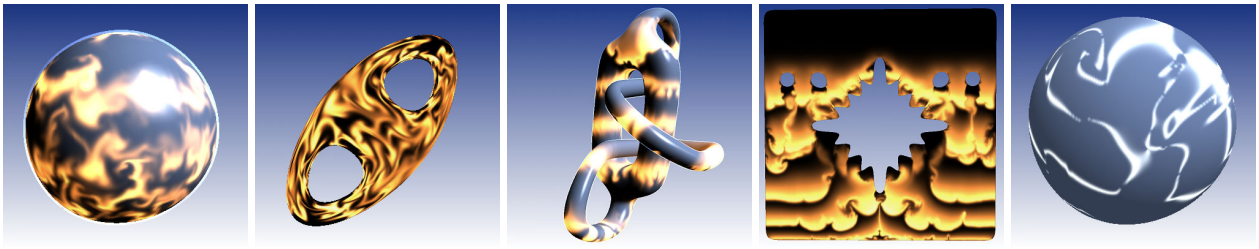
Flows on Surfaces of Arbitrary Topology

Jos Stam*

Alias | wavefront



(a) (b) (c) (d) (e)

Figure 1: Snapshots of flows on various surfaces computed using our novel technique.

Abstract

In this paper we introduce a method to simulate fluid flows on smooth surfaces of arbitrary topology: an effect never seen before. We achieve this by combining a two-dimensional stable fluid solver with an atlas of parametrizations of a Catmull-Clark surface. The contributions of this paper are: (i) an extension of the Stable Fluids solver to arbitrary curvilinear coordinates, (ii) an elegant method to handle cross-patch boundary conditions and (iii) a set of new external forces custom tailored for surface flows. Our techniques can also be generalized to handle other types of processes on surfaces modeled by partial differential equations, such as reaction-diffusion. Some of our simulations allow a user to interactively place densities and apply forces to the surface, then watch their effects in real-time. We have also computed higher resolution animations of surface flows off-line.

CR Categories: 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Computational fluid dynamics, Subdivision Surfaces.

1 Introduction

The simulation of fluid flows in computer graphics has recently experienced a renaissance with the advent of new algorithms and faster hardware. It is now possible to simulate fluid flows in real time for reasonably sized problems. Most of these models, however, assume that the fluids "live in a box", either a square-like domain in two-dimensions or a box-like domain in three-dimensions. Therefore, we thought that it would be interesting and challenging to extend current solvers to handle fluids that live on surfaces of arbitrary topology. A well studied example is that of a fluid flow on a sphere, which can serve as a model for our atmosphere. Indeed, there is a large body of literature in the physical sciences which deals with this problem. However, little or no attention has been devoted to the problem of simulating fluid flows on arbitrary surfaces. We believe that a fluid solver on surfaces of arbitrary topology can result in many practical applications in computer graphics. For example, the flows can be used to decorate surfaces like the ones depicted in Figures 1.(b) and 1.(c) with complex textures. Also the flow field can be used to define a local frame at every point of the surface. This is useful, for example, in specifying a principal direction of anisotropy as shown in Figure 1.(e). More generally, since assigning consistent frames to a surface is not easy our solvers might be of help. Moreover, our surfaces, when restricted to lie in the plane, are actually curved two-dimensional grids which can be applied to the problem of computing flows over arbitrarily curved objects as shown in Figure 1.(d). This is an improvement over current solvers which typically voxelize the objects into a grid. More generally, our approach should be relevant to solve other types of equations on arbitrary surfaces. However, our motivation was mainly intellectual curiosity: what would a fluid flow look like on arbitrary surfaces ? In this paper we provide an answer.

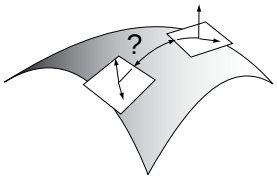


Figure 2: Comparing vectors lying in different tangent spaces is ambiguous without a parametrization.

In order to model fluid flows on surfaces we need both a model for the surface and a model for the fluid simulation. When simulating fluids it is desirable to have a surface representation which

*Alias | wavefront, 210 King Street East, Toronto, ON, Canada M5A 1J7. jstam@aw.sgi.com

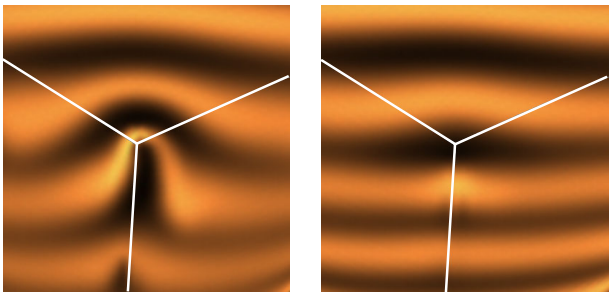


Figure 3: The distortions of the parametrization are apparent in a simulation that ignores the metric (left) and are considerably reduced in our simulations (right).

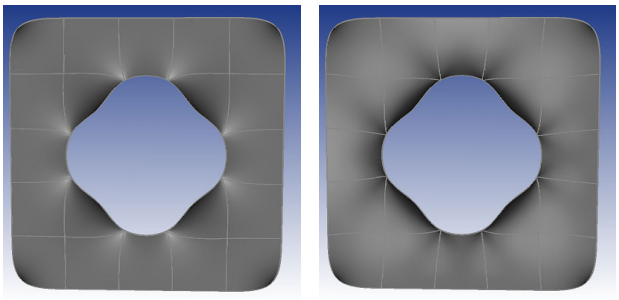


Figure 4: Effect of boundary refinement rules on the parametrization: original Catmull-Clark (left) and modified edge rules by Biermann et al. (right).

admits a global atlas of smooth parametrizations. The reason is that fluid simulators require the comparison of vectors at different points on the surface. In general, these vectors are in different tangent planes as shown in Figure 2, and we need a way to compare them in a common space. A possible solution would be to use some "translation on the surface" mechanism to bring the tangent planes together, but how to do this properly without a parametrization is not obvious. Incidently, this problem does not exist for scalar functions defined on the surface. In this case we can solve equations directly on the surface without a global parametrization, see for example [Turk 1991] or [Desbrun et al. 1999]. On the other hand, parametric surfaces allow tangent vectors to be compared in their parameter domains. Therefore, Catmull-Clark subdivision surfaces are an ideal candidate for our surface model: they are smooth, can handle arbitrary topologies and have a natural quad patch parametrization. That is all we need.

