coefficient between \mathbf{u}_{00}^v and \mathbf{u}_{01}^v is defined to be $\frac{\theta}{\Theta}$. Thus, the interpolated velocity at \mathbf{v}_0 is

$$\mathbf{u}_0^v = \left(1 - \frac{\theta}{\Theta}\right) \mathbf{u}_{00}^v + \frac{\theta}{\Theta} \mathbf{u}_{01}^v. \quad (11)$$

The interpolated velocity \mathbf{u}_2^v at \mathbf{v}_2 is defined similarly. The final interpolated velocity \mathbf{u}_p at \mathbf{p} is

$$\mathbf{u}_p = \alpha \mathbf{u}_0^v + \beta \mathbf{u}_2^v + (1 - \alpha - \beta) \mathbf{u}_0^f \quad (12)$$

where \mathbf{u}_0^f is the face velocity at the center of T_0 , and $(\alpha, \beta, 1 - \alpha - \beta)$ represent the barycentric coordinates of \mathbf{p} inside the triangle formed by \mathbf{v}_0 , \mathbf{v}_1 and \mathbf{c}_0 .

Advection of Scalar Fields

At the last substep of each time step, the scalar density field is advected and interpolated. This part only differs from the advection of the velocity field in two aspects. For a density field defined on vertices, it is not obvious from which face to start the tracing and with what velocity. We use the average vertex velocity in its parametrization plane to determine the face where the velocity vector lies and map the average velocity back onto it. And the rest of tracing is exactly the same as that of velocity field. Unlike velocity advection, we do not need to transport vectors and perform vector interpolation. Density interpolation at the end of the trajectory is much simpler since density values are defined on vertices instead of face centers. We only need to use the barycentric coordinates of the endpoint of the trajectory with respect to the vertices of the terminating face to interpolate from the density values at those vertices. The interpolated density is deposited onto the vertex where the tracing started.

Implementation

The sparse matrix equation in (6) can be solved using either conjugate gradient (CG), or preconditioned conjugate gradient (PCG). In the case of PCG, we use the Incomplete Cholesky Factorization (ICF) as in [8].

To handle meshes with open boundaries, a few places in the implementation need to be modified. When solving the Poisson equation for pressure, the pressure values on the

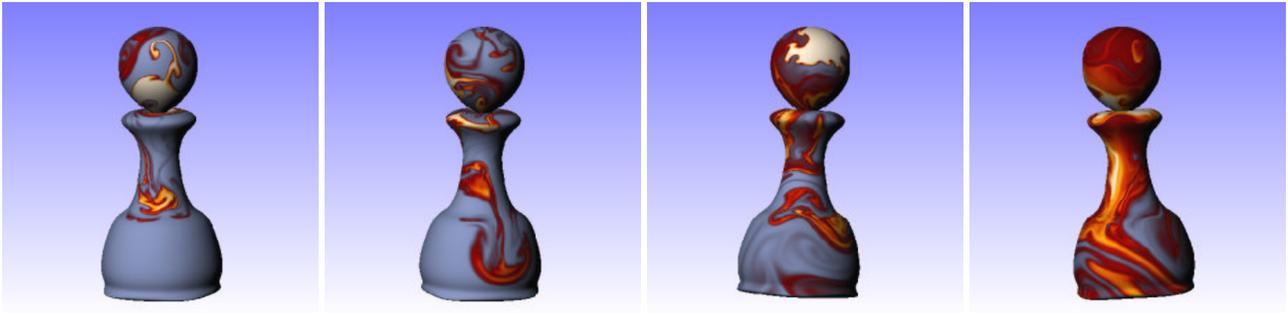


Figure 8: Density is periodically deposited onto the top of a Pawn model, and gravity drives the rest of the simulation. The mesh has 130050 vertices and 260096 faces.

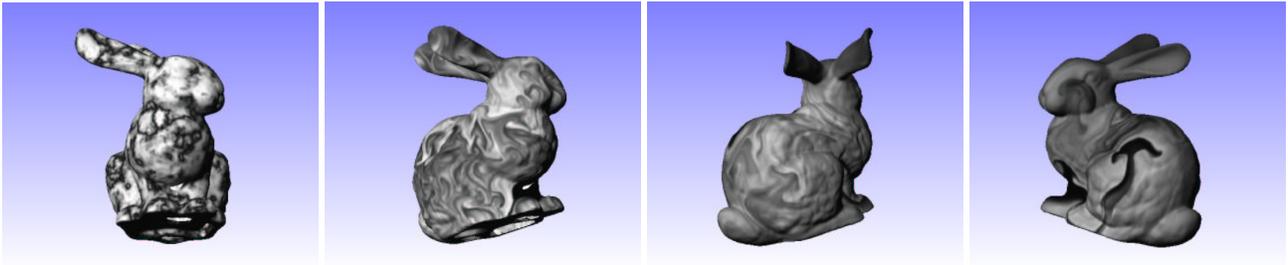


Figure 9: Fluid flows on a Bunny model. The mesh has 139122 vertices and 277804 faces.

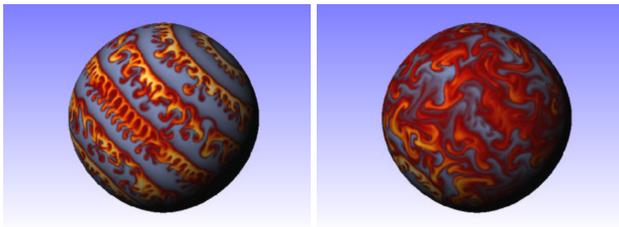


Figure 7: Fluid flows on a Sphere model with a striped initial density distribution. The mesh has 98306 vertices and 196608 faces.

