AN ABSTRACT OF THE THESIS OF

Patrick Neill for the degree of Master of Science in Computer Science presented on June 5, 2008.

Title: Fluid Flow on Interacting, Deformable Surfaces

Abstract approved: ______________________________

Eugene Zhang

Fluid simulation is an interesting research problem with a wide range of applications including mechanical engineering, special effects in movies and games, and scientific simulation. Due to the complex nature of typical fluid flow equations, there are circumstances where a full volumetric fluid simulation may not be necessary to generate the desired effect. Fluid flow on surfaces, such as in the case of rain-drops or moving rivers, can be solved more effectively by using a surface simplification to the normally expensive 3D Navier-Stokes equations. We present such a system in which the user can guide fluid flow on surfaces that are not only deforming, but also colliding with other surfaces in an environment. We also describe a technique for rendering the fluid on surfaces as a height field, which allows nearly volumetric effects to be achieved through a computationally less expensive surface simulation. Such a framework, we believe, can be extended to allow interactive control and visualization of surface flows carving into surfaces.

Fluid Flow on Interacting, Deformable Surfaces

by

Patrick Neill

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 5, 2008
Commencement June 2009

ACKNOWLEDGEMENTS

I would like to thank all the students in the Graphics Lab at Oregon State University for assisting in my excellent education over the years. I would especially like to thank Eugene Zhang for all the thorough discussions we had as we worked through the material presented in this thesis. To Ron Metoyer for getting me into research as an undergraduate, and to Mike Bailey for his never-ending enthusiam for everything graphics related. Two years is too short of a time to be working with such great people!

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

DEDICATION

To my family for supporting and inspiring me, and to Sierra for patiently listening to me when I became especially excited about a math topic.

Chapter 1 – Introduction

The physical simulation of fluid is a difficult problem! Näive implementations of fluid simulation can be programmed in under a day. In stark contrast, a highly accurate and accelerated fluid simulator can be the topic of research and development for years. Implementing this type of simulation requires a wide breadth of knowledge. For example, an in-depth understanding of differential equations describing the physical motion of fluid is a problem a physicist or mechanical engineer can handle. Careful numerical considerations are also required in order to ensure numerical error doesn't propagate exponentially through the simulation; topics mathematicians are suited for. A computer scientist, on the other hand, is best equipped to handle the rendering, visualization, acceleration, and control of the fluid.

The focus of this thesis is to provide a background and material study on a specific sub-problem in fluid dynamics: fluid flow on deforming, colliding surfaces. The majority of fluid simulators operate on either a planar or volumetric domain, i.e. in two or three dimensions. An example of a two dimensional simulation is shown in Figure 1.1. Space is discretized according to finite differencing methods, which are used to solve the differential equations describing the fluid flow. Additional constructs such as particle tracing or higher order interpolation methods are also defined in order to assist with accuracy and efficiency. The work we present in

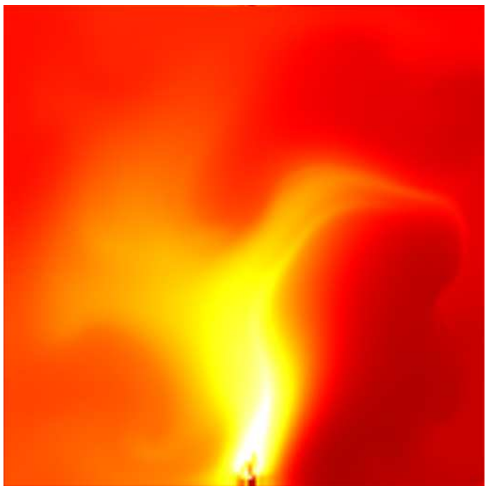


Figure 1.1: A planar simulation of fluid dynamics (Source: Author).

this thesis deals with fluid flow on deforming surfaces. We feel that restricting the domain of computation to a surface will achieve attractive properties for certian special effects especially suited to that domain. As primary motivation for our research, many physically interesting fluid phenomena can be intuitively simplified to a deforming, surface-like representation. Examples of such physical behavior are large-scale oceans and other slow fluid flows, rain-drops flowing down surfaces, paintings on flexible surfaces, and dynamically flow-like deformations.

Fluid flow on surfaces may also present an attractive, relatively inexpensive alternative to computing full volumetric simulations. Volumetric simulations of fluids, while producing a wide range of fluid-like effects, traditionally suffer from a lack of control and interactivity. There is also the difficult question of how to deal with the complex interface between a voxelized volume and non 'grid' aligned geometry. A planar simulation, on the other hand, offers the user an interactive environment to design and control fluid flow. Planar simulations, however, are too restricted to be used in environments where underlying computational domains are non-planar. Fluid flow on surfaces may offer a compromise between the planar and volume case, achieving effects that are not achievable in planar simulations, or effects that require complex interface handling in volumes. Fluid flow on a dynamic, deforming medium also presents an interesting physical and numerical challenge. How does the deformation effect the flow of the fluid? How does the flow of the fluid effect the deformation of the surface? What are the numerical issues associated with a dynamically changing computational domain?

The set of problems related to the flow of a fluid on deformable surfaces is not simple. One has to deal with an irregular discretization over the surface. To emphasize interactivity, the collection of partial differential equations of Navier-Stokes have to also be solved over the surface in a fraction of a second. Adding deformation into the system causes the differential properties of the surface to dynamically change, which prevents important pre-computation present in other fluid simulators. One also has to deal with the ambiguity associated with vectors lying in different tangent planes, a classic differential geometry problem of lacking global parameterizations. Since the surface is embedded in three dimensions, surface-to-surface interaction in the form of collisions or transference of fluid is another topic the system needs to address. Functionalities that allow the control and design of fluid and deformation also need to be implemented. This thesis discusses how to couple these features together in a comprehensive, interactive framework. An example of an image generated by this system is shown in Figure 1.2.

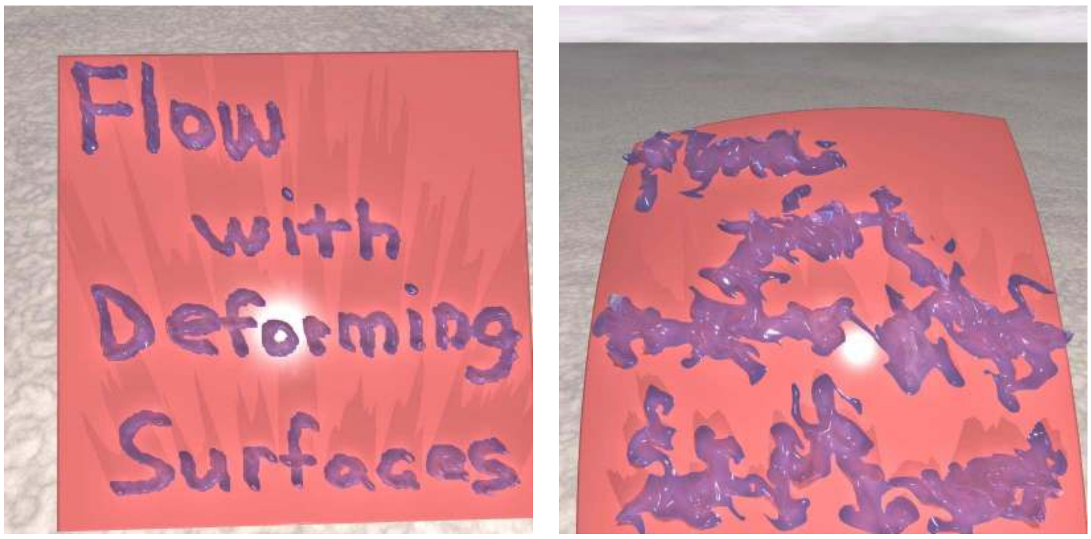


Figure 1.2: Fluid flowing on a deformable surface. Even though the fluid appears volumetric, fluid flow is directly computed on the triangular surface.

Therefore, this thesis presents the following contributions.

1. A unified framework and a system for the simulation of fluid flows on deformable and interacting surfaces. To the best of our knowledge, this is the first time such a system has been developed.

2. At the core of this system is a numerical fluid solver that not only is stable during surface deformation, but also supports efficient computation such as performing diffusion, Helmholtz-Hodge decomposition, and Semi-Lagrangian advection on surfaces.

3. We discuss vital ideas such as geodesic polar maps and parallel transport from classical differential geometry and apply them to the solution of fluid flow on surfaces.

4. We provide a set of functionalities that allows the user to design and control not only the density and velocity of the fluid but also the motion and deformation of the underlying surface.

5. We present various visualization techniques which assist in the control and design of the fluid and deformation of the surface. As a proof-of-concept for emulating volumetric surface fluids, a high quality post-process rendering step is also presented which gives the illusion of volumetric flow on surfaces.

Chapter 2 – Literature Review

2.1 Conventional Fluid Dynamics

Solving various forms of the Navier-Stokes equations has been a thoroughly researched problem. Foster and Metaxas [13] provide a finite-differencing approach with explicit integration steps to simulate gas motion in a volume. Their use of explicit integration introduces numerical instability during advection and diffusion for large time steps. Later Stam [43] showed that through the use of implicit integration for diffusion and semi-Lagrangian advection, the previous time-step constraint can be eliminated at the cost of introducing numerical dissipation typically present in implicit numerical methods.

Ensuring fluid incompressibility is typically done through Helmholtz-Hodge Decomposition. Restricting the computational domain to a regular grid of same-sized cells makes Helmholtz-Hodge decomposition and semi-Lagrangian tracing easy to define. It is much more difficult for triangular or tetrahedral meshes due to irregular tessellation. Polthier and Preuss [28, 29] define Helmholtz-Hodge decomposition of discrete vector fields on arbitrary surfaces. Their work was later extended by Tong et al. [49] to irregular 3D tetrahedral volumes. Our model of Helmholtz-Hodge Decomposition on surfaces follows the work found in [49] in a straightforward fashion.

Evolving the front of a fluid accurately requires precise tracking and correction to achieve realistic fluid-to-air interactions. Massless marker particles are typically used with an implicit surface representation in order to achieve a high order of numerical accuracy [12, 9]. Losasso et al. [24] provide an effective way of handling multiple front interactions between fluids of different compositions. Intelligent particle placement algorithms also help with preserving certain features of the fluid like vortices [36, 14]. By combining a height field underneath the surface with a full volumetric simulation near the surface front, Irving et al. [17] are able to concentrate computational time on regions most visible by the user. Resolving the interface between dynamically moving objects and fluids was recently addressed by Carlson et al. [6].

2.2 Surface-based Fluid Dynamics

Work describing fluid flow on surfaces has been less covered than volumetric simulations. The primary problem with fluid flow on arbitrary surfaces is that the semi-Lagrangian advection and diffusion are more difficult to define. Stam [44] originally proposed the use of Catmull-Clark subdivision surfaces in order to transform the Navier-Stokes equations to a parameterized domain where all tangent planes are continuous. This effectively eliminated the ambiguity associated with vectors defined on different tangent planes, allowing tracing and diffusion to occur identically to the planar case. The use of the Catmull-Clark subdivision surfaces allowed the Navier-Stokes equations to be solved on a surface, but not without noticeable artifacts due to the distortion introduced by surface parameterization, and the extra cost incurred by working in such representations. While their technique did much to reduce distortions through the use of a deformation metric, artifacts were still present, specifically on the boundaries between patches.

Shi and Yu [37] presented an idea of solving the Navier-Stokes equations directly on the mesh without the use of special surfaces. The paper makes use of Geodescic Polar maps, used also by [54, 30, 22, 59], to transport vectors defined on vertices to incident triangles. Shi and Yu [37] also presented an interesting advection method, where the curved mesh is essentially rotated into a flat planar representation in the direction of the tracing.

Wang et al. also presented a framework similar to ours, but used a generalized version of the Shallow Wave Equations instead of Navier-Stokes [52]. Their work used a completely implicit integration scheme, ignored velocity advection and diffusion, and incorporated external forces, surface tension, and rigid body interactions from earlier work.

Fluid design and control is important to animators who would rather deal with the high level control rather than low level variable tweaking. Patrick Witting presents an approach on incorporating fluid dynamics into a tradional animation environment [55]. McNamara et al. [25] make use of the adjoint method and emphasize control over the fluid surface through minimizing objective functions. Shi and Yu discuss controlling smoke with objects [38], and creating force fields to influence the fluid into certain shapes or animations [39].