For the fluid solver we decided to implement our Stable Fluids algorithm [Stam 1999] over others [Foster and Metaxas 1996; Foster and Metaxas 1997; Chen et al. 1997; Witting 1999] for the following reasons: it is fast (stable), relatively easy to implement and forms the basis of more sophisticated solvers that simulate swirly smoke [Fedkiw et al. 2001], water [Foster and Fedkiw 2001; Enright et al. 2002] and fire [Nguyen et al. 2002]. The Stable Fluids algorithm uses a semi-Lagrangian solver for the advection [Courant et al. 1952], a projection step to ensure incompressibility [Chorin 1967] and an implicit treatment of the viscosity. The combination of these techniques results in an unconditionally stable solver for the fluid's velocity.

In our first implementation we blindly applied the Stable Fluids solver to each parameter domain while enforcing the cross-patch boundary conditions. However, visible artefacts appeared in our flows such as the one shown in Figure 3 (left). We soon realized that this was due to the fact that we ignored the distortions caused by the parametrization of the Catmull-Clark surface. Fortunately, the equations of fluid flow can be re-written in general curvilinear coordinates which take into account these distortions. These equations are somewhat more complicated than the usual Navier-Stokes Equations, however. To express them concisely (even in two-dimensions) requires some notational devices which we will introduce below. Fortunately, our solver still works in this framework and Figure 3 (right) shows that our novel algorithm drastically reduces these artefacts. Note that artefacts are still apparent but disappear as the grid is refined.

We mention that there is a large body of literature concerned with solving the Navier-Stokes equations in general coordinates, especially in engineering where very often "body-fitted" grids are used to compute flows over complex shapes. Standard references are [Hirsch 1990; Warsi 1998]. Often these solvers use arbitrary triangular meshes combined with a finite element based solver. These solvers are usually more complex and are consequently slower than the one proposed in this paper. More importantly, it is not at all obvious how to implement the semi-Lagrangian algorithm in their framework.

The rest of the paper is organized as follows. In the next section we introduce some notation and briefly mention some properties of Catmull-Clark surfaces. In Section 3 we present our stable fluid solver in general curvilinear coordinates. Section 4 presents the implementation and an elegant way to handle the boundary conditions between patches. Next, Section 5 discusses some results generated with our algorithm. Finally, in Section 6 we conclude and present interesting directions for future research.

2 Catmull-Clark Surfaces

Catmull-Clark surfaces are a generalization of piecewise bi-cubic B-spline surfaces to meshes of arbitrary topology. They were introduced in 1978 by Catmull and Clark [1978] where they are defined as a sequence of refinements of an arbitrary (base) mesh. The refinement rules were designed to generalize the midpoint knot insertion rules for B-Splines. Consequently, where the mesh is regular, Catmull-Clark surfaces agree exactly with B-spline surfaces. Long after their introduction it was proved that these surfaces are in fact tangent plane continuous [Reif 1995; Zorin 1997] and could be evaluated everywhere [Stam 1998]. These two properties make Catmull-Clark surfaces relevant to our problem. Another important property of Catmull-Clark surfaces is that they admit a global atlas of parametrizations. Without loss of generality we assume that the base mesh is made up only of quads. If not, refine once. We therefore have a one to one correspondance between the quads of the base mesh and the patches of the surface. This fact allows us to uniquely label each patch of the surface. Adjacent patches overlap only along the edges of their parameter domains. The transitions between the domains can easily be computed and will be crucial in our algorithm to enforce the correct boundary conditions.

To handle surfaces with open boundaries such as the one shown in Figure 1.(d) we use the rules provided in [Biermann et al. 2000]. These rules use B-spline curve refinement masks on the boundary and a modified edge rule for interior vertices adjacent to an irregular boundary vertex. Fortunately, both tangent continuity and exact evaluation remain valid for these modified rules. As shown in Figure 4 these new rules considerably reduce the distortions due to the parametrization at the boundaries, which is desirable.

We now introduce some notation to make our description more precise. We label the set of quads of the base mesh by $p = 1, \cdots, P$ and assume that the parameter domain of each patch is the unit square $\Omega_p = [0,1] \times [0,1]$. Each quad then defines three functions:

$$y_p^k(x^1, x^2) \quad : \quad \Omega_p \longrightarrow \mathbf{R}, \quad k = 1, 2, 3. \tag{1}$$

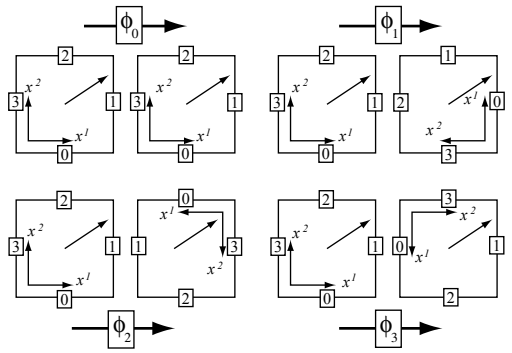


Figure 5: The overlap functions between patches are restricted to the four cases shown. This figure also shows our labeling of the edges.

The patches overlap along their boundaries $\partial\Omega_p$. In order to correctly handle the flows across patches we need to specify the transition functions between adjacent patches as well as their derivatives. First we label each edge of the parameter domain with a number between 0 and 3 as shown in Figure 5. Notice that the labels are ordered counterclockwise. We then observe that the possible transition functions depend only on four functions. Given two adjacent edges with labels e_1 and e_2 it can be shown by inspection that the transition function depends only on the number

$$< e_1, e_2 >= (4 + e_1 - (e_2 + 2)\%4)\%4, \tag{2}$$

where "%" denotes the modulo operator. This simple relation greatly simplifies the implementation of the boundary conditions given below. Indeed, the transitions functions between patches depend only on the following four functions:

$$\phi_0(x^1, x^2) = \left(x^1, x^2\right), \tag{3}$$

$$\phi_1(x^1, x^2) = \left(x^2, 1 - x^1\right), \tag{4}$$

$$\phi_2(x^1, x^2) = \left(1 - x^1, 1 - x^2\right) \quad \text{and} \tag{5}$$

$$\phi_3(x^1, x^2) = \left(1 - x^2, x^1\right). \tag{6}$$

Consequently, the transition function between two edges labeled e_1 and e_2 depends on the function $\phi_{<e_1,e_2>}$. Their derivatives which we need to transform vectors between patches are:

$$\mathbf{M}_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{M}_1 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \tag{7}$$

$$\mathbf{M}_2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \qquad \mathbf{M}_3 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \tag{8}$$

These matrices tell us how we should transform the coordinates of a vector so that it remains unchanged in the new coordinate system. We illustrate this fact in Figure 5 by the central vector in each patch.