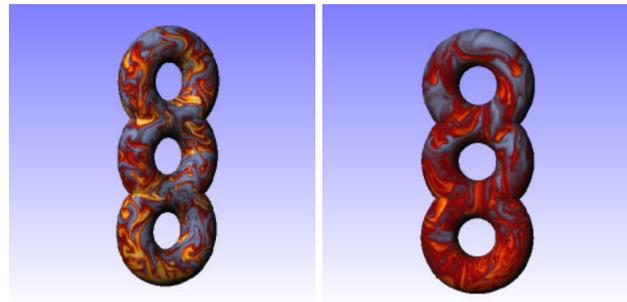


Figure 10: Fluid flows on a high genus Torus model. The mesh has 271356 vertices and 542720 faces.

mesh boundaries are actually the boundary condition for the Poisson equation. In practice, we use Dirichlet boundary conditions and set the pressure to be 0 at boundary vertices. The coefficient matrix in (6) needs to be adjusted to accommodate this type of boundary conditions. When setting up the local affine transforms during velocity interpolation at boundary vertices, we keep the original angles when they are mapped into the parametrization plane unless the summation of all the angles surrounding a vertex exceeds 2π , which in practice we did not notice any. Semi-Lagrangian tracing can go beyond the mesh boundary, in which case smoothly decreasing the

density helps prevent undesirable discontinuities. Note that our current implementation does not enforce zero divergence at the boundary vertices.

Experimental Results

Listed here are the results of several examples. In the Pawn example shown in Figure 8, we periodically deposit density

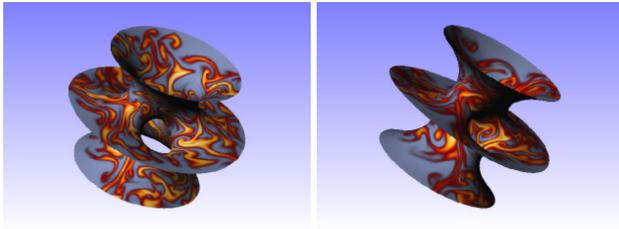


Figure 11: Fluid flows on the Hypersheet model. The mesh has 117877 vertices and 234752 faces.

onto the top of it, and let gravity drive the rest of the simulation. The model is derived from an original Pawn model after five iterations of Loop subdivision. If we run our program on the original coarse Pawn model with 16256 faces, we can achieve a frame rate of 6fps on a 1.7GHz Pentium IV Xeon processor. Because of incompressibility, the fluid moves in the opposite direction of gravity in some regions. In terms of the bunny model shown in Figure 9, we initially place density as horizontal strips, and use gravity to pull them down. Note that the bunny model is not a waterproof model and has some holes on the bottom and between the forelegs. The fluid moves interestingly around these regions. The fluid behavior at open mesh boundaries is even more clear in the hypersheet model shown in Figure 11 where the fluid escapes when crossing the boundary. We use Dirichlet boundary condition for both examples. Note that it is also a straightforward implementation to reflect the fluid velocity vector once it reaches the boundary. In the triple torus example shown in Figure 10, we show our method works equally well for high genus models.

The videos for the examples in this paper may be found at:
http://www-sal.cs.uiuc.edu/~yyz/research/surface_flow

Discussions

In this paper, we described a novel fluid simulation method directly performed on triangle meshes. It eliminates parametrization distortions and enforces incompressibility on closed surfaces. This work can be potentially generalized to other types of manifold meshes and even unstructured 3D grids. So far our algorithm works for inviscid and incompressible fluid simulation. For a more complicated and realistic fluid solver, further studies of the underlying physics

foundation is essential. We would also like to improve the performance of our solver especially by using an appropriate preconditioner for the conjugate gradient method.

References

- [1] M. Bertalmio, L.T. Cheng, S. Osher, and G. Sapiro. Variational problems and partial differential equations on implicit surfaces. *Computational Physics*, 174(2):759–780, 2001.
- [2] J. Stam. Flows on surfaces of arbitrary topology. *ACM Transactions on Graphics*, 22(3):724–731, 2003.
- [3] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Univ. of Utah, 1974. Report UTEC-CSc-74-133.
- [4] C. Loop. Smooth subdivision surfaces based on triangles. Master’s thesis, University of Utah, Department of Mathematics, 1987.
- [5] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH 97 Conference Proceedings*, pages 181–188, 1997.
- [6] J. Stam. Stable fluids. In *SIGGRAPH 99 Conference Proceedings*, pages 121–128, 1999.
- [7] A. Staniforth and J. Cote. Semi-lagrangian integration schemes for atmospheric models: A review. *Monthly Weather Review*, 119:2206–2223, 1991.
- [8] R. Fedkiw, J. Stam, and H.W. Jensen. Visual simulation of smoke. In *SIGGRAPH 01 Conference Proceedings*, pages 15–22, 2001.
- [9] X. Wei, Y. Zhao, Z. Fan, W. Li, S. Yoakum-Stover, and A. Kaufman. Blowing in the wind. In *ACM Symposium on Computer Animation*, 2003.
- [10] R. Abraham, J. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 75. Springer, 1988. Applied Mathematical Sciences.
- [11] K. Polthier and E. Preuss. Variational approach to vector field decomposition. In *Proc. Eurographics Workshop on Scientific Visualization*, 2000.
- [12] Y. Tong, S. Lombeyda, A.N. Hirani, and M. Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graphics*, 22(3):445–452, 2003.