2.3 Deformation in Computer Graphics

The field of deformation can be classified into a few different categories, each with distinct benefits and disadvantages. Since deformation is not the primary contribution of this thesis, interested readers can refer to [27] for a more complete introduction to the field.

Non-physically based methods focus on controlling and designing qualities about the object, and emphasize the preservation of certain features of the surface or volume while the shape is undergoing deformation. Some existing techniques [51, 15] that preserve volume and provide control mechanisms that make deformation design simpler. An intuitive deformation controller based on Inverse Kinematics is presented in [46]. A technique that preserves surface features such as distance-preserving or angle-preserving while the model undergoes shape interpolation or extrapolation is presented in [20]. Finally, animated deformation transfer between two vaguely related animationed shapes is demonstrated by Sumner et al. [45].

Performance based methods focus on acquiring a plausible, not necessarily physically based deformation in real-time. Primary applications include interactive applications such as games or educational tools. Typically these techniques are also merited by their ease of implementation when compared to other classes of methods. Mass-spring systems found in common literature simulate deformation-like behavior at real-time speeds. They do, however, suffer from numerical instability and generally non-plausible deformations. By combining the intuition gained from mass-spring systems with a process known as shape matching, Matthias et al. and Rivers et al. [26, 34] eliminate numerical instability and incorporate non-linear deformations in a real-time, easily-controllable framework. [19] provides an interesting approach where deformations of a model are described by a hierarchical tree structure, with physical forces being applied to the bounding geometry rather than the actual model.

Physically based methods usually use a finite element method in which the model undergoing deformation is volumetrically discretized into tetrahedra. Physical descriptions of the deformation are usually coupled with volume preservation constraints solved by energy minimization to generate the most physically realistic, albeit slowest, class of deformations. Fundamental literature on elastic deformations based on finite differencing and finite elements is described in [47]. Finite element methods for representing different types of deformations are seen in [18, 3]. When large deformations occur, the model may become inverted due to extreme forces or numerical error accumulation. Irving et al. [16] demonstrate a finite element method which can handle such mesh inversions. Finally, Wu et al. [56] show that nonlinear finite element methods can be computed interactively, if the simulation concentrates on features which are identified by progessive meshing.

Chapter 3 – Background

3.1 Fluid Dynamics

The background of knowledge required in computational fluid dynamics is indeed large. There are a variety of methods available, each with advantages and disadvantages. For the sake of clarity and conciseness, I will only present simple solutions to the required equations. A short introduction to fluid dynamics on an evenly discretized domain is presented here.

3.1.1 Navier-Stokes Equations

To be able to simulate how an incompressible constant density fluid flow changes over time, we will need equations describing the differential behavior of the system. There are a collection of Partial Differential Equations(PDEs), known commonly as the Navier-Stokes equations, which describe this behavior over time:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \frac{1}{\rho}\nabla P = \nu \nabla \cdot \nabla \vec{u} + \vec{f} \tag{3.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{3.2}$$

Each term has a fairly intuitive solution, at least for the purposes of this thesis. A good starting point is to first deal with each term individually. $\vec{u}$ represents the velocity of the fluid, so $\frac{\partial \vec{u}}{\partial t}$ represents the change in the fluid's velocity over time. This term is what we will be eventually using to advance our simulation during each time-step.

The second term, $(\vec{u} \cdot \nabla)\vec{u}$, is usually called advection, transport, or convection. The form of this equation is a bit of an abuse of notation (what is the dot product with the gradient operator anyways?), but will be explained in Section 3.1.3. Combining this term with the first term gives us $\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u}$, which is usually called the material derivative, convective term, Lagrangian, Stokes, or particle derivative.

$\frac{1}{\rho}\nabla P$ represents the forces that high pressure regions of the fluid exert on low pressure regions. P represents the potential of these forces, while ρ represents the density of the fluid or gas. Incompressible fluids aggressively attempt to maintain constant pressure, as is implied in the name. Visually, this manifests itself as forces applied to the fluid that force the conservation of mass and momentum. We will be using this term in order to correct the velocity field so that mass or energy, ideally, is not introduced or taken away from the simulation. Due to numerical issues, enforcing the conservation of mass turns out to be a rather tricky task. This step is typically called pressure projection, and is discussed in Section 3.1.5.

The diffusion term, $\nu \nabla \cdot \nabla \vec{u}$, is a term usually left out in most fluid simulators. The constant ν, called the kinematic viscosity, is defined as the dynamic viscosity divided by the density of the fluid, or μ/ρ. The term $\nabla \cdot \nabla$, also written as ∇^2 or Δ, is known as the Laplacian or Laplace operator and represents the tendency of fluid flow at a point to move at the same speed as the fluid around it. Think of it as an averaging, or smoothing operator. A high viscosity correlates to a fluid like oil or molasses, while a low viscosity allows for fluids like water or air. Diffusion will be discussed in Section 3.1.4.

The last term, $\vec{f}$, represents all external forces to the fluid like user input, or gravity.

We will now look at how to solve Equation 3.1. The key to the equation is incompressibility (Equation 3.2) and to utilize a technique called splitting in which we handle each component in the PDE separately in an iterative manner. Assuming we have functions named advect for advection, diffuse for diffusion, and project for pressure projection, a basic pipeline for finding $\mathbf{u}^{t+1}$ from an initial vector field $\mathbf{u}^t$ might look like:

1. $\mathbf{u}_a = \text{advect}(\mathbf{u}^t)$
2. $\mathbf{u}_d = \text{diffuse}(\mathbf{u}_a)$
3. $\mathbf{u}_f = \mathbf{u}_d + \mathbf{f}^t$
4. $\mathbf{u}^{t+1} = \text{project}(\mathbf{u}_f)$

Advection, diffusion, and incorporating external forces can be solved directly by numerical algorithms. To solve for the pressure projection found in step 4, a slight modification to the Navier-Stokes equations must be made which will be shown in Section 3.1.5.

We will now discuss numerical methods for solving each step of the pipeline.

3.1.2 Domain Discretization

How we decide to discretize the domain of computation has wide-ranging implications. An excellent explanation of domain discretization is found in [5], which is paraphrased and extended upon here.

For simplicity purposes, let's first assume we are working in a plane with an regular grid-like discretization. This type of spatial discretization and solution is called an Eulerian framework. A Lagrangian framework, on the other hand, would have us using particle tracing, but the definition of a derivative becomes non-trivial. In an Eulerian description, all information can be stored in the center of a grid cell. This is typically called a **collocated** grid due to all of the information being stored in one location. For a given scalar q in cell i, q_i, the description of the derivative in a single spatial dimension x using a central differencing scheme becomes:

$$(\frac{\partial q}{\partial x})_i = \frac{q_{i+1} - q_{i-1}}{2\Delta x} \tag{3.3}$$

This equation, using a Taylor Series expansion and simplifying, is accurate to the second order. While the level of accuracy seems acceptable, a more serious problem exists: the sample point q_i is not considered in the computation of the difference. Imagine a scenario that tries to find if the first derivative of a given function are zero, i.e. determine if a function is constant. [5] uses the example $q_i = (-1)^i$. As long as q_{i-1} is equal to q_{i+1}, $(\frac{\partial q}{\partial x})_i$ will show a constant difference at q_i no matter what q_i actually is! This comes down to the **null-space** of equation 3.3 containing

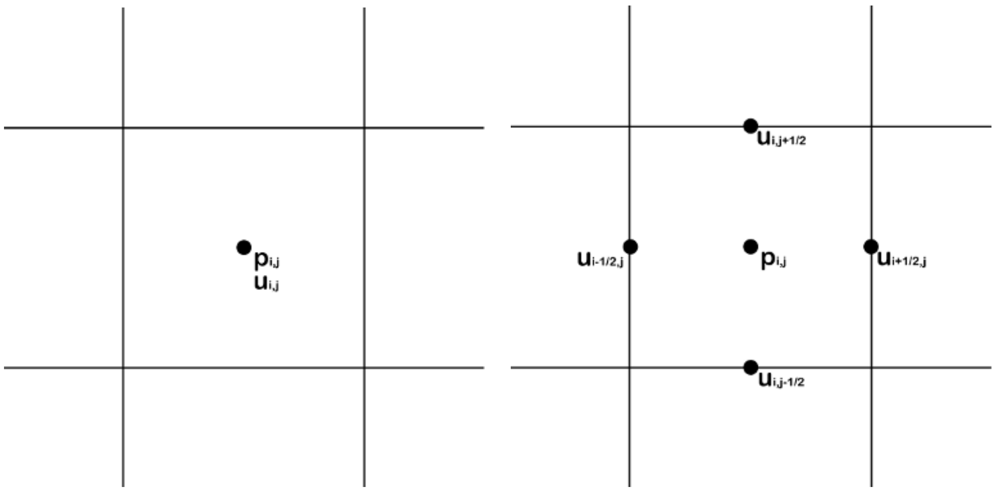


Figure 3.1: The collocated grid (left) suffers from non-trivial null-spaces, which are avoided by using a staggered grid representation (right).

more functions than simply the ones evaluating to zero.

The solution to this is to use a **staggered grid**, also known as a **Marker-and-Cell** (**MAC**) grid. In this situation, quantities are stored on the edges of a cell as well as in the center. See Figure 3.1 for more information. The description of a derivative computed at the center of a cell, when a quantity q is stored on the edges of a cell is therefore:

$$(\frac{\partial q}{\partial x})_i = \frac{q_{i+1/2} - q_{i-1/2}}{\Delta x} \tag{3.4}$$

This formulation is also accurate to the second order, and avoids the differencing issue of equation 3.3. For convenience purposes, we store fluid velocity $\vec{u}$ on cell edges, and pressure values p on cell centers. The extension to three dimensions is relatively straightforward.

3.1.3 Advection

At first glance, the equation representing advection can be solved, for example, through typical finite central differencing methods as described in the previous section. In the case of advection, advecting a vector quantity $\vec{u}$ gives us:

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = 0$$

By using central differencing for the spatial derivative, and forward differencing for time on a collocated grid:

$$\frac{\vec{u}_i^{n+1} - \vec{u}_i^n}{\Delta t} + \vec{u}_i^n \frac{\vec{u}_{i+1}^n - \vec{u}_{i-1}^n}{2\Delta x} = 0$$

And finally rearranging in a common Forward Euler style gives us the integratable equation:

$$\vec{u}_i^{n+1} = \vec{u}_i^n + \Delta t \vec{u}_i^n \frac{\vec{u}_{i+1}^n - \vec{u}_{i-1}^n}{2\Delta x} \tag{3.5}$$

There is, however, a stability problem with using Forward Euler in this manner. The region of absolute stability for Forward Euler is extremely small, so for even reasonable values of Δt and Δx, stability of this solution would still give us problems. Other integrators, such as higher order Runge-Kutta, would merely delay the error propagation because the problem is specifically with central differencing on this type of PDE(hyperbolic). If explicit integrating is required, there are

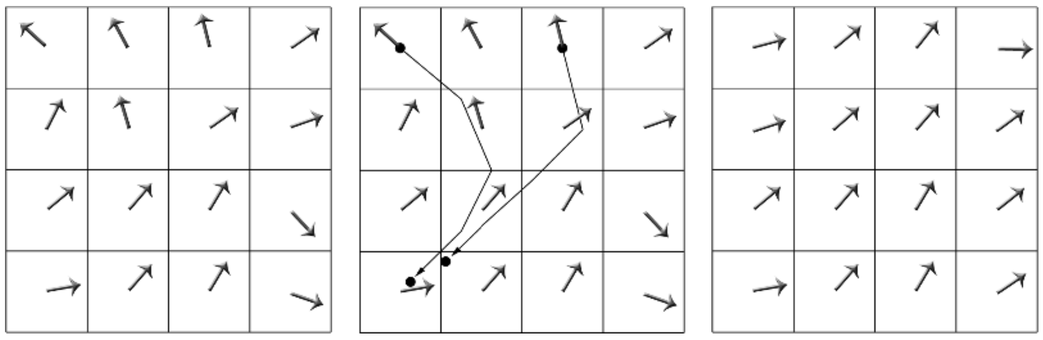


Figure 3.2: Example of Semi-Lagrangian advection. The starting vector field (left) is traced backwards in time from each node (middle), and velocities are transported to the starting nodes resulting in an advected velocity field (right).

"upwind" differencing techniques which choose a biased direction for computing derivatives based on the direction the wave is traveling.