To counter the distortions caused by the parametrization we need a measure of deformation. For a surface these are given by the local metric $(g_{i,j})$ which is computed directly from the parametrizations defined by Equation 1 [Aris 1989]:

$$g_{i,j} = \sum_{k=1}^{3} \frac{\partial y^k}{\partial x^i} \frac{\partial y^k}{\partial x^j}, \quad i, j = 1, 2, \tag{9}$$

and where the dependence on p has been dropped. By definition this is a 2×2 symmetrical positive definite matrix, so that $g_{1,2} = g_{2,1}$ and the determinant

$$g = \det\left(g_{i,j}\right) = g_{1,1}g_{2,2} - g_{1,2}^2 > 0. \tag{10}$$

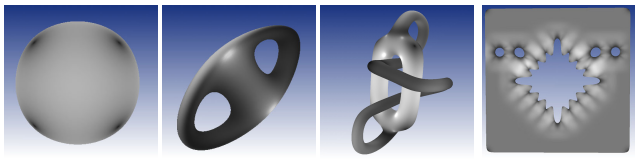


Figure 6: Measure of the distortion $\sqrt{g}$ on the surfaces of Figure 1.

This shows that the inverse of the metric is well defined, which is important since we are interested in defeating the deformations caused by the parametrization. The elements of the inverse are denoted by $g^{i,j}$ and their exact expressions are

$$g^{1,1} = \frac{1}{g} g_{2,2}, \quad g^{2,2} = \frac{1}{g} g_{1,1} \quad \text{and} \quad g^{1,2} = g^{2,1} = -\frac{1}{g} g_{1,2}. \tag{11}$$

Figure 6 shows the values of $\sqrt{g}$ corresponding to the surfaces shown in Figure 1. Notice the deformations near the points on the surface corresponding to the irregular points of the base mesh.

3 Stable Fluids Solver on Surfaces

3.1 Basic Algorithm

In this section we briefly recall the main steps of the Stable Fluids algorithm. For further details we refer the reader to our original paper [Stam 1999]. This algorithm solves the incompressible Navier-Stokes equations, which can be written in the following compact form:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}\{-(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f}\}, \tag{12}$$

where $\mathbf{u} = (u^1, u^2)$ is the fluid's velocity, ν is the viscosity and $\mathbf{f} = (f^1, f^2)$ are external forces. The operator $\mathbf{P}\{\mathbf{v}\}$ projects a vector field $\mathbf{v}$ onto its incompressible (divergence free) component. In addition to providing an update rule for the velocities, our algorithm also includes a method to compute the motion of a density ρ immersed in the fluid. The equation for the evolution of the density is an advection-diffusion equation:

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \nabla)\rho + \kappa \nabla^2 \rho + S, \tag{13}$$

where κ is a diffusion rate and S are sources of density.

For each time step Δt the algorithm solves the Navier-Stokes equation in four steps. Starting from the velocity field of the previous time step $\mathbf{u}_0$, the algorithm resolves each term of Equation 12 sequentially (why this works was first proven in [Temam 1969]):

$$\mathbf{u}_0 \overbrace{\Longrightarrow}^{\text{add force}} \mathbf{u}_1 \overbrace{\Longrightarrow}^{\text{diffuse}} \mathbf{u}_2 \overbrace{\Longrightarrow}^{\text{advect}} \mathbf{u}_3 \overbrace{\Longrightarrow}^{\text{project}} \mathbf{u}_4 \tag{14}$$

The first step is easy to compute, we simply add the force field multiplied by the time step to the velocity: $\mathbf{u}_1 = \mathbf{u}_0 + \Delta t\ \mathbf{f}$. The second step uses a simple implicit solver for the diffusion equation:

$$(\mathbf{I} - \Delta t \nu\ \nabla^2)\mathbf{u}_2 = \mathbf{u}_1. \tag{15}$$

The third step is given by an advection equation:

$$\frac{\partial \mathbf{u}_3}{\partial t} = -(\mathbf{u}_2 \cdot \nabla)\mathbf{u}_3, \tag{16}$$

and is solved with a semi-Lagrangian technique [Courant et al. 1952]:

$$\mathbf{u}_3(\mathbf{x}) = \mathbf{u}_2(\mathbf{x} - \Delta t \mathbf{u}_2(\mathbf{x})).$$

For clarity we assume a simple Euler step for the "backtrace." Higher order schemes are of course also handled by our algorithm. Finally, the last step projects the velocity field onto the incompressibles [Chorin 1967]. This step involves the solution of a Poisson equation

$$\nabla^2 \varphi = \nabla \cdot \mathbf{u}_3, \tag{17}$$

followed by the correction $\mathbf{u}_4 = \mathbf{u}_3 - \nabla\varphi$. Our goal now is to generalize these steps to velocities living in a Catmull-Clark surface.

3.2 Extension to Curvilinear Coordinates

In order for the flows to be truly intrinsic we have to take into account the distortions caused by the parametrizations. The difference from a flat space is that all differential operators such as the gradient "∇" now depend on the metric ($g_{i,j}$) introduced in Section 2. In Appendix A, we provide the expressions of the operators occurring in the Navier-Stokes Equations. These formulas were obtained from the excellent book by Aris [1989], which is a good introduction to the Navier-Stokes Equations in general coordinates.

