Fluid Animation on Arbitrarily-Shaped Structured Grids

Vitor B. R. B. Barroso
Tecgraf / PUC-Rio, Brazil

Waldemar Celes
Tecgraf / PUC-Rio, Brazil

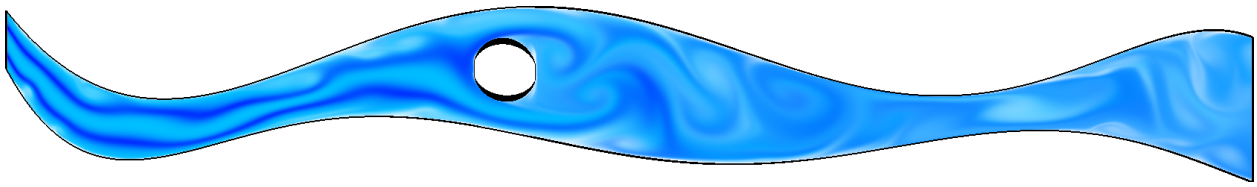

Figure 1: *Example of flow confined in a river-like boundary geometry. The image illustrates blue ink advected by the flow.*

Abstract

In this paper, we present a fast and straightforward technique to simulate two-dimensional fluid flows on planar structured grids with arbitrary shape and parameterization. Although driven by a regular uniform grid, the resulting flows correctly follow and interact with arbitrary boundary walls and internal obstacles. This is accomplished by using Jacobian matrices to relate field derivatives in the world and parameter spaces, which allows us to solve the reduced Navier-Stokes equations directly in the latter. The technique is demonstrated by using a GPU-based Eulerian Stable Fluid solver to generate real-time animations of flows confined in river-like geometry with arbitrary boundaries.

Keywords: Computational Fluid Dynamics, Eulerian Fluids, Fluid Animation, Domain Parameterization

Authors' Contact:
{vbarata,celes}@tecgraf.puc-rio.br

1 Introduction

Fluids and fluid flows are strongly present in our everyday lives. We can easily find examples such as winds, sea waves, smoke steaming from a vent, and the continuous flow of a river. Because of that, the modeling, simulation, and animation of fluid motion has been an active research topic for years in diverse areas, including engineering, movie special effects, and games.

Fluid motion is governed by the Navier-Stokes equations, a set of partial differential equations whose precise numerical solution is remarkably difficult and computationally expensive. Therefore, interactive applications such as games must trade accuracy for efficiency and robustness, while generating interesting and convincing fluid-like flows. For this purpose, a reduced form of the equations is used to drive the simulation of somewhat coarse representations of the fluid.

There are two main approaches for simulating fluids in real-time: Lagrangian techniques, such as Smoothed Particle Hydrodynamics [Stam and Eugene 1995], represent a fluid by a set of moving particles. On the other hand, Eulerian techniques, such as Stable Fluids [Stam 1999], describe a fluid by sampling its properties on each element of a stationary space subdivision, and integrate these properties over time to generate motion. Several domain discretizations may be used, from a regular grid to a dynamic multi-resolution unstructured mesh.

In this paper, our scope is limited to the Eulerian approach. More precisely, we investigate an efficient method to animate fluids on structured grids with curved and arbitrarily-shaped boundaries, such as the winding riverbed in Figure 1. The idea is to simplify the simulation by working on an underlying uniform grid, whose coordinate derivatives are related to the world space ones by means of Jacobian matrices.