An alternative, and often a bit more intuitive, **unconditionally stable** solution called Semi-Lagrangian advection will now be covered. Originally presented to the graphics community by Stam [43], Semi-Lagrangian advection acquires its name from performing a Lagrangian-like integration to compute a Eulerian calculation. The Lagrangian part of the name comes from tracing a particle through a domain, while the Eulerian calculation comes from the discretized grid representation of space. It starts by asking the question, what will the velocity at a given point be at the next time step? The answer is, whatever velocity is advected to that specific point during the current timestep. Using this intuition, from a given cell i we trace a massless particle backwards through the velocity field a distance of Δt. Typically Forward Euler is good enough, although higher order Runge-Kutta methods will help maintain vortices and defined features of the fluid. From the end point, we interpolate (bi-linear in the grid case) between the nearest neighbors. This interpolated value is **transported** back to the original spot, where the process is repeated for every cell. Refer to Figure 3.2 for more information.

3.1.4 Diffusion

Solving for the diffusion component is a bit more difficult, as it involves solving a large sparse symmetric positive definite matrix of linear systems. As stated before, diffusion describes the viscosity of the fluid, which is the tendency of a fluid to move at the speed of its neighbors. Again, the diffusive term is:

$$\nu\nabla \cdot \nabla\vec{u} = \frac{\partial\vec{u}}{\partial t}$$

The term $\nabla \cdot \nabla$ is commonly referred to as the Laplace operator, Δ, which is also very similar to the Laplacian in both form and function. An excellent source discussing the Laplace operator is found in [53]. We will be using the discrete version to numerically approximate the operator. By using a central differencing approach for a two dimensional staggered grid, with velocity $\vec{u} = (u, v)$, we get the following equation for grid cell i, j and grid spacing $\Delta x = \Delta y$ for diffusing component u with constant viscosity ν:

$$\begin{aligned} (\nu\Delta u)_{i,j} &= \frac{\partial\vec{u}}{\partial t} \\ \nu\left(\frac{u_{i+1/2,j} + u_{i-1/2,j} - 2u_{i,j}}{\Delta x^2} + \frac{u_{i,j+1/2} + u_{i,j-1/2} - 2u_{i,j}}{\Delta x^2}\right) &= \frac{\partial\vec{u}}{\partial t} \\ \frac{\nu}{\Delta x^2}(-4u_{i,j} + u_{i+1/2,j} + u_{i-1/2,j} + u_{i,j+1/2} + u_{i,j-1/2}) &= \frac{\partial\vec{u}}{\partial t} \end{aligned} \tag{3.6}$$

The same is repeated for each component of $\vec{u}$. The series of equations generated from 3.6 can be easily written in Matrix-Vector form, $Ax = f$, with x being the vector of velocity unknowns, and A being the matrix of coefficients of Equation 3.6. Matrix A is typically modified by dividing by -4, leading to a very well-conditioned and intuitively calculated matrix. Solving for all the components of $\vec{u}$ in a single matrix can easily be done, although care should be taken in ordering the components so the matrix stays diagonally dominant. The matrix A turns out to be sparse and symmetric positive definite, which is very easy to solve with typical numerical algorithms. Note that the coefficients of A are directly associated with the numerical approximations of the derivatives. In the next chapter, we will see that due to the non-uniformity and directional ambiguity of surfaces, matrix A is much more difficult to setup.

3.1.5 Projection

The solution for the Navier-Stokes equations relies on the enforcement of incompressibility. Using vector calculus, this measurement of compressibility can be computed by the divergence operator, $\nabla\cdot$. With this operator, equation 3.2 can be interpreted as "the divergence of the velocity field **u** is equal to zero". The opposite of the divergence operator, which measures incompressibility, is known as the curl operator, $\nabla\times$. Incompressible fluids consist of all curl, and no divergence. These operators are a slight abuse of notation, but treating the gradient as simply a vector of partial derivatives will derive more familiar forms. For example, here are the per-component three-dimensional operators with a vector $\vec{u} = (u, v, w)$:

$$\begin{aligned} \nabla \cdot \vec{u} &= \frac{\partial}{\partial x}u + \frac{\partial}{\partial y}v + \frac{\partial}{\partial z}w \\ \nabla \times \vec{u} &= (\frac{\partial}{\partial y}w - \frac{\partial}{\partial z}v, \frac{\partial}{\partial z}u - \frac{\partial}{\partial x}w, \frac{\partial}{\partial x}v - \frac{\partial}{\partial y}u) \end{aligned}$$

Typically we interpret the curl vector as an axis of rotation that the fluid is moving around at a given point. Suppose that we were able to solve every term in the equation except the pressure term, $\frac{1}{\rho}\nabla P$. We would then have a fluid flow which would violate incompressibility, due to the forces generated by the pressure not being present. By measuring the divergence of the current flow field after advection, diffusion, and external force addition, we can solve for the correct amount of pressure to apply to the field in order to enforce incompressibility via equation 3.2. This is done by first measuring the divergence of the current vector field after advection, diffusion, and external force addition:

$$\nabla \cdot (\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} + \frac{1}{\rho}\nabla P) = \nabla \cdot (\nu \nabla \cdot \nabla \vec{u} + \vec{f})$$

Followed by rearranging the terms:

$$\nabla \cdot (\frac{\partial \vec{u}}{\partial t}) + \nabla \cdot (\frac{1}{\rho}\nabla P) = \nabla \cdot (-(\vec{u} \cdot \nabla)\vec{u} + \nu \nabla \cdot \nabla \vec{u} + \vec{f})$$

Notice that the first term, $\nabla \cdot (\frac{\partial \vec{u}}{\partial t})$, making the reasonable assumption that the derivatives are sufficiently smooth, can be rewritten as $\frac{\partial}{\partial t}(\nabla \cdot \vec{u})$. By equation 3.2, the incompressibility constraint, this term is equal to zero. Rewriting and solving for P:

$$\nabla \cdot (\frac{1}{\rho}\nabla P) = \nabla \cdot (-(\vec{u} \cdot \nabla)\vec{u} + \nu\nabla \cdot \nabla\vec{u} + \vec{f}) \tag{3.7}$$

Once we know P, we can subtract $\frac{1}{\rho}\nabla P$ from the divergent field to find the incompressible field. The final vector field at timestep $n+1$ for an advected and diffused vector field at timestep n is therefore:

$$\vec{u}^{n+1} = \vec{u}^{n} - \Delta t\frac{1}{\rho}\nabla P \tag{3.8}$$

An associated theoretical description, called Helmholtz-Hodge decomposition, describes the process in the context of vector fields [29, 28, 49]. Equation 3.7 is often called a Poisson Equation, and is similar to how we solved diffusion in Section 3.1.4. In fact, the matrices composed on the left-hand side are the exact same (in the case of solving diffusion for a single scalar). Note, however, that the right-hand side of the equation is not zero like in the diffuse case, but rather the divergence of the vector field at timestep n after advection, diffusion, and external force addition.

An excellent description of the following material is found in the pressure equations section of [5]. The authors' derivations are summarized and extended upon here. To be able to calculate divergence for the right hand side of the Poisson Equation 3.7, we make use of discrete version of the divergence and curl operators. For example, for a two dimensional staggered grid representation at grid cell i, with $\vec{u} = (u, v)$ and $\Delta x = \Delta y$, the discrete divergence is equal to:

$$\begin{aligned}(\nabla \cdot \vec{u})_i = & \tfrac{\partial}{\partial x} u + \tfrac{\partial}{\partial y} v \\ = & \tfrac{1}{\Delta x}((u_{i+1/2,j} - u_{i-1/2,j}) + (v_{i,j+1/2} - v_{i,j-1/2}))\end{aligned} \tag{3.9}$$

We are enforcing incompressibility at the next timestep, therefore we use Equation 3.8 to write Equation 3.9 in terms of the current timestep n, to get the following derivation for $(u^{n+1}_{i+1/2,j} - u^{n+1}_{i-1/2,j})$:

$$((u_{i+1/2,j} - \frac{\Delta t}{\rho \Delta x}(P_{i+1,j} - P_{i,j})) - (u_{i-1/2,j} - \frac{\Delta t}{\rho \Delta x}(P_{i,j} - P_{i-1,j}))) \tag{3.10}$$

And for $(v^{n+1}_{i,j+1/2} - v^{n+1}_{i,j-1/2})$:

$$((v_{i,j+1/2} - \frac{\Delta t}{\rho \Delta x}(P_{i,j+1} - P_{i,j})) - (v_{i,j-1/2} - \frac{\Delta t}{\rho \Delta x}(P_{i,j} - P_{i,j-1}))) \tag{3.11}$$

Take note that the unknowns in the equation are the pressure values, P. Collect like terms, and move the velocities over to the right-hand side to get the completed two-dimensional equations that need to be solved:

$$\begin{aligned}&\tfrac{\Delta t}{\rho \Delta x^2}(4P_{i,j} - P_{i+1,j} - P_{i-1,j} - P_{i,j+1} - P_{i,j-1}) = \\ &\quad \tfrac{1}{\Delta x}(u_{i+1/2,j} - u_{i-1/2,j} + v_{i,j+1/2} - v_{i,j-1/2})\end{aligned} \tag{3.12}$$

We now collect all the terms in the form of $Ax = b$ where A is a matrix known as the discrete Laplacian, and b is a vector representing the divergence of the vector field at the current timestep. This can be solved by using a variety of numerical solvers, such as the Bi-Conjugate Gradient Solver (BCGS) found in [31]. This is such a not so straight-forward derivation, but the actual implementation details stay fairly concise.

3.1.6 Boundary Conditions

Until now, we have been assuming that the computational domain is all fluid, and that the boundary conditions, like the edge of the domain, are not handled. We will now discuss two boundary conditions of upmost importance to fluids.

The first, Dirichlet boundary conditions, are applied to free surfaces of the fluid such as movement of the fluid into air. By making the observation that fluid is able to freely move into such regions, setting the pressure in an air cell to zero will produce the correct behavior. Notice that we literally do just that in the previous equations. For an air cell i, j, we set the corresponding pressure in the cell, $P_{i,j} = 0$, and modify the Discrete Laplace operator accordingly.

Neumann boundary conditions occur when the fluid encounters an obstacle. Naturally, the fluid should not leak through the solid, but rather slide along it or bounce off. Written in the language of pressure, the pressure values on the boundaries need be solved for so that the velocity of the fluid at the boundary points normal to the object is equal to the normal velocity of the object. This will allow the fluid to slide by without penetrating through the object. We therefore need to solve for a pressure value that will enforce this constraint. Assuming u_s is the velocity of of the solid:

$$P_{i+1,j} = P_{i,j} - \frac{\Delta t}{\rho \Delta x}(u_{i+1/2,j} - u_s) \tag{3.13}$$

In the case of the discrete Laplace operator in Equation 3.12, this amounts to decrementing the diagonal, and adding the second term in Equation 3.13 to the right hand side.

This formulation is not completely physically correct, as the projection of the fluid's velocity onto an object's normal should be **greater than** or equal to zero, rather than strictly zero. However, enforcing an inequality constraint is beyond the scope of this thesis. For further reference, see [4] for an excellent discussion of an alternate solution which incorporates inequality constraints.

3.1.7 Numerical Considerations

A discussion of the numerical complexities of fluid simulations can take an entire thesis to explain. I will limit my discussion here to concepts that I have personally encountered or found useful.

When trying to represent a fluid like water it is important to maintain a high degree of numerical accuracy. Numerical errors typically manifest themselves as the dissolution of features in the fluid, and can be interpreted as diffusion. This will amount to key fluid visual features like vortices or waves failing to form or maintain structure. Diffusion, in most cases, should probably be dropped, and

Semi-Lagrangian advection should use a higher-order tracing scheme, such as the midpoint or trapezoidal method.

Interpolation methods also need to be considered. Since interpolation amounts to an averaging of neighbors, high frequency signals in the velocity field tend to be smoothed out. Higher-order interpolation methods help keep this smoothing to a minimum, but tend to be a bit cumbersome to implement. An effective method was described by [52], in which boundary cells are tracked and initialized with negative values in order to describe the fluid-air interface more accurately. By using a type of fast-marching method, C^1 continuity can be achieved along the fluid interface, and high quality results can be maintained throughout the simulation. This does not address interpolation inside a fluid region. This same type of reasoning can be applied to Neumann boundary conditions, where the pressure in the solid cell can be set to a value that, during interpolation, accurately describes the interface between the fluid and the solid. These negative values applied to the boundary cells are typically call **Ghost** pressures.