The first step of the algorithm is unaffected: $u_1^k = u_0^k + \Delta t f^k$. In the second step we have from the expression for the Laplacian that (see Appendix A):

$$\left[\mathbf{I} - \Delta t\nu \, \frac{1}{\sqrt{g}} \frac{\partial}{\partial x^i} \left(\sqrt{g} g^{i,j} \frac{\partial}{\partial x^j} \right) \right] u_2^k = u_1^k, \tag{18}$$

where $k = 1, 2$ and we assume an "Einstein-like" summation between occurrences of identical upper/lower and lower/upper indices (see Appendix A for more details). The advection step is written similarly using the expression of the gradient in Appendix A as:

$$\frac{\partial u_3^k}{\partial t} = - \left(u_2^1 g^{1,j} + u_2^2 g^{2,j} \right) \frac{\partial u_3^k}{\partial x^j}. \tag{19}$$

This equation can be recast in the same form as the usual advection equation (Equation 16) by defining the new velocity field:

$$\bar{u}_2^k = u_2^1 g^{1,k} + u_2^2 g^{2,k}. \tag{20}$$

The beauty of this insight is that we can now apply the same semi-Lagrangian technique of the original Stable Fluids solver with $\mathbf{u}_2$ replaced by $\bar{\mathbf{u}}_2$:

$$\mathbf{u}_3(\mathbf{x}) = \mathbf{u}_2(\mathbf{x} - \Delta t \bar{\mathbf{u}}_2(\mathbf{x})).$$

The same applies to the solution of the advection Equation 13 for the density.

Finally the last step of the Stable Fluids solver involves the solution of the following Poisson-like equation:

$$\frac{\partial}{\partial x^i} \left(\sqrt{g} g^{i,j} \frac{\partial \varphi}{\partial x^j} \right) = \frac{\partial}{\partial x^i} \left(\sqrt{g} \, u_3^i \right). \tag{21}$$

Once φ is obtained from this equation we compute the final velocity by subtracting its gradient from $\mathbf{u}_3$:

$$u_4^k = u_3^k - g^{k,j} \frac{\partial \varphi}{\partial x^j}. \tag{22}$$

This completes the theoretical development of our fluid solver in curvilinear coordinates. In summary, steps two and four of the algorithm still involve inverting linear operators. However, they are now more complicated and depend on the spatially varying metric of the parametrization. It came as a big relief to us that the semi-Lagrangian treatment of the advection term remains applicable to this more general situation. One simply has to replace the advecting field with the one given in Equation 20. We also point out that the equations given in this section remain valid in three-dimensions and could be applied to the computation of fluids on deformed grids for example. We will now show how to put all this theory to work in a practical algorithm to simulate fluid flows on surfaces.

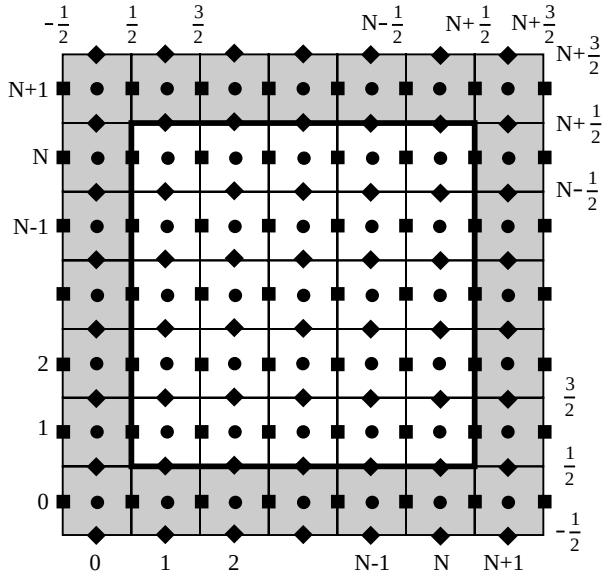


Figure 7: Discretization of one parameter domain. Scalars are defined at the cell centers (dots), the x^1-components of the velocity are defined at the squares, while the x^2-components are defined at the diamonds. The shaded grid cells are not part of the parameter domain but handle the boundary conditions.

4 Implementation

In this section we assume that the reader is familiar both with an implementation of our Stable Fluids solver and of Catmull-Clark surfaces. For the former see the original paper [Stam 1999] and for the latter see for example the most recent notes of the subdivision surface courses given at SIGGRAPH [Zorin et al. 2000].

4.1 Discretization

In our implementation we discretize every patch domain into $N \times N$ grid cells. Instead of the cell centered configuration of the original Stable Fluids solver we use the one shown in Figure 7. This is the so-called MAC configuration which is more stable with respect to the "project" step of the algorithm [Foster and Metaxas 1997; Fedkiw et al. 2001]. All scalars such as the density are defined at the cell centers, while the x^1-components of the velocity are defined on the vertical edges of the grid and the x^2-components are defined on the horizontal edges. The cell centered variables are denoted by

$$\rho_{i,j}, f^1_{i,j}, f^2_{i,j} \quad \text{and} \quad \varphi_{i,j}, \qquad i,j = 0, \cdots, N+1,$$

respectively. The velocity on the other hand is denoted using halfway indices:

$$u^1_{i+\frac{1}{2},j} \quad \text{and} \quad u^2_{j,i+\frac{1}{2}} \qquad i = -1, \cdots, N+1, \; j = 0, \cdots, N+1.$$

This labeling is illustrated in Figure 7. We also added an additional layer of grid cells to handle boundary conditions. The values at these cells will be filled in from the grids of the neighboring patches.

Using the exact evaluation procedure for Catmull-Clark surfaces [Stam 1998], we first compute the metric at every node of the grid using Equation 9. This step only has to be performed once for a given surface and grid resolution. With these values at all the grid points we can discretize the different steps of the Stable Fluids solver as shown in Appendix B. Since the resulting linear systems for the diffusion and the projection steps are symmetric we

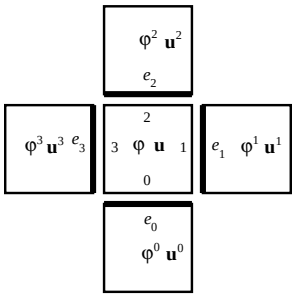


Figure 8: Labelling of the boundary patches and edges used when setting boundary conditions.

solve them using a preconditioned conjugate gradient method as was done in [Fedkiw et al. 2001; Foster and Fedkiw 2001]. The semi-Lagrangian advection step is similar to that of the original Stable Fluids solver except that the modified velocity field $\bar{\mathbf{u}}_2$ replaces $\mathbf{u}_2$. The only tricky part is the handling of boundaries which we will describe in Section 4.3 below.