Our research builds on the early work on Stable Fluids by Stam [Stam 1999]. His technique was the first to achieve an unconditionally stable real-time Eulerian fluid simulation, and it is still the base of many modern Eulerian algorithms. We extend the method in a way similar to Stam's own generalization technique [Stam 2003], but instead of a Catmull-Clark Surface parameterization, our approach handles only simple planar surfaces without holes in two dimensions, while on the other hand being more easily extensible to arbitrarily-shaped volumes in three dimensions. Another important distinction is that our proposed method requires no explicit topological information, no transition functions between mesh patches, and less complex math: all we need are the positions of the parameterized grid nodes and/or centers in world space, from which we can compute Jacobian matrices that relate field derivatives in both coordinate systems. This well-known mathematical technique allows us to efficiently integrate the reduced Navier-Stokes equations directly in the parameter space, accounting for curves and deformations in a natural way. As a result, we avoid the overhead of working with simplicial meshes with explicit topology or refining the grid near boundaries, which would normally be necessary to accurately model curved geometries, and thus we are able to produce faster river-like flow animations.

The remainder of the paper is organized as follows: in Section 2, we briefly review related works on Eulerian fluid simulation. Section 3 recalls the reduced Navier-Stokes model, the finite difference formulation, and the standard Stable Fluids integrator. In Section 4, we describe the mathematical tools used in our proposed technique, including the Jacobian-based coordinate transformation, and show how to apply them to each step of the simulation. Implementation details are described in Section 5, followed by a discussion of the results in Section 6. Finally, in Section 7, we conclude the work and point out directions for future research.

2 Related work

Fluid simulation techniques for computer graphics have been studied for many years. However, the work by Foster and Metaxas [Foster and Metaxas 1997] was one of the first to achieve real-time frame rates with an entirely physically-based Navier-Stokes integration scheme, without the need for an animator to manually define some of the flow's features. Their method uses a regular grid and integrates the flow equations explicitly, leading to a fast but potentially unstable solution. As a consequence, their approach imposes significant limits on the time step, grid resolution, and fluid viscosity.

The first unconditionally stable real-time Eulerian fluid simulation technique was developed by Stam [Stam 1999], who solved the Navier-Stokes equations on a regular grid in several sequential steps by integrating one term at a time. His method replaces the advection step by an implicit semi-Lagrangian integration. Moreover, it also approaches the diffusion and pressure computations implicitly by solving Poisson equations. This technique has become quite popular and was implemented in GPU by Harris [Harris 2004].

Several works built on the stable solution proposed by Stam. Some tried to model the free surface of fluid flows [Enright et al. 2002], the interactions between different liquids [Hong and Kim 2005; Losasso et al. 2006b], or between a liquid and a rigid or deformable solid [Guendelman et al. 2005; Batty et al. 2007; Robinson-Mosher et al. 2008]. Others attempted to simulate phase changes [Losasso et al. 2006a] or special phenomena like fire [Horvath and Geiger 2009]. Different techniques have also been proposed to handle the inherent numeric dissipation problem of the original model [Fedkiw et al. 2001; Mullen et al. 2009].

Another line of research attempts to improve the modeling of boundary geometries and obstacles. In the early work by Foster and Metaxas, boundaries and obstacles were voxelized into the regular grid by simply marking the non-fluid cells [Foster and Metaxas 1997]. As a result, objects with complex shapes required excessive grid discretization, and those whose orientation did not match the grid introduced visible artifacts in the animation. Several methods have attempted to address this problem, which is still under active research.

Some methods use accurate normals in an attempt to enforce correct boundary conditions [Foster and Fedkiw 2001], or modify the calculations performed in boundary cells to capture object geometries [Johansen and Colella 1998; Roble et al. 2005], but they still suffer from artifacts or face difficulties being evaluated robustly [Batty et al. 2007]. There are also techniques that try to capture fine fluid motion detail only where needed by using octrees [Shi and Yu 2002; Losasso et al. 2004], simplicial meshes [Feldman et al. 2005; Klingner et al. 2006; Chentanez et al. 2007; Mullen et al. 2009] or moving domain representations [Rasmussen et al. 2004; Shah et al. 2004], and techniques that simulate fluids on 3D surfaces of arbitrary topology [Stam 2003; Shi and Yu 2004]. Although some of these are very compelling, their algorithms tend to become more complex to implement, and there is usually a considerable overhead when compared to a regular grid approach.