Another way of improving interface tracking is by combining the Eulerian mechanics of a grid, with the Lagrangian mechanics of particle tracing. Fluid simulators using Lagrangian mechanics are typically called particle methods. By incorporating a grid-based implicit surface representation of the fluid-air interface, the so called particle level-set methods are created. They represent the most accurate and computationally expensive fluid simulators in modern research.

3.2 Surface Deformation

Since the primary contribution of this thesis is in the area of fluid dynamics and control, not deformation, only the deformation technique implemented by this thesis will be covered.

The deformation algorithm used by our system is a method based on shape matching first presented in [26]. For reference, a summary of their technique is presented here. In typical mass-spring models with explicit integration schemes, like Forward Euler or Runge-Kutta, care must be taken in choosing the timestep and spring stiffness due to the possibility of overshooting the equilibrium.

Meshless shape matching addresses the problem of updating velocity by creating an unconditionally stable integration scheme which does not overshoot the equilibrium, thus always conserving momentum. In this scheme, the mesh is represented by two sets of particles, the mesh in its transformed deformed state, and the mesh in its resting static state. By matching up a particle in the deformed position with the static counterpart, the algorithm can design an integration scheme which moves the particle towards a goal position without erroneously increasing the energy of the system.

The first step in this algorithm is to find the transformation matrix which matches the particles representing the static mesh to the deformed state. Let x_i^0 and x_i be the two sets of particles of mass m_i that represent the static and deformed shapes of the mesh respectively, while x_{cm} represents the center of mass of particle set x. The goal is to find a rotation matrix R and translation vector t

that together best describe the transformation of the static mesh to its deformed state. The translation vector t is simply the vector that translates the center of the static mesh to the center of the deformed one. Computing the rotation matrix R involves first computing a matrix A:

$$
\begin{aligned}
q_i &= x_i^0 - x_{cm}^0 \\
p_i &= x_i - x_{cm} \\
A &= (\textstyle\sum_i m_i p_i q_i^T)(\textstyle\sum_i m_i q_i q_i^T)^{-1} = A_{pq} A_{qq}
\end{aligned}
\tag{3.14}
$$

The rotational part R of this matrix is found via Polar Decomposition [40]. Specifically $R = A_{pq} S^{-1}$ where $S = \sqrt{A_{pq}^T A_{pq}}$ represents the symmetric, non-deforming part of the transformation. Evaluating the square root of a function is done by diagonalizing the matrix through a method like Jacobi rotation. The square root is then applied to the eigenvalues of the system before composing the matrix. Once matrix R has been found, the goal positions and the updates to particle velocities are found by rotating the original particle, $x_i^0 - x_{cm}^0$, by R, and translating by x_{cm}. By linearly interpolating matrix A with the rotation matrix R during the transformation process, stretching and shearing can be introduced into the shape matching. Following the same idea, twisting and bending can be created by computing a larger matrix which encodes quadratic deformation similar to Equation 3.14. Our implementation of this method is shown in Figure 3.3.

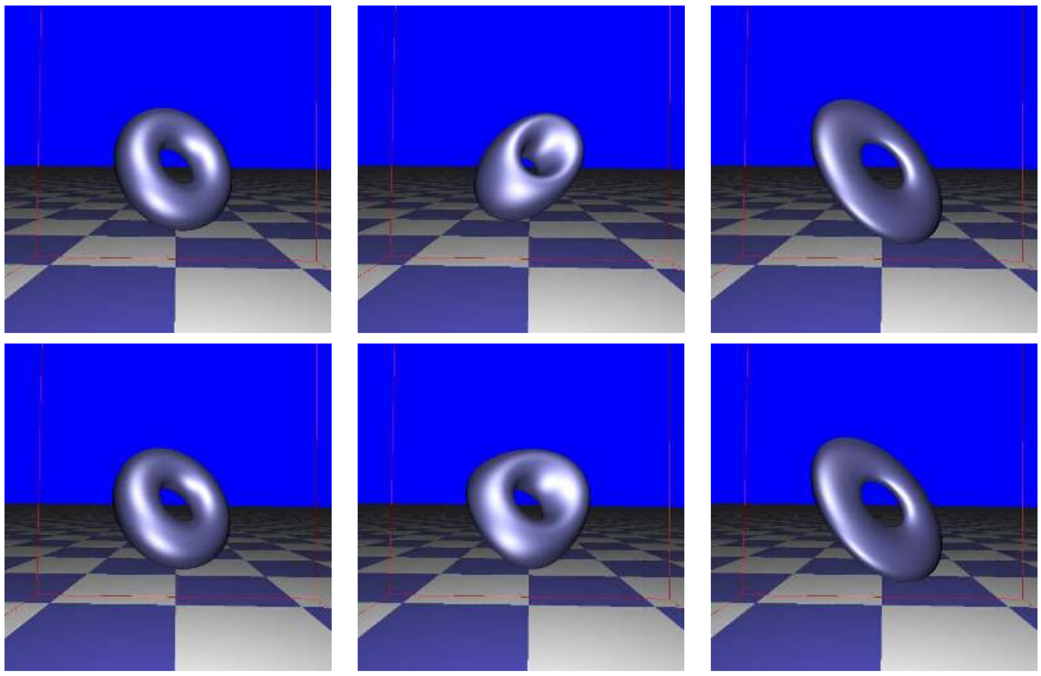


Figure 3.3: Meshless shape matching linear deformation (top) versus quadratic deformation (bottom).

Chapter 4 – Fluid Flow on Interacting Deformable Surfaces

4.1 Fluid Dynamics on Surfaces

While there have been many excellent techniques for solving the Navier-Stokes equations in planes and volumes volumes, few solvers exist for non-planar surfaces. There has been some success in this field by storing a height value and velocity at each surface fluid cell, and solving the Shallow Wave Equations which are a height-field interpretation of the Navier-Stokes equations. Typically, velocity advection and diffusion in those frameworks are ignored, while external force integration and control are enforced through the pressure control of the height field [52]. The largest contribution presented in this thesis is the interactive, controllable framework of designing and controlling fluids on deformable surfaces, while maintaining a pure 2D representation of the full Navier-Stokes equations. The only height field interpretation of the data comes from advecting a scalar quantity along the velocity field, as covered in Section 5.2.2.

Here, again, is the outline of the fluid simulation part of the framework. To compute the velocity of the fluid at the next timestep, $\mathbf{u}^{t+1}$, we perform the following operations:

1. $\mathbf{u}_a = \text{advect}(\mathbf{u}^t)$
2. $\mathbf{u}_d = \text{diffuse}(\mathbf{u}_a)$

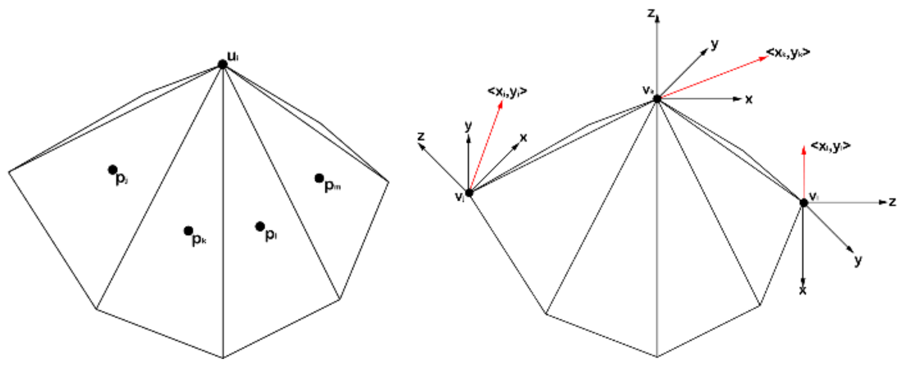


Figure 4.1: Velocities, u, are stored on vertices while pressure values, p, are stored on the barycenter of triangles (left). Local coordinate frames with corresponding tangential velocities are shown in red for vertices j, k, and l (right).

3. $\mathbf{u}_f = \mathbf{u}_d + \mathbf{f}^t$

4. $\mathbf{u}^{t+1} = \text{project}(\mathbf{u}_f)$

An overview of some of the issues concerning fluid dynamics on surfaces is now presented.

It is important to recognize a fundamental difference between a grid representation and a surface based approach. For every vertex, triangle, and edge in the mesh, a local coordinate frame is built where values like velocity, pressure, and curl are stored. This is analogous to the *staggered grid* approach shown in Section 3.1.2, where the pressure values are stored in the center of cells, and the velocity is stored at the edges. This assists, as in the planar and volume case, with differentiation. The up-vector of the local coordinate plane corresponds to a smooth normal at a vertex or triangle. The two perpendicular vectors which complete the local basis, and represent a plane tangent to the surface are chosen arbitrarily as Parallel

Transport alleviates the need for locally smooth parameterizations. Also, velocity in the local coordinate frame representation has only two coordinates. The fluid's velocity in the tangent plane is defined as: $\vec{u} = (u, v)$ where (u, v) is a linear combination with respect to the tangent plane basis vectors, i.e. $\vec{u} = ue_0 + ve_1$ with e_0, e_1 being the two tangent plane basis vectors from the local coordinate frame. Velocity is stored per-vertex, while curl and pressure are stored at the per-triangle barycenter. Refer to Figure 4.1 for more information.

Part of the difficulty for solving the Navier-Stokes equations on surfaces is how exactly do we compute advection, diffusion, and projection. Remember that the discrete differential operators in an evenly-spaced grid are relatively straight forward to define, shown in Equation 3.9. In a triangle representation, this is not the case as triangle area and edge length vary dramatically, which can significantly complicate derivative computation. However, much research has been done on effective numerical approximations of derivatives on triangular or tetrahedral primitives, so we will utilize Mean Value Coordinates as in [11]. We will have to amend all equations requiring spatial derivatives to use the more accurate surface based alternative. Specifically, this affects the divergence calculation for projection in 4.1.5.

A more serious problem also exists in that, on a surface, a question of orientation arises. Specifically, we are easily able to trace particles, interpolate, and average inside a grid or volume because the orientation in Euclidean space is always consistent, i.e. we have a clearly defined up, down, left, and right. Non-Euclidean geometry, such as the surfaces presented in this thesis, do not have a clear orien-

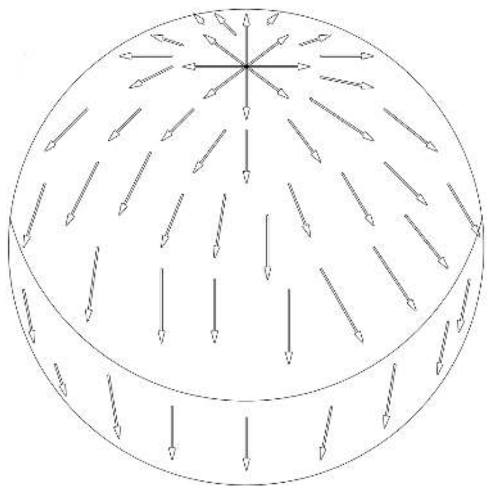


Figure 4.2: An intuitive globally assigned direction of south on a sphere leaves singular points at the poles, where the direction south becomes ambiguous and difficult to deal with.

tation. This is described in the case of a sphere in Figure 4.2. Any attempt to *flatten* or parameterize the surface onto an orientation-consistent plane, for any sufficiently complex surface, will be met with some amount of distortion [44], or singularities where derivatives become impossible to solve. Näively ignoring this problem of orientation causes velocities defined in different coordinate frames to be operated upon! A solution to this problem is presented in Section 4.1.1.

The deformation of the surface effects the simulation numerically as well as visually. Large amounts of computation dependent on the differential properties of the surface, like the Discrete Laplace operator and local coordinate frames, need to be recomputed if the surface deforms. Physics and deformation also have an impact on the fluid flow. This behavior is described in Sections 4.2, 4.2.1, and 4.2.2.

The interactive framework presented in this thesis allows the user to control and design fluid flow and deformation. Control and design mechanisms for fluid flow and deformation are presented in Section 5.1. Effectively visualizing the fluid flow is also a vital problem. Procedures which allow the display of vector field quantities as well as fluid density advection are shown in Sections 5.2.1 and 5.2.2. Since we are essentially doing a 2D simulation on a 3D surface, we need a way of "faking" a volumetric fluid behavior, like rain-drops flowing down the surface. Proof-of-concept of this effect is provided as part of the high-quality visualization discussed in Section 5.2.3.