4.2 Boundary Conditions

The linear solvers we use require that the values at the boundaries are well defined. To facilitate the treatment of the boundaries we have added an extra layer of grid cells. After each iteration of the conjugate gradient solver or any other update of the grids, we populate these layers using the grid versions of the transition functions of Equations 3-6:

$$[0,i,j] = (i,j)\,, \tag{23}$$
$$[1,i,j] = (j,N+1-i)\,, \tag{24}$$
$$[2,i,j] = (N+1-i,N+1-j) \quad \text{and} \tag{25}$$
$$[3,i,j] = (N+1-j,i)\,. \tag{26}$$

Let e_k be the index of the edge adjacent to the edge labelled by "k" and φ^k be the scalar grid values corresponding to the adjacent patch as shown in Figure 8. Given these notations, we set the boundary cells of our grid using the following formulas:

$$\varphi_{0,i} = \varphi^3_{[t_3,N,i]}, \quad \varphi_{N+1,i} = \varphi^1_{[t_1,1,i]},$$
$$\varphi_{i,0} = \varphi^0_{[t_0,i,N]}, \quad \varphi_{i,N+1} = \varphi^2_{[t_2,i,1]},$$

where $i = 1,\cdots,N$ and $t_k =< k, e_k >$ (recall Equation 2).

The strategy to set the boundary values for the velocity are more difficult but follow the same line of reasoning. As for the scalar case let us denote the velocity fields of the adjacent patches by $\mathbf{u}^0$, $\mathbf{u}^1$, $\mathbf{u}^2$ and $\mathbf{u}^3$. Then we have the following update rules for the boundary cells:

$$\left(u^1_{-\frac{1}{2},i}, u^2_{0,i}\right) = \mathbf{M}_3 \left((u^3)^1_{[t_3,N+\frac{1}{2},i]}, (u^3)^2_{[t_3,N,i]}\right),$$
$$\left(u^1_{N+\frac{3}{2},i}, u^2_{N+1,i}\right) = \mathbf{M}_1 \left((u^1)^1_{[t_1,\frac{1}{2},i]}, (u^1)^2_{[t_1,1,i]}\right),$$
$$\left(u^1_{i,0}, u^2_{i,-\frac{1}{2}}\right) = \mathbf{M}_0 \left((u^0)^1_{[t_0,i,N]}, (u^0)^2_{[t_0,i,N+\frac{1}{2}]}\right),$$
$$\left(u^1_{i,N+1}, u^2_{i,N+\frac{3}{2}}\right) = \mathbf{M}_2 \left((u^2)^1_{[t_2,i,1]}, (u^2)^2_{[t_2,i,\frac{1}{2}]}\right).$$

4.3 Advection

We now explain how to treat the semi-Lagrangian algorithm near boundaries. Recall that the semi-Lagrangian procedure in the original Stable Fluids solver involves tracing paths backwards through the grid and interpolating values there. When these paths remain within the patch's grid we can simply apply the original algorithm. Handling the case when these paths wander off to adjacent patches requires more care. Our interpolation routine first checks whether the coordinate lies in the current patch domain. When this is not the case the patch in which the coordinate lies is found, one coordinate at a time. Following is the algorithm that implements these ideas.

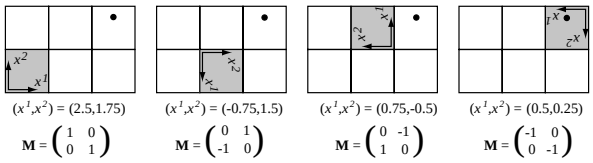


Figure 9: For each point that exits a domain we iteratively find the domain where it lives.

```
Clip ( x^1, x^2, p, M )
  M = I
  while not (x^1, x^2) ∈ Ω_p
    get edge indices e_0, e_1, e_2 and e_3
    if x^1 < 0 then
      p = index of neighboring patch
      (x^1, x^2) = φ_<3,e_3>(x^1 + 1, x^2)
      M = MM_3
    if x^1 > 1 then
      p = index of neighboring patch
      (x^1, x^2) = φ_<1,e_1>(x^1 - 1, x^2)
      M = MM_1
    if x^2 < 0 then
      p = index of neighboring patch
      (x^1, x^2) = φ_<0,e_0>(x^1, x^2 + 1)
      M = MM_0
    if x^2 > 1 then
      p = index of neighboring patch
      (x^1, x^2) = φ_<2,e_2>(x^1, x^2 - 1)
      M = MM_2
  end
```

This step is then followed by a linear interpolation from the grid of patch p at the location (x^1, x^2). When interpolating velocities we also multiply them by the matrix $\mathbf{M}$ to account for the change in parameter domains. Figure 9 illustrates how the coordinate and the matrix $\mathbf{M}$ are computed for a specific example. We note that near an irregular vertex we might get different results if we change the order of the if statements in the above algorithm. To remedy this, we could base the order on the slope of the backtrace direction for example. However, in practice we found that this algorithm worked well.

5 Results

We wrote a program that allows a user to apply forces and deposit densities on the surfaces interactively. All the pictures included in this paper are snapshots from our program: all renderings are done via standard OpenGL calls. The flows are depicted one patch at the time by rendering them into texture maps which are then applied to the rendered mesh.

We also added two surface forces. One is simply a force that simulates gravity. For each surface point (x^1, x^2) we added a force that is proportional to the projection of the downward direction $\mathbf{d}$ onto the tangent plane and the density ρ at that point:

$$\mathbf{f}_g = G\rho\,\mathbf{u}_g,$$

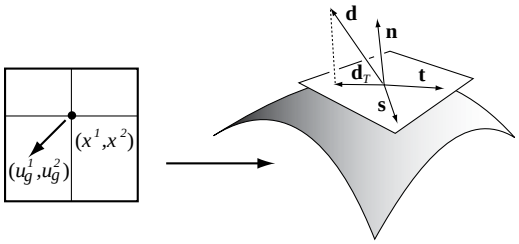


Figure 10: Vectors involved in computing the effects due to gravity in the parameter domain.