Similarly to our work, Stam himself has also proposed a technique to generalize the Stable Fluids algorithm to arbitrary parametric surfaces [Stam 2003]. His approach is more general in that it can handle holes and nonplanar surfaces with arbitrary topology. However, it requires an implementation of Catmull-Clark subdivision surfaces [Catmull and Clark 1978] and their exact evaluation at arbitrary parameter values [Stam 1998]. Additionally, it depends on knowledge about the mesh topology and has to deal with overlapping neighboring surface patches by introducing special boundary conditions and update rules, as well as transition functions to convert vector quantities between different patch parameter spaces. Furthermore, his work also needs the precomputation of a local metric from [Aris 1989] that is far more complex than the approach adopted here.

In contrast to the previous work above, our proposed method is less general in that it handles only planar deformed regular grids without holes. However, it provides a contribution in that it uses a straightforward approach based on per-cell Jacobian matrices, which are easy to understand and precalculate. Also, it assumes an underlying single-patch regular grid topology which can be described trivially, requiring only the positions of the grid nodes and/or centers. Moveover, it requires simpler math and no special boundary conditions or transition functions, allowing for a more efficient implementation. And lastly, just like the original Stable Fluids, the technique described here can be readily extended to three-dimensional volumetric domains, which is not so simple with the Catmull-Clark parameterization.

Another interesting approach to handle irregular boundaries and obstacles is proposed in [Batty et al. 2007]. In their work, the usual pressure projection step is rephrased as a kinetic energy minimization, which leads to results free from grid artifacts with relatively coarse regular discretizations. One disadvantage of their approach is that they still require information about the entire domain, including many cells that might be marked as out of bounds, such as in a C-shaped path.

3 Technical background

In this section, we briefly review the reduced Navier-Stokes equations [Chorin and Marsden 1993], the finite difference method, and the Stable Fluids technique [Stam 1999] in 2D. As mentioned before, the model and method can be easily extended to 3D.

3.1 Navier-Stokes equations

A fluid can be described by a velocity vector field ($\vec{u} = [u, v]^T$) and a pressure scalar field (p). We can also consider other properties like temperature (T), density (ρ), kinematic viscosity (ν), and external forces ($\vec{F}$), constant or not, along the fluid.

The evolution of the velocity and pressure fields over time for an incompressible fluid is given by the reduced Navier-Stokes equations:

$$\nabla \cdot \vec{u} = 0 \tag{1}$$

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{u} + \frac{1}{\rho}\vec{F} \tag{2}$$

Note that (2) is actually vectorial and corresponds to two scalar equations in 2D and three in 3D. Also, the differential operators are expanded as follows, where $\vec{u}$ is a vector and r is a scalar:

$$\nabla r = \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \end{bmatrix} \qquad \nabla \cdot \vec{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

$$\nabla^2 r = \frac{\partial^2 r}{\partial x^2} + \frac{\partial^2 r}{\partial y^2} \qquad (\vec{u} \cdot \nabla) r = \frac{\partial r}{\partial x} u + \frac{\partial r}{\partial y} v$$

Equation (1) indicates that the velocity field of an incompressible fluid is divergence-free, which means there can be no point where mass is concentrated or dissipated. Equation (2) can be best understood by examining each of its terms: the first, advection, models how the fluid movement carries any property with it, including the fluid velocity itself. The second, pressure gradient, implies that the fluid is dragged to regions of lower pressure. The third, diffusion, models internal friction, i.e. how the fluid resists to changes in its movement due to viscosity. Finally, the last term corresponds to the acceleration caused by external forces, such as gravity.

3.2 Grid representation and finite differences

In the simplest Eulerian approach, a fluid is described by sampling its properties in a regular structured grid. Each cell is identified by an $[i, j]^T$ index and has the same size $[\delta_x, \delta_y]^T$. Samples are taken on each cell center $\vec{x} = [x, y]^T$. In this discretized space, we need to approximate derivatives