4.1.1 Parallel Transport

Parallel Transport gives a solution to the orientation problem *locally* around a given point on the surface. The region for transporting between non-oriented coordinate frames is going to be restricted to sufficiently small regions around a given point, anything larger is not necessary for our purposes. For example, given a vertex v_i, we can consider coordinate frames centered on adjacent vertices v_j and triangles T_k. For the barycenter of a triangle, T_i, we can consider adjacent vertices v_j and triangles T_k. Between two non-oriented local coordinate frames, we are going to construct a mapping based on **geodesics** that will allow us to transport a vector from one to another, allowing us to do averaging, differencing, and particle tracing on a surface. See Figure 4.3 for a visual representation of this transportation.

A more technical explanation of connecting geodesics is described by [59]; the author's description is summarized here. A **geodesics** on a curved surface is

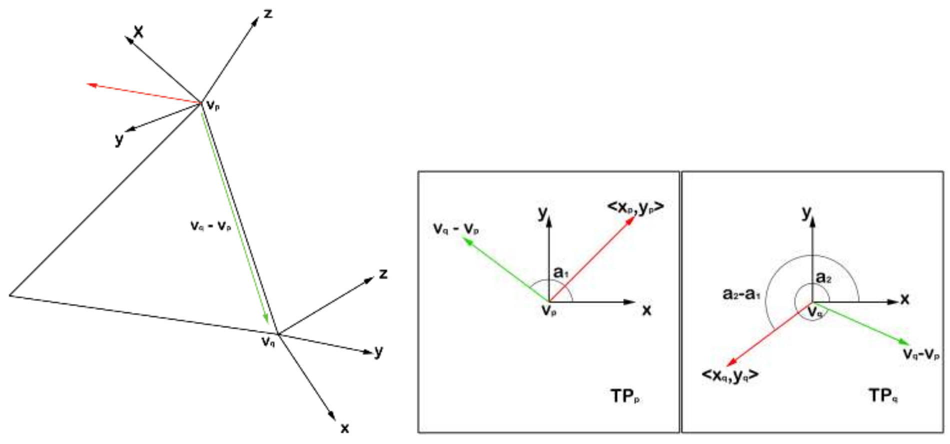


Figure 4.3: The vector Vp-Vq represents the geodesic that the red vector is transported on (left). The vectors in the tangent planes are rotated according to the angle differences between the geodesic and the x-axis (right).

a locally shortest and straightest curve connecting two points. It is basically a generalization of a straight line in the plane. Given a surface S and two points $\mathbf{p}, \mathbf{q} \in S$, there is a **geodesics** γ connecting them, i.e. if $\gamma(0) = \mathbf{p}$ and $\gamma(1) = \mathbf{q}$. Let $\vec{v}_0$ and $\vec{v}_1$ be tangent vectors defined at points $\mathbf{p}$ and $\mathbf{q}$, respectively. If the oriented angle between $\gamma'(0)$ and $\vec{v}_0$ equals that between $\gamma'(1)$ and $\vec{v}_1$, then $\vec{v}_0$ and $\vec{v}_1$ are *parallel* with respect to γ, and $\vec{v}_1$ is said to be the *parallel transport* of $\vec{v}_0$ along γ. γ gives rise to an orthonormal and bijective linear map between $TP_{\mathbf{p}}$ and $TP_{\mathbf{q}}$, the tangent planes at $\mathbf{p}$ and $\mathbf{q}$.

Technical details aside, implementation of Parallel Transport is fairly straightforward. Transporting a vector $\vec{v}_0$ defined in the tangent plane of $\mathbf{p}$, $TP_{\mathbf{p}}$, to a corresponding vector in the tangent plane of $\mathbf{q}$, $TP_{\mathbf{q}}$, is defined as follows:

1. $\vec{u}_{pq} = \mathbf{p} - \mathbf{q}$ where $\mathbf{p}$ and $\mathbf{q}$ are the two *locally close* points in world space

that we are transporting between.

2. $\vec{u}_p = WorldToLocal_{\mathbf{p}}(\vec{u}_p q)$ and $\vec{u}_q = WorldToLocal_{\mathbf{q}}(\vec{u}_p q)$ where $WorldToLocal$ is a transformation between world space and the local coordinate frame defined at $\mathbf{p}$ and $\mathbf{q}$.

3. $\Delta\theta_{=} \tan^{-1}(\frac{v_q}{u_q}) - \tan^{-1}(\frac{v_p}{u_p})$ where $\vec{u} = (u, v)$

4. $\vec{v}_1 = Rotate(\Delta\theta)\vec{v}_0$

Remember that $\vec{v}_0$ and $\vec{v}_1$ are defined in the tangent space of the surface, so the bijective linear mapping, *Rotate*, is simply a two dimensional rotation. Written explicitly:

$$Rotate(\Delta\theta) = \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{pmatrix} \tag{4.1}$$

This rotation from an element j to an element i will, from here on, be referred to as the Transport operator T_{ij}. As long as the differentiable properties of the mesh do not change, $\Delta\theta$ stays constant so the rotation $Rotate(\Delta\theta)$ can be precomputed for fast transportation between local coordinate frames. The restriction of two points being *locally close* is the same as the restriction outlined in the first paragraph. We are now able to bypass the orientation problem with surfaces, and are now able to define interpolation, diffusion, advection, and projection.

4.1.2 Surface Interpolation Using Parallel Transport

We can now use the Transport operator to perform interpolation operations. Interpolation is a necessary part for Semi-Lagrangian advection much like in the planar case. The end point of the massless particle trace will end up somewhere between the vertices we store data at. Interpolation is also necessary for projection. Helmholtz-Hodge decomposition, which is the theoretical foundation for divergence-free projection, is defined for the center of triangles so an interpolation step that transfers from vertex velocities to triangle velocities and back is required. For scalar interpolation, we use a linear basis function:

$$f(x) = \sum \Phi_i(x) f_i \tag{4.2}$$

where $\Phi_i(x)$ is a piecewise linear basis function valued 1 at vertex i, v_i, and 0 everywhere else for a given triangle [49]. This linear basis function serves the same purpose as the familiar Cartesian basis vectors, i.e. $e_0 = (1,0,0)$, $e_1 = (0,1,0)$, $e_2 = (0,0,1)$. A defined function described in a particular basis can be written as a linear combination of its basis vectors. For a Cartesian basis, this turns out to be: $f = \vec{e}_0 f + \vec{e}_1 f + \vec{e}_2 f$. For a triangular basis composed of vertices v_0, v_1, v_2, by Equation 4.2 this turns out to be: $f = \Phi_0 f_0 + \Phi_1 f_1 + \Phi_2 f_2$.

For interpolating to an interior point inside a triangle, the values of the linear basis function $\Phi(x)$ turn out to be the α, β, and γ values of the familiar barycentric coordinates. For interpolating velocities defined at the vertices, to a location inside a triangle we use the Transport operator defined in Section 4.1.1 to handle surface orientation ambiguities. We transport the velocity at each vertex j, $\vec{u}_j$, to the triangles local coordinate frame and perform a weighted summation to get the interpolated triangle velocity, $\vec{u}_i$:

$$\vec{u}_i = \sum_{j \in Tri} \Phi_{ij} T_{ij}(\vec{u}_j) \tag{4.3}$$

where T_{ij} is the Transport operator between triangle vertex j and triangle i, as described in Item 4 of Section 4.1.1. Φ_{ij} is defined as the barycentric weight associated to vertex v_j inside triangle i calculated by Equation 4.2. The interpolation from triangles to vertices is done similarly, with velocities located at the center of triangles being transported to each vertex, weighted, and summed. We now have an interpolation algorithm we will be able to utilize from now on.

4.1.3 Particle Advection on Surfaces

In this section, we use the Transport operator defined in Section 4.1.1 to present an intuitive, unique definition of a per-vertex surface particle advection routine described in the context of fluid dynamics. The algorithm can be broken up into five parts:

1. One-ring flattening to determine the triangle to trace from
2. Time Integration to move along the trajectory
3. Edge Transportation to move a velocity across a non-planar edge

4. Triangle Interpolation to sample at a point inside a triangle
5. Velocity Transport to move the interpolated velocity back to the starting point.

Recall that Semi-Lagrangian advection, described in the planar case in Section 3.1.3, traces the velocity field back in time to determine what velocity should be at the current point. Our current domain discretization stores velocities at the vertices. Since this is a per-vertex algorithm, the number of elements required to advect a velocity field defined on a surface is much less than triangle based approaches in other papers [37].

A näive approach would project the velocity into three-dimensions to perform tracing. Not only is this inconsistent with our surface-based approach, but tracing outside the coordinate frame of a triangle will eventually lead to an incorrect solution due to the trajectory of the three-dimensional trace deviating from the surface. Interpolation requirements would also require excessive switching of coordinate frames with every sample of the surface's velocity field and interpolation. A much more preferred and intuitive solution would be to perform the tracing directly within the tangent space of the mesh, taking advantage of our previously defined Transport operator to handle surface ambiguities and interpolation.

Since the velocity $\vec{u}_i$ is stored in the tangent space at a vertex i, step 1 requires that we first determine which triangle to start tracing from. We use the concept of Geodesic Polar Maps to flatten the one-ring neighborhood onto a planar disk, applying the resulting affine mapping to the velocity stored at the center of the

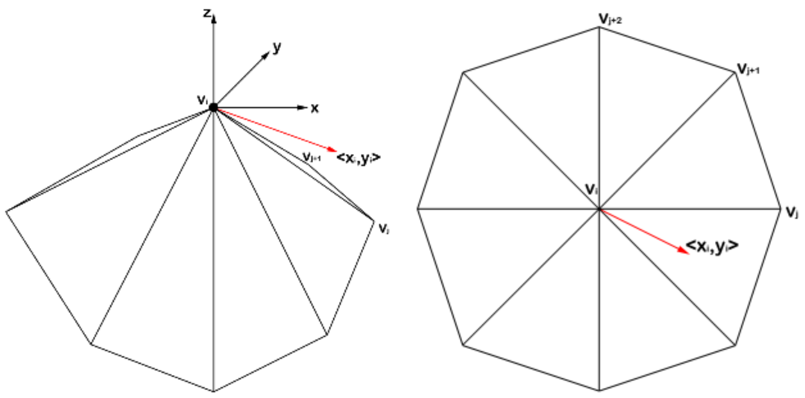


Figure 4.4: The one-ring neighborhood of a vertex is parameterized onto a unit disc, where we evaluate which triangle contains the velocity.

neighborhood. Angles between edges are normalized to add up to 2π, and we compute an edge whose projection onto the tangent plane of the vertex is minimal. The normalized angles are rotations which are applied to the edge in order to find the other edges' representations in this space. Using simple cross products with the edges emanating from the center vertex in this mapping identifies the starting triangle. This is described visually in Figure 4.4. Using the Transport operator, the velocity is mapped to the triangle, where it is interpreted as a ray, $r = \mathbf{o} + \Delta t \vec{v}$, that will be traced on the surface.

To integrate Step 2 through time, we intersect each edge of the triangle with the ray, with the successful intersection resulting in a parameterized distance from the origin d. If $d \leq \Delta t$, integration is stopped and we move onto Step 4. If not, the ray's origin is reparameterized to the intersection point on the edge, while $\Delta t = \Delta t - d$, and the algorithm moves onto Step 3 indicating that the need to trace farther on the surface. Higher order integrators, such as the midpoint method, can be applied here to increase numerical accuracy. Benefits and drawbacks of this are discussion in Section 4.1.6.

Edge Transportation in Step 3 is fairly straightforward due to the Transport operator. The velocity is being transported from triangle element i to triangle element j, which is within the definition of being $l$ocal as discussed in Section 4.1.1. The Transport operator, T_{ij}, is applied to the rays direction $\vec{v}$ and Step 2 continues until the termination condition $d \leq \delta t$ is hit.

Step 4 is again straightforward due to the Transport operator. The ray is traced the remainder of Δt, which is guaranteed to terminate within the current triangle. The velocities are interpolated via Section 4.1.2 to acquire the final velocities.

Step 5 is a little more involved. Since the final coordinate frame the trace ended at is different from one at the vertex origin , a geodesic will need to be defined that transports the velocity back. Luckily, the ray we have traced is just that, a geodesic. Therefore, we can compute $\Delta\theta$ with respect to the velocity vector's representation at the origin at termination, and apply the rotation to our interpolated velocity to arrive at the final result.

In comparison with other surface advection algorithms [37], the algorithm presented here provides: a smaller and more consistent interpolation framework, increased consistency with differential geometry concepts (Parallel Transport and Geodesic Polar Maps), and smaller velocity interpolation calculations (per-vertex versus per-triangle).