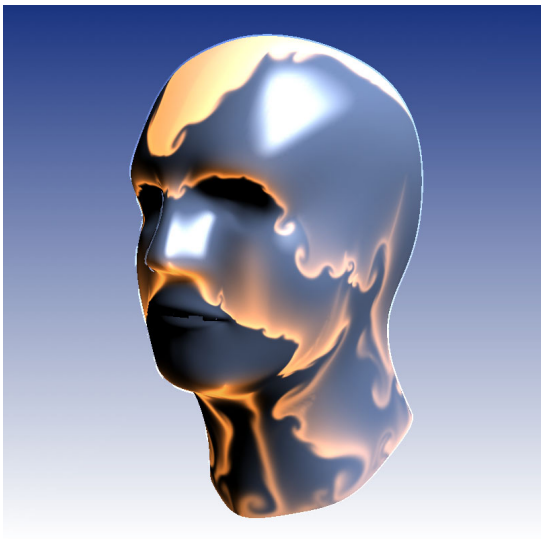


Figure 11: Densities flowing down a head due to gravity.

where G is a constant and $\mathbf{u}_g$ are the coordinates of the projection $\mathbf{d}_T$ of $\mathbf{d}$ in the tangent plane. See Figure 10 for a depiction of this situation. To compute these coordinates we need the frame $(\mathbf{s}, \mathbf{t}, \mathbf{n})$ given by the parametrization in the tangent plane. The vectors $\mathbf{s}$ and $\mathbf{t}$ are the normalized coordinate directions and $\mathbf{n}$ is the normal. In general, this frame is not orthogonal. The projection of $\mathbf{d}$ is given by:

$$\mathbf{d}_T = \mathbf{d} - (\mathbf{n} \cdot \mathbf{d})\mathbf{n}.$$

To obtain the coordinates of $\mathbf{d}_T$ in this frame, however, is not difficult. First compute the dot products

$$a = \mathbf{d}_T \cdot \mathbf{t}, \quad b = \mathbf{d}_T \cdot \mathbf{s} \quad \text{and} \quad c = \mathbf{t} \cdot \mathbf{s},$$

and then set the coordinates $\mathbf{u}_g = (u_g^1, u_g^2)$ equal to

$$u_g^1 = \frac{a - bc}{1 - c^2} \quad \text{and} \quad u_g^2 = b - cu_g^1.$$

Using this force we can simulate densities flowing down a surface, like paint dripping from a human head, as shown in Figure 11. The downward vector is computed from the current viewing direction. In our interactive solver the user can rotate the object while watching the densities flowing downward due to gravity. The head in Figure 11 is comprised of about 450 patches. In this simulation we allocated grids of size 32×32 for each patch. Even at this level of complexity we were still able to produce a frame every five seconds on our Dell Inspiron laptop which has a Pentium III 1GHz processor and a nVidia Geforce2Go graphics card. Figure 12 is another complex example comprised of 223 patches.

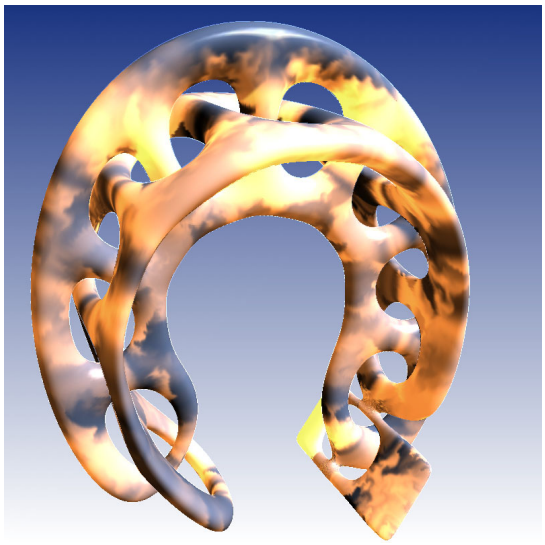


Figure 12: Flow on a surface of complex topology.

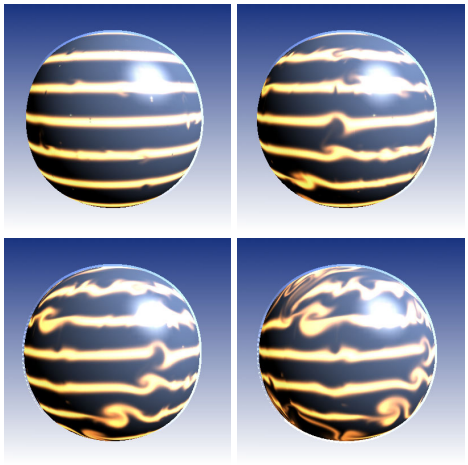


Figure 13: Rotating sphere with a Coriolis force.

A second force we introduce is due to the rotation of an object: the so-called Coriolis effect. We assume that our surface rotates around a vector $\mathbf{r}$ at a rotational speed Ω. Our Coriolis force is then

$$\mathbf{f}_c = \Omega\ (\mathbf{n} \cdot \mathbf{r})\ (u^2, -u^1),$$

where as before $\mathbf{n}$ is the normal to the surface. Notice that this force is proportional to the velocities of the flow. In our simulations we assign a random set of velocities at a sparse set of locations on the patches at the beginning of the simulation. Figure 13 shows four frames from an animation of this effect on a sphere, where we have set the initial distribution of density as a set of horizontal stripes. Notice how the vortices in the flow rotate clockwise in the "northern" hemisphere and counterclockwise in the "southern" hemisphere.

Figure 14 depicts that our model can be used to simulate the flow over arbitrarily curved domains in two-dimensions. This is more accurate than the voxel-occupancy techniques used in previous work [Foster and Metaxas 1997; Fedkiw et al. 2001; Foster and Fedkiw 2001].

Figure 15 shows another application of our model where we use the velocity field to specify the principal direction of anisotropy.

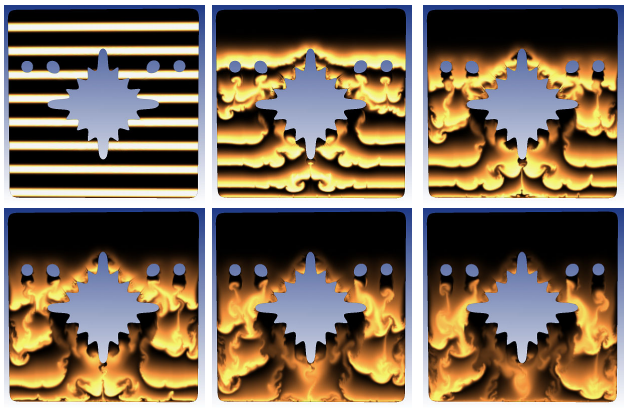


Figure 14: Densities dropping down a domain with curvilinear boundaries.

Figure 15: The velocity field is used as the principal direction of anisotropy in this sequence.

This direction evolves over time and produces interesting animations.

Finally, we can also apply our framework to solve other types of equations on surfaces. Figure 16 shows different reaction diffusion textures created using our solver. In fact, our technique is similar to that of Witkin and Kass [1991]. However, we extend their model to surfaces of arbitrary topology.

6 Conclusions and Future Work

In this paper we have for the first time shown how to simulate fluid flows on surfaces of arbitrary topology. We achieved this by using a combination of fast fluid solvers and Catmull-Clark subdivision surfaces. To minimize the distortions caused by the parametrization we had to use a formulation of the Navier-Stokes equations in general coordinates. Our model can also be applied to the problem of computing flows over arbitrarily curved boundaries in two-dimensions. This suggests an obvious extension of our algorithm to three-dimensions: computing flows over arbitrary objects. This extension would require a formulation of the Catmull-Clark algorithm in three-dimensions. The model described in [MacCracken and Joy 1996] would be a good candidate once an evaluation procedure like [Stam 1998] were developed for it. Extending our model to work on arbitrary meshes is another interesting direction for future research, where the main challenge is an extension of the semi-Lagrangian algorithm.

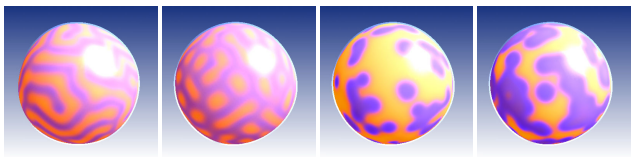


Figure 16: Textures created using a reaction-diffusion process.

A Partial Differential Operators in Curvilinear Coordinates

Working in general coordinates is considerably more difficult than the usual flat euclidean case. The main difficulty consists in mastering the notational devices which have been invented to make the manipulation of formulas more manageable. There's no space here for a short tutorial so we refer the reader to the excellent book by Aris [1989]. The main notational device that we need is the summation convention over repeated indices. The coordinates of points and vectors are always denoted with an index in the upper position as in x^i and u^i. The derivative with respect to a coordinate, denoted by $\frac{\partial}{\partial x^i}$ has by convention an index in a lower position. We can now state the summation convention: any index repeated once in the upper position and once in the lower position in a product of terms is called a dummy index and is held to be summed over the range of its values. In our case the range of the index is always $1, 2$. For example, we have that:

$$g^{i,j}\frac{\partial}{\partial x^j} = g^{i,1}\frac{\partial}{\partial x^1} + g^{i,2}\frac{\partial}{\partial x^2}.$$

We now provide the expressions in general coordinates of the differential operators that appear in the Stable Fluids solver. They are taken from [Aris 1989] (pages 169-70). Gradient:

$$\nabla\varphi = g^{i,j}\frac{\partial\varphi}{\partial x^j}.$$

Divergence:

$$\nabla\cdot\mathbf{u} = \frac{1}{\sqrt{g}}\frac{\partial}{\partial x^i}\left(\sqrt{g}u^i\right).$$

Laplacian:

$$\nabla^2\varphi = \frac{1}{\sqrt{g}}\frac{\partial}{\partial x^i}\left(\sqrt{g}g^{i,j}\frac{\partial\varphi}{\partial x^j}\right).$$

These expressions are also valid in three-dimensions with the dummy indices taking the values $1, 2, 3$. For the vorticity confinement force [Fedkiw et al. 2001] we also require the curl of a velocity field (in two-dimensions):

$$\nabla\times\mathbf{u} = \frac{1}{\sqrt{g}}\left(-g^{i,2}\frac{\partial u^1}{\partial x^i} + g^{i,1}\frac{\partial u^2}{\partial x^i}\right).$$

B Discretization of the Operators

Given the labelling shown in Figure 7 we now provide the discrete versions of the differential operators appearing in each step of our algorithm. We assume that the grid spacing is $h = 1/N$. The Laplacian operator is discretized as follows:

$$(\nabla^2\varphi)_{i,j} = \frac{1}{h^2(\sqrt{g})_{i,j}}\left(D^{1,1}_{i,j} + D^{1,2}_{i,j} + D^{2,1}_{i,j} + D^{2,2}_{i,j}\right),$$

where

$$\begin{aligned}
D^{1,1}_{i,j} &= A_{i+\frac{1}{2},j}D^{1,1}_{i+\frac{1}{2},j} - A_{i-\frac{1}{2},j}D^{1,1}_{i-\frac{1}{2},j},\\
D^{1,2}_{i,j} &= B_{i+\frac{1}{2},j}D^{1,2}_{i+\frac{1}{2},j} - B_{i-\frac{1}{2},j}D^{1,2}_{i-\frac{1}{2},j},\\
D^{2,1}_{i,j} &= B_{i,j+\frac{1}{2}}D^{2,1}_{i,j+\frac{1}{2}} - B_{i,j-\frac{1}{2}}D^{2,1}_{i,j-\frac{1}{2}},\\
D^{2,2}_{i,j} &= C_{i,j+\frac{1}{2}}D^{2,2}_{i,j+\frac{1}{2}} - C_{i,j-\frac{1}{2}}D^{2,2}_{i,j-\frac{1}{2}},
\end{aligned}$$

$A = \sqrt{g}g^{1,1}$, $B = \sqrt{g}g^{1,2}$, $C = \sqrt{g}g^{2,2}$,

$$
\begin{aligned}
D^{1,1}_{i+\frac{1}{2},j} &= \varphi_{i+1,j} - \varphi_{i,j}, \\
D^{1,1}_{i-\frac{1}{2},j} &= \varphi_{i,j} - \varphi_{i-1,j}, \\
D^{1,2}_{i+\frac{1}{2},j} &= \frac{1}{4}\left(\varphi_{i+1,j+1} + \varphi_{i,j+1} - \varphi_{i+1,j-1} - \varphi_{i,j-1}\right), \\
D^{1,2}_{i-\frac{1}{2},j} &= \frac{1}{4}\left(\varphi_{i,j+1} + \varphi_{i-1,j+1} - \varphi_{i,j-1} - \varphi_{i-1,j-1}\right), \\
D^{2,1}_{i,j+\frac{1}{2}} &= \frac{1}{4}\left(\varphi_{i+1,j+1} + \varphi_{i+1,j} - \varphi_{i-1,j+1} - \varphi_{i-1,j}\right), \\
D^{2,1}_{i,j-\frac{1}{2}} &= \frac{1}{4}\left(\varphi_{i+1,j} + \varphi_{i+1,j-1} - \varphi_{i-1,j} - \varphi_{i-1,j-1}\right), \\
D^{2,2}_{i,j+\frac{1}{2}} &= \varphi_{i,j+1} - \varphi_{i,j}, \\
D^{2,2}_{i,j-\frac{1}{2}} &= \varphi_{i,j} - \varphi_{i,j-1}.
\end{aligned}
$$