4.1.4 Diffusion on Surfaces

Using the Transport operator, solving for diffusion becomes straightforward. We need to again solve the following equation:

$$\nu \nabla \cdot \nabla \vec{u} = \frac{\partial \vec{u}}{\partial t} \tag{4.4}$$

In Section 3.1.4, we solved Equation 4.4 by using a discrete version Laplace operator. We will be doing the same on a surface, except a surface version of the Laplace operator, called Laplace-Beltrami, will be used instead. Finite element approximation of this operator turns out to be a bit more difficult and is not within the scope of this thesis, but we direct the reader to an excellent description by [11]. With the assistance of Mean Value Coordinates and the Transport operator T_{ij}, only the coefficients of $\vec{u}$ in Equation 3.6 change. A linear system of Equation 4.4 can then be written in summation form:

$$\vec{u}_i = \sum_{j \in J} \frac{\omega_{ij}}{\sum_{j \in J} \omega_{ij}} T_{ij} \vec{u}_{ij} \tag{4.5}$$

with ω_{ij} being the weights described in [11]. The division by the summation of the weights is similar to dividing Equation 3.6 by -4 in order to make the resulting matrix simple to compute. The Matrix-Vector form of Equation 4.5 can be solved efficiently by taking advantage of the diagonal, symmetric structure of the coefficient matrix by using the Bi-Conjugate Gradient Solver of [31].

4.1.5 Projection on Surfaces

The largest difficulty in solving projection on surfaces is defining the differential operators for the gradient, divergence, and curl. Once we have surface-based differential operators, projection proceeds similarly to the grid case in Section 3.1.5, albeit with slightly different linear systems. A theoretical basis, called Helmholtz-Hodge Decomposition states that any vector field can be decomposed into three orthogonal fields: one divergence-free, one curl-free, and another incompressible field termed harmonic. Again, the goal is to solve for a pressure scalar field whose gradient indicates the forces that maintain incompressibility, or the divergence-free characteristics of the fluid. Helmholtz-Hodge decomposition is used to compute a curl-free field that represents the compressibility of the system. The gradient field of the potential solved by the decomposition correlates to the $\frac{1}{\rho}\nabla P$ term of the Navier-Stokes equations. Since Helmholtz-Hodge decomposition is defined when velocities are inside triangles, not on vertices, an interpolation step is required before computation of the curl-free potential. Otherwise, the pressure field is computed in the same way as [49], so I direct the reader there for more information.

4.1.6 Numerical Considerations

There are a few areas in this framework where careful understanding of numerical issues is important. Like in the planar case, numerical error during Semi-Lagrangian advection in Section 4.1.3 manifests itself as diffusion. Specifically, important fluid features like vortices or waves tend to diffuse quickly or fail to form at all. While this is fine for effects like smoke, fluids like water and syrup become especially difficult to simulate. This can be partially fixed by using a higher order integration scheme, like midpoint or trapezoidal method, or by segmenting the timestep into smaller intervals. Also, making sure the field is divergence-free before and after advection helps maintain features. All of this is at the price of performance, so we opted to use a second-order tracing method.

The timestep and sampling size, or tessellation, of the surface effect numerical stability as well as convergence. Typically the ratio of largest triangle area to smallest is a decent metric for measuring conditioning, while triangles with poor angle ratios effect accuracy, specifically during interpolation and diffusion. We found that a timestep of 1/100 seconds is usually sufficient for most high-quality visual simulations.

Interpolation around the edges of fluid can be particularly problematic, again contributing to diffusion. In [52], Wang et al. keep track of the boundary cells and initialize cells outside the fluid region with C^1 continuous negative values in order to obtain a more accurate interpolation around the boundaries. An interesting research area involving boundaries would be to investigate contour tracking on surfaces. Perhaps this would lead to a particle level-set method on surfaces that can accurately track water regions on surfaces, and vastly improve numerical issues related to interpolation and advection.

4.2 Surface Deformation

For surface deformation, we implemented the method based on shape matching first presented by [26]. For more information, see Section 3.2. When a surface undergoes deformation, the Discrete Laplace operator, as well as local coordinate frames need to be recomputed. This turns out to be rather fast, being linear with respect to the number of vertices and triangles.

The interactivity of this approach made it a good choice for the system we developed. Currently, Meshless Deformation applies the deformation globally across the whole model, making it less suitable for objects with complex, varying geometry undergoing deformations of different magnitude. The original authors solved this problem through overlapping clusters subdivided across the model. These clustered are then matched to their original configurations, and the particles contained within are updated appropriately. Further refinements to the technique involved lattices in order to increase the level of control and detail in the deformation [34].

We are currently looking into using the Sphere Bounding Hierarchy tree used by our physics system to spatially partition the surface into overlapping regions. Different levels of the tree could demonstrate different levels of stiffness, increasing or decreasing the amount of clusters matched based on dynamically changing stiffness or user interaction. Accurate deformation is not of primary importance to our main goals, so it would mostly be done for aesthetic purposes.

4.2.1 Incorporating Physics

To provide interaction between multiple deformable surfaces, we have implemented a standard collision detection system with a hybrid collision response system that imparts forces onto the object as a rigid body to produce rigid body motion and imparts forces onto the vertices to produce deformations.

The linear and angular motion of each deformable object is governed by typical rigid body dynamics. As objects move through space, we use a hierarchical bounding sphere algorithm to efficiently identify collisions with other deformable surfaces similarly to [32]. Because our objects deform over time, the bounding spheres are updated if deformation is above a certain tolerance threshold.

Our system runs at a time step determined by the user, which is typically set to 1/60 to 1/30 seconds. We use a Runge-Kutta (RK4) integration scheme to integrate the rigid body dynamics forward in time. The RK4 timestep is further subdivided to provide a reasonable timestep for detecting and responding to collisions accurately without unacceptable penetration. Once a collision is successfully detected, we employ an impulse response method to compute and apply forces directly into the rigid body dynamics [2]. The impulse force is also used to scale the position of the colliding vertex, resulting in a deformation that propagates through the surface. See Section 3.2 for more information on deformation.

4.2.2 Fluid-Deformation Interaction

To understand the impact on the fluid by the motion and deformation of the underlying surface, recall the force equation $\vec{F} = M\vec{a}$. Consider a surface S with the mass M, velocity $v(t)$, and external force $F(t)$. The instant acceleration is $\frac{F(t)}{M}$. Given a small concentration of fluid in S with a mass m, the force by S on the fluid is then:

$$f_i(t) = -m\frac{F(t)}{M} \tag{4.6}$$

This force is used to account for the impact on the fluid by the changes in motion and shape of the underlying surface. We also consider the friction between the surface and fluid, which is $f_r = \alpha v$ where v is the fluid velocity relative to the underlying surface, and α is a frictional coefficient representing roughness. Then the total force that exercised by the surface to the fluid is $f_i + f_r$.

Essentially this states that the fluid in this representation is separate from the surface it is represented in, and is affected by frictional forces. If the floor moves out from under it, a fluid with $\alpha = 1$ stays stationary with respect to the surface point under it, while a fluid with $\alpha = 0$ moves completely independent of the surface. This can violate the divergence-free constraint of the Navier-Stokes equation in the case of linear acceleration, as Δv projected onto a spherical object's mesh essentially constitutes a divergent source or sink.

Collisions also introduce ill-suited vector fields at the point of contact, signifying possible compression of the fluid similar to explosive shockwaves in the tangential velocity representation. Angular acceleration, however, is well suited, resolving in

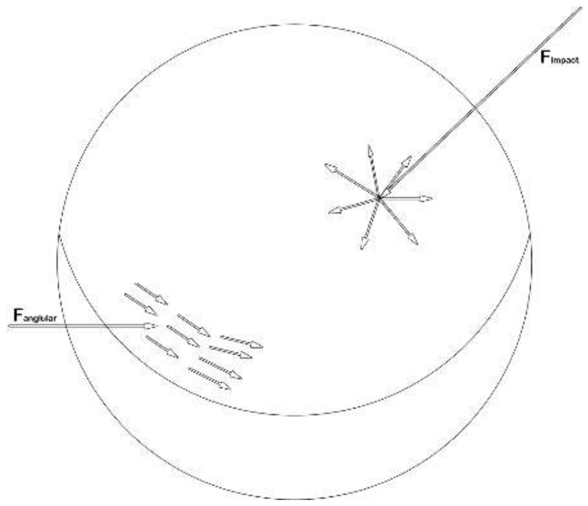


Figure 4.5: Forces due to direct impacts intuitively create divergence forces on the tangential velocity field, which are quickly eliminated during pressure projection. Angular forces caused by rotations create curl forces which are suitable for incompressibility.

the case of the sphere to a pure rotational field. Further exploration into simulating compression would be needed to refine these rules, and to ensure mass is conserved during compressibility. A comparison of these forces is presented in Figure 4.5.

Fluid also is implicit interacted upon by the deformation of the underlying surface. As the surface compresses and decompressed, the surface area the fluid is traveling over expands and contracts, allowing the fluid to flow faster or slower over areas. This can be thought of as the fluid slipping over a surface moving under it. It is unclear, however, if this type of behavior is desirable, as it reduces control over the fluid especially in circumstances undergoing extreme amounts of deformation. A remeshing step which attempts to conserve a uniform sampling distribution could be implemented to avoid this problem.

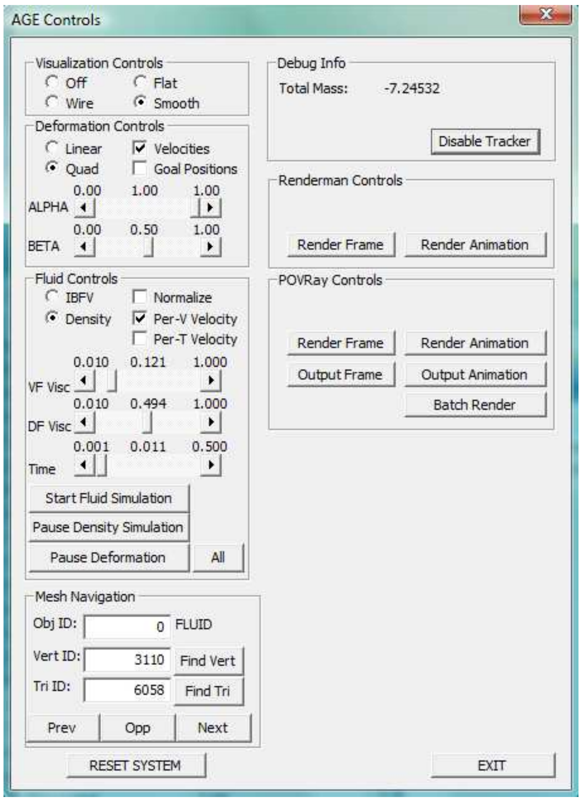
Chapter 5 – Results

5.1 User-Interaction

How the user and the environment interact with the fluid and deformation of the system is outlined in Figure 5.1 as well as the controls available to the user. In this section, control mechanisms for the fluid and deformation part of the simulation are described.

5.1.1 Fluid Control

The user is able to interactively paint velocities and density sources directly onto the surface. Viscosity on both the velocity field and density field are tunable, as well the timestep used in fluid integration. We also provide a decoupling mechanisms in order to simulate each section of the framework individually. This is useful for when the user wants to interactively design a fluid on high-quality meshes. And example of painting a fluid onto a static velocity field is shown in 5.2. Procedurally introduced forces can be used in to generate a complex, highly turbulent flow. The framework also supports density textures, which was used in Figure 5.8 to write the words **deformable** and **flow** on the torus and bunny.


AGE Controls

Visualization Controls
- ○ Off ○ Flat
- ○ Wire ● Smooth

Deformation Controls
- ○ Linear ☑ Velocities
- ● Quad ☐ Goal Positions

ALPHA 0.00 1.00 1.00
BETA 0.00 0.50 1.00

Fluid Controls
- ○ IBFV ☐ Normalize
- ● Density ☑ Per-V Velocity
- ☐ Per-T Velocity

VF Visc 0.010 0.121 1.000
DF Visc 0.010 0.494 1.000
Time 0.001 0.011 0.500

Mesh Navigation
- Obj ID: 0 FLUID
- Vert ID: 3110
- Tri ID: 6058

Debug Info
Total Mass: -7.24532

Figure 5.1: The controls available to the user during the Open-GL visualization.