The gradient is discretized as

$$
\begin{aligned}
(\nabla\varphi)^1_{i+\frac{1}{2},j} &= \frac{1}{h}\left((g^{1,1})_{i+\frac{1}{2},j} D^{1,1}_{i+\frac{1}{2},j} + (g^{1,2})_{i+\frac{1}{2},j} D^{1,2}_{i+\frac{1}{2},j}\right), \\
(\nabla\varphi)^2_{i,j+\frac{1}{2}} &= \frac{1}{h}\left((g^{1,2})_{i,j+\frac{1}{2}} D^{2,1}_{i,j+\frac{1}{2}} + (g^{2,2})_{i,j+\frac{1}{2}} D^{2,2}_{i,j+\frac{1}{2}}\right).
\end{aligned}
$$

And finally the divergence is discretized as

$$
(\nabla \cdot \mathbf{u})_{i,j} = \frac{1}{(\sqrt{g})_{i,j} h}\left(D^1_{i,j} + D^2_{i,j}\right),
$$

where

$$
\begin{aligned}
D^1_{i,j} &= (\sqrt{g})_{i+\frac{1}{2},j}(u^1)_{i+\frac{1}{2},j} - (\sqrt{g})_{i-\frac{1}{2},j}(u^1)_{i-\frac{1}{2},j}, \\
D^2_{i,j} &= (\sqrt{g})_{i,j+\frac{1}{2}}(u^2)_{i,j+\frac{1}{2}} - (\sqrt{g})_{i,j-\frac{1}{2}}(u^2)_{i,j-\frac{1}{2}}.
\end{aligned}
$$

References

ARIS, R. 1989. *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Dover, New York.

BIERMANN, H., LEVIN, A., AND ZORIN, D. 2000. Piecewise Smooth Subdivision with Normal Control. In *SIGGRAPH 2000 Conference Proceedings, Annual Conference Series*, 113–120.

CATMULL, E., AND CLARK, J. 1978. Recursively Generated B-Spline Surfaces On Arbitrary Topological Meshes. *Computer Aided Design 10*, 6, 350–355.

CHEN, J. X., DA VITTORIA LOBO, N., HUGHES, C. E., AND MOSHELL, J. M. 1997. Real-Time Fluid Simulation in a Dynamic Virtual Environment. *IEEE Computer Graphics and Applications* (May-June), 52–61.

CHORIN, A. 1967. A Numerical Method for Solving Incompressible Viscous Flow Problems. *Journal of Computational Physics 2*, 12–26.

COURANT, R., ISAACSON, E., AND REES, M. 1952. On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences. *Communication on Pure and Applied Mathematics 5*, 243–255.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 1999. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow . In *Computer Graphics Proceedings, Annual Conference Series, 1999*, 317–324.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and Rendering of Complex Water Surfaces. In *SIGGRAPH 2002 Conference Proceedings, Annual Conference Series*, 736–744.

FEDKIW, R., STAM, J., AND JENSEN, H. 2001. Visual Simulation of Smoke. In *SIGGRAPH 2001 Conference Proceedings, Annual Conference Series*, 15–22.

FOSTER, N., AND FEDKIW, R. 2001. Practical Animation of Liquids. In *SIGGRAPH 2001 Conference Proceedings, Annual Conference Series*, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic Animation of Liquids. *Graphical Models and Image Processing 58*, 5, 471–483.

FOSTER, N., AND METAXAS, D. 1997. Modeling the Motion of a Hot, Turbulent Gas. In *Computer Graphics Proceedings, Annual Conference Series, 1997*, 181–188.

HIRSCH, C. 1990. *Numerical Computation of Internal and External Flows, Volume 2: Computational Methods for Inviscid and Viscous Flows*. John Wiley and Sons.

MACCRACKEN, R., AND JOY, K. 1996. Free-form deformations with lattices of arbitrary topology. In *Computer Graphics Proceedings, Annual Conference Series, 1996*, 181–188.

NGUYEN, D., FEDKIW, R., AND JENSEN, H. 2002. Physically Based Modeling and Animation of Fire. In *SIGGRAPH 2002 Conference Proceedings, Annual Conference Series*, 736–744.

REIF, U. 1995. A Unified Approach To Subdivision Algorithms Near Extraordinary Vertices. *Computer Aided Geometric Design 12*, 153–174.

STAM, J. 1998. Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values. In *Computer Graphics Proceedings, Annual Conference Series, 1998*, 395–404.

STAM, J. 1999. Stable Fluids. In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, 121–128.

TEMAM, R. 1969. Sur l'approximation de la solution des équations de navier-stokes par la méthodes des pas fractionnaires. *Arch. Rat. Mech. Anal. 33*, 377–385.

TURK, G. 1991. Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. *ACM Computer Graphics (SIGGRAPH '91) 25*, 4 (July), 289–298.

WARSI, Z. U. A. 1998. *Fluid Dynamics. Theoretical and Computational Approaches*. CRC Press.

WITKIN, A., AND KASS, M. 1991. Reaction-Diffusion Textures. *ACM Computer Graphics (SIGGRAPH '91) 25*, 4 (July), 299–308.

WITTING, P. 1999. Computational Fluid Dynamics in a Traditional Animation Environment. In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, 129–136.

ZORIN, D., SCHRÖDER, P., DEROSE, T., KOBBELT, L., LEVIN, A., AND SWELDENS, W. 2000. Subdivision for modeling and animation. SIGGRAPH 2000 Course Notes.

ZORIN, D. N. 1997. *Subdivision and Multiresolution Surface Representations*. PhD thesis, Caltech, Pasadena, California.