5.1.2 Deformation Control

Deformation control is done by dragging individual vertices in a direction perpendicular to the viewing direction. The vertex is displaced and scaled, allowing the deformation system to take over and fit a deformation field to the surface, described in detail in Section 3.2. An example of the deformation system implemented in this thesis can be seen in Figure 5.3, while the original work can be found here [26]. Deformation can be decoupled from the fluid simulation so a shape can

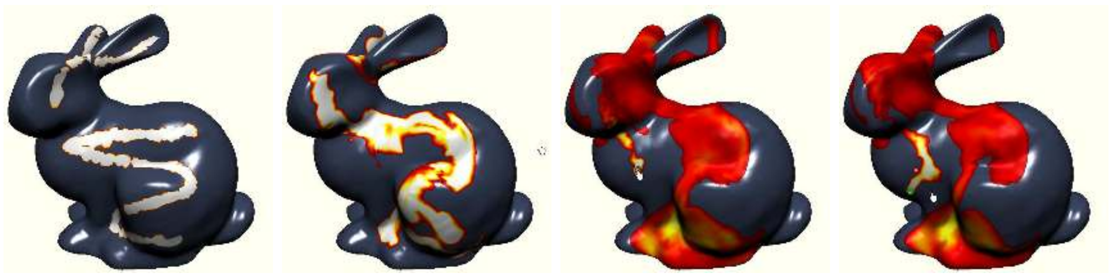


Figure 5.2: Fluid flow on the motionless rigid Bunny surface, where the user is interactively adding density to the low-quality visualization.

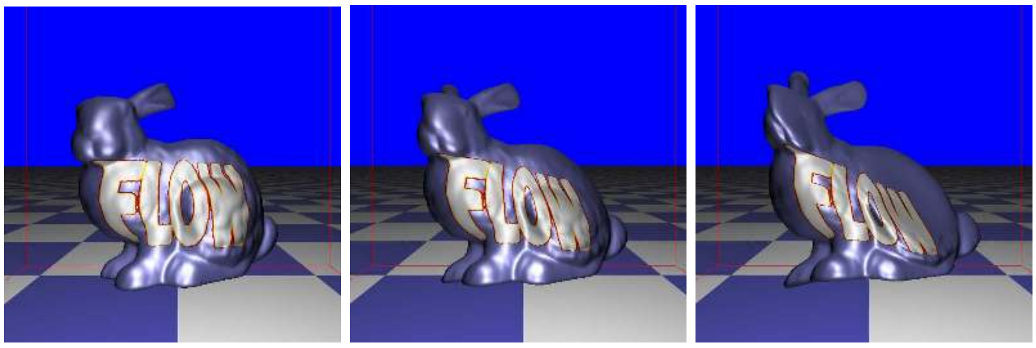


Figure 5.3: A deformation force is applied to a bunny with the words FLOW textured on.

be manipulated to a specific purpose.

5.2 Visualizations

Effective visualization of the fluid is necessary in guiding the design of the fluid. We provide three groups of techniques that the user is able to switch between at run-time: Vector Field visualization, OpenGL, and POV-Ray high-quality post-process. The visualization of vector fields is a heavily researched topic. Being able

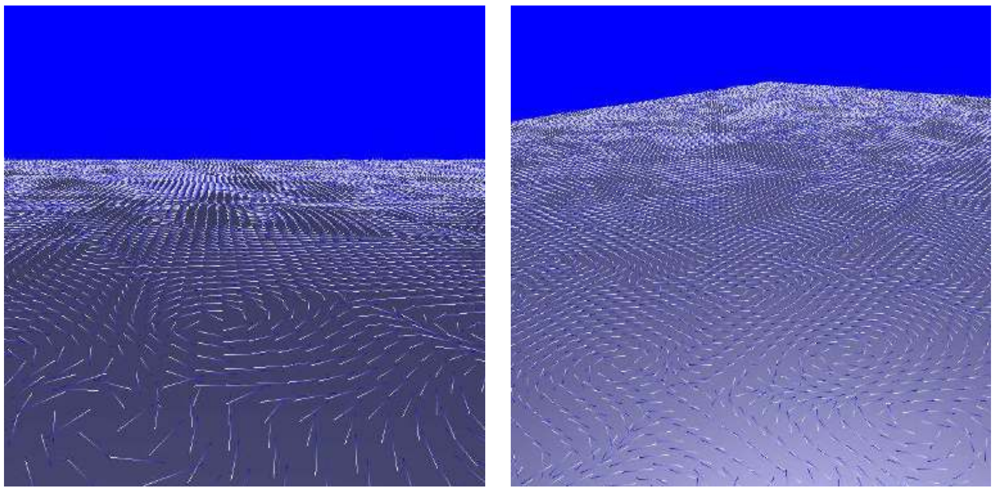


Figure 5.4: Vectors representing velocities at a given vertex are shown in blue.

to identify features of the field is vital for mechanical engineering, visualization, and geometric processing. For fluids, vector field visualization allows us to see fluid flow in regions where fluid may not be present, or regions where the visual characteristics do not visually reflect the characteristics of the flow (saddle-points, or small singularities). We also present an OpenGL framework which serves as a low-quality, interactive design studio. Geometric data is then output to the high quality visualization. In the post-processing visualization stage, photon mapping and ray-tracing are used in the open source ray-tracer POV-Ray to produce the best images.

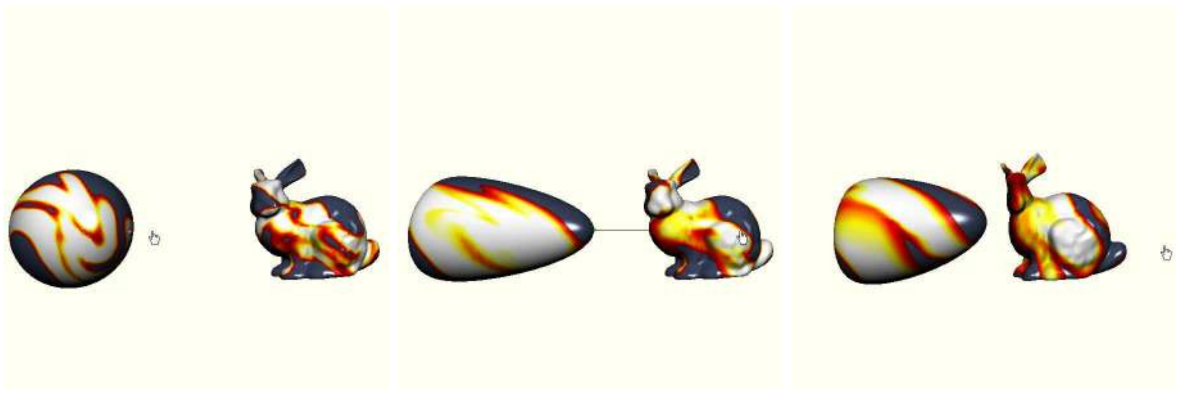


Figure 5.5: Deforming shapes with fluids flowing on their surfaces collide.

5.2.1 Vector Field

For the visualization of vector fields, we use two techniques. We implemented a noise-texture warping method for surfaces presented by [50]. While it produces excellent results, sometimes areas of the field tend to become blurred making it difficult to see the flow in complex regions of the surface. A per-vertex vector field velocity visualization technique is shown in Figure 5.4. This visualization technique scales well with the tessellation of the surface, as the velocities can be scaled or normalized easily.

5.2.2 OpenGL

The OpenGL visualization provides an interactive control studio which allows the user to manipulate and modify the fluid and the surface. Figure 5.5 shows the OpenGL interactive visualization.

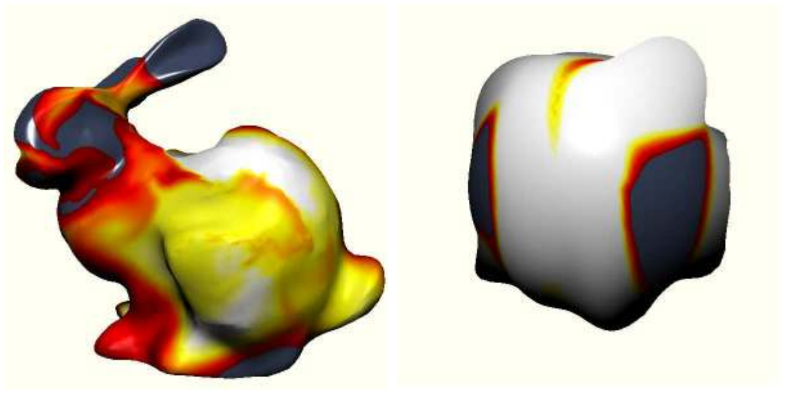


Figure 5.6: Surface geometry being displaced negatively (left) and positively (right) by an advected and diffused height value.

A scalar value representing the density, height, or temperature at a vertex is advected and diffused along the fluid's velocity field. A warm-to-hot (dark red to white) color scale is used which can also reflect the temperature of a fluid at a point. Since this visualization emphasizes interactivity and design, not much concern is spent on quality. The scalar value can be used to displace the vertices through a GLSL vertex shader. A positive displacement provides a visual indicator of fluid accumulation, while a negative displacement gives the illusion of an eroded surface as shown in Figure 5.6.

The performance of the OpenGL visualization is generally dominated by the pressure solution, which involves solving a large spare linear system. Direct precomputed methods like Cholesky Decomposition were considered, but could only be utilized when the differential properties of the surface do not change. There are a variety of numerical methods out there, so more investigation needs to be done on which solver would be best suited for this type of application.

	# of Tri's	Advect	Diffusion	Projection	Total
Sphere	8192	.016	.006	.059	.081
Bunny	72754	.142	.060	.928	1.130
Square	107520	.181	.084	1.107	1.372
Torus	204800	.371	.167	2.828	3.366

Figure 5.7: Timing in seconds per frame averaged over 100 frames on Intel Core 2 Duo 3.0Ghz with NVIDIA 8800GT.

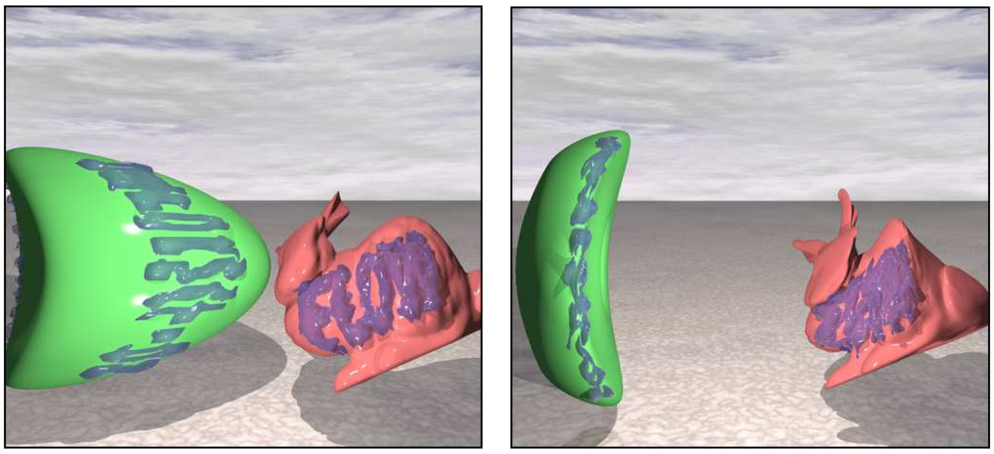


Figure 5.8: The torus and bunny collide!

We used OpenMP to multithread sections of the code suitable for parallelism, like independent updates to the surface's coordinate frames or particle advection. OpenGL performance numbers for meshes of varying complexity with procedurally changing forces every fifty frames are shown in Figure 5.7. The test machine: a Intel Core 2 Duo 3.0Ghz with a NVIDIA 8800GT on Windows Vista.

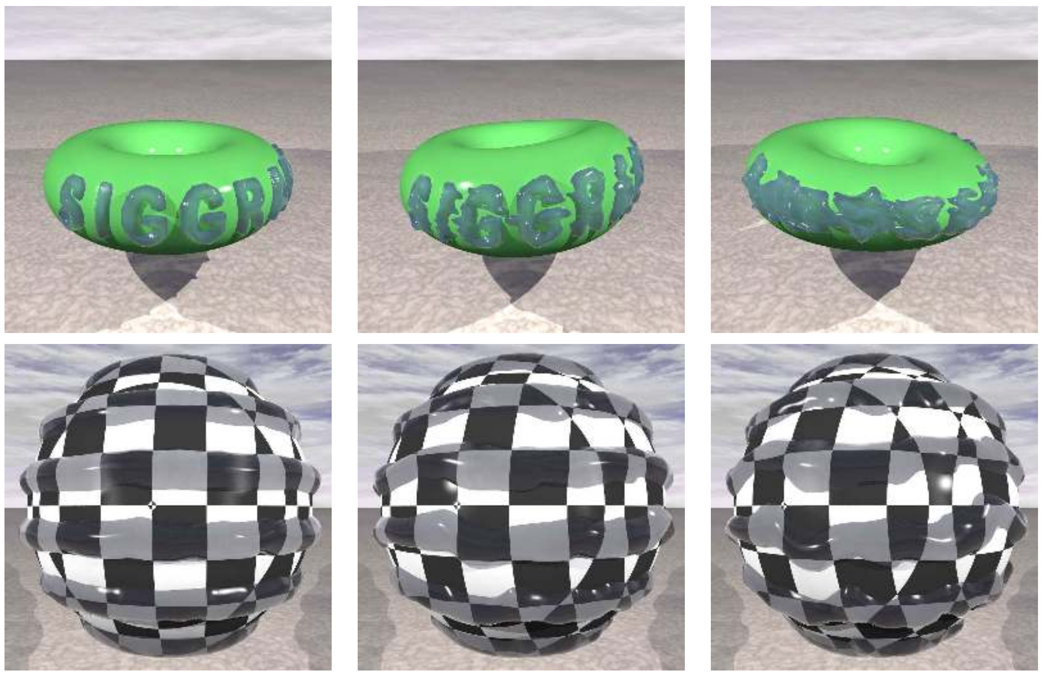


Figure 5.9: Torus and Sphere surfaces are displaced according to an advected height scalar and rendered in a post-processing step through POV-Ray.

5.2.3 POV-Ray Post-Process

Geometric data is piped out from the OpenGL visualization, where it is run through POV-Ray. A separate, displaced mesh based on the scalar value at a vertex which represents a fluid on the surface is also exported for visualization. A Gaussian weighting function is convolved with the fluid's geometric data upon export from the OpenGL visualization in order to smooth out the sharper displacements of the fluid. This gives the visual illusion of a fluid flowing on the surface. Caustics are generated through traditional photon mapping. Parameters specified on a per-object basis reflect the color, transparency, and smoothness of the fluid.

The POV-Ray post-process visualization provides a high-quality render of the scene at the click of the button. Due to expensive ray-tracing effects like photon mapping, each frame can take upwards of three minutes to render. The state of the scene at each frame in an animation can be saved to a file, so that a distributed batch rendering of the frames can be done at a later time.

Chapter 6 – Conclusion

6.1 Future Work

We have demonstrated some of the potential benefits restricting the computational domain of fluid flow to surfaces. While working on the research presented in this thesis, more questions came up than answers. We will now provide a discussion of future work we feel may benefit the current project.

6.1.1 Height-field Description of Surface-Fluid

While a still image of flow on surfaces is moderately convincing, the animation of fluid flow on deforming surfaces does not give the full impression of being physically correct. Vortices and wave propagations which indicate a fluid are present, but the three-dimensional interactions of the fluid with the environment are not fully represented. If the purpose of the fluid is to mimic behavior of three-dimensional flow on surfaces, velocity and pressure in the normal and tangential directions need to be captured. Currently, our system only supports velocities in the surface tangential direction, which limits the types of special effects we are able to generate. A recent paper on solving a generalized version of the Shallow Wave Equations [52] attempts to address the problem of surface flow in the normal direction. The authors use a fully implicit method to enforce incompressibility, which leads to a fair amount of success. However they assume the movement of the fluid is slow, which also limits the types of effects they are able to generate on surfaces.

6.1.2 Fluid-Deformation Coupling

A difficult problem for computation fluid dynamics is the interaction of fluid with deformable surfaces. Restricting the computational domain of the fluid to a surface has obvious advantages in not needing to resolve the complex boundary conditions typical volumetric simulators are required to. The question then becomes how can we describe a physically plausible interaction between the fluid and the deforming surface? What forces does the flow impart upon the surface, and the surface upon the flow? How can we guarantee smooth animation when the domain the fluid is being computed on is dramatically deformed? An interesting approach would be to solve the Navier-Stokes equations alongside the equations describing surface deformation, coupling the two together in a large system of equations. Future research into this possibility may lead to interesting surface simulations of rivers carving into deforming geometry, or rain drops eroding a rock bed.

6.1.3 2D-3D Fluid Transferance

The interaction of a surface based fluid with the environment requires the transference of fluid to and from the surface domain. This problem is especially difficult, as great care would need to be taken in ensuring a smooth transition of fluid to and from the surface. At what point does the fluid start to fall of, or transfer onto the surface? How quickly does this happen? What is a visually smooth way to remesh a water droplet dripping off a surface? These questions all need to be addressed if this type of important physical behavior is to be simulated.

6.1.4 CPU Multicore

Exploiting the recent paradigm shift to multiple cores is essential to the success of breaking through expensive computations to interactive simulation. Soon, computers will be equipped with dozens of cores capability of carrying out embarassingly parallel computations with a near linear speed-up with respect to the number of cores. Parallelism in the framework we have presented here is extremely prevalent. Semi-Lagrangian particle advection is an excellent example, as each vertex can be traced independently of the others, leading to a algorithm which can be easily mapped to multiple cores. The multi-threaded version yielded an almost three times speed-up in this specific algorithm on a hyper-threaded dual-core. Numerical methods used in diffusion and pressure projection can also be mapped to multiple threads as long as the method can be parallelized, like Jacobi iteration. More research would need to be done in order to determine other areas in which these types of optimizations can be effective.

6.1.5 GPU Implementation

The power of the GPU in processing heavily parallel data-intensive tasks like image processing or numerical iterative methods is undeniable. As GPU programming languages turn the GPU into a general purpose arithmetic power-house, large computational speed-ups can be made. A full investigation into porting a surface-based fluid simulator onto the GPU could be an interesting project. While fluid simulators for the GPU have been easily programmed for volumetric or planar simulations, a full-triangle based implementation has not yet been shown. Simply porting part of the pipeline over to the GPU incurs a great cost of data transferring latency. A part way port, in my opinion, is not indicative of the performance a full GPU implementation would have, so a full GPU simulator may prove useful.

6.2 Final Thoughts

The work presented in this thesis could eventually be a vital component of a larger, more interactive, and physically correct fluid design and deformation studio. Rivers could carve canyons over plains, or lava could flow over volcanic surfaces all at real-time speeds in a high-quality visualization. Video games could finally incorporate non-planar fluid simulation without losing a significant amount of performance.

While much research still needs to be completed to fully realize this goal of faking volumetric fluids on surfaces, a proof-of-concept of such a system has been demonstrated by this thesis.

Bibliography

[1] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 617–625, New York, NY, USA, 2005. ACM Press.

[2] David Baraff. Rigid body dynamics, 2001.

[3] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph.*, 26(3):16, 2007.

[4] Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 100, New York, NY, USA, 2007. ACM.

[5] Robert Bridson and Matthias Müller-Fischer. Fluid simulation: Siggraph 2007 course notes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 1–81, New York, NY, USA, 2007. ACM.

[6] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 377–384, New York, NY, USA, 2004. ACM Press.

[7] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *Proceeding of Eurographics*, pages 209–218, 2002.

[8] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schr&#246;der, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics*, 26(1), 2007.

[9] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, 2002.

[10] Ronald P. Fedkiw. Coupling an eulerian fluid calculation to a lagrangian solid calculation with the ghost fluid method. *J. Comput. Phys.*, 175(1):200–224, 2002.

[11] Michael S. Floater. Mean value coordinates. *Comput. Aided Geom. Des.*, 20(1):19–27, 2003.

[12] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 23–30, New York, NY, USA, 2001. ACM.

[13] Nick Foster and Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 181–188, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[14] Simone E. Hieber and Petros Koumoutsakos. A lagrangian particle level set method. *J. Comput. Phys.*, 210(1):342–367, 2005.

[15] Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Subspace gradient domain mesh deformation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1126–1134, New York, NY, USA, 2006. ACM.

[16] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[17] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph.*, 25(3):805–811, 2006.

[18] Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph.*, 26(3):13, 2007.

[19] Doug L. James and Dinesh K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), August 2004.

[20] Martin Kilian, Niloy J. Mitra, and Helmut Pottmann. Geometric modeling in shape space. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 64, New York, NY, USA, 2007. ACM.

[21] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. Fluid animation with dynamic meshes. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 820–825, New York, NY, USA, 2006. ACM Press.

[22] Vivek Kwatra, David Adalsteinsson, Nipun Kwatra, Mark Carlson, and Ming C. Lin. Texturing fluids. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 63, New York, NY, USA, 2006. ACM.

[23] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 362–371, 2002.

[24] Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. Multiple interacting liquids. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 812–819, New York, NY, USA, 2006. ACM.

[25] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 449–456, New York, NY, USA, 2004. ACM.

[26] Matthias Muller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 471–478, New York, NY, USA, 2005. ACM Press.

[27] A. Nealen, M. Mller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. In *Computer Graphics Forum 25*, page 809836. Blackwell Publishing, 2006.

[28] K. Polthier and E. Preuss. Variational approach to vector field decomposition, 2000.

[29] K. Polthier and E. Preuss. Identifying vector fields singularities using a discrete hodge decomposition, 2002.

[30] K. Polthier and M. Schmies. Geodesic flow on polyhedral surfaces. In E. Gröller, H. Löffelmann, and W. Ribarsky, editors, *Data Visualization '99*, pages 179–188. Springer-Verlag Wien, 1999.

[31] William H. Press, William T. Vetterling, Saul A. Teukolsky, and Brian P. Flannery. *Numerical Recipes in C++: the art of scientific computing.* 2002.

[32] S. Quinlan. Efficient distance computation between non-convex objects. *IEEE International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[33] N. Rasmussen, D. Enright, D. Nguyen, S. Marino, N. Sumner, W. Geiger, S. Hoon, and R. Fedkiw. Directable photorealistic liquids. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 193–202, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[34] Alec R. Rivers and Doug L. James. Fastlsm: fast lattice shape matching for robust real-time deformation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 82, New York, NY, USA, 2007. ACM.

[35] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416, 2001.

[36] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914, 2005.

[37] Lin Shi and Yizhou Yu. Inviscid and incompressible fluid simulation on triangle meshes: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):173–181, 2004.

[38] Lin Shi and Yizhou Yu. Controllable smoke animation with guiding objects. *ACM Trans. Graph.*, 24(1):140–164, 2005.

[39] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–236, New York, NY, USA, 2005. ACM.

[40] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. In *Proceedings of Graphics Interface '92*, pages 258–264, 1992.

[41] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 355–362, 2002.

[42] J. Stam. Real-time fluid dynamics for games, 2003.

[43] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[44] Jos Stam. Flows on surfaces of arbitrary topology. *ACM Trans. Graph.*, 22(3):724–731, 2003.

[45] Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 399–405, New York, NY, USA, 2004. ACM.

[46] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popović. Mesh-based inverse kinematics. *ACM Trans. Graph.*, 24(3):488–495, 2005.

[47] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH Comput. Graph.*, 21(4):205–214, 1987.

[48] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, and W. Strasser. Collision detection for deformable objects. In *Eurographics State-of-the-Art Report (EG-STAR)*, pages 119–139. Eurographics Association, Eurographics Association, 2004.

[49] Yiying Tong, Santiago Lombeyda, Anil N. Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graph.*, 22(3):445–452, 2003.

[50] J. van Wijk. Image based flow visualization for curved surfaces.

[51] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.

[52] Huamin Wang, Gavin Miller, and Greg Turk. Solving general shallow wave equations on surfaces. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–238, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[53] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: no free lunch. In *SGP '07: Proceedings of the fifth*

Eurographics symposium on Geometry processing, pages 33–37, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[54] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 247–256, New York, NY, USA, 1994. ACM Press.

[55] Patrick Witting. Computational fluid dynamics in a traditional animation environment. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 129–136, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[56] Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshe. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3), pages 349–358. Blackwell Publishing, 2001.

[57] Gary D. Yngve, James F. O'Brien, and Jessica K. Hodgins. Animating explosions. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 29–36, 2000.

[58] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics*, 24(1):1–27, 2005.

[59] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. *ACM Transactions on Graphics*, 25(4):1294–1326, 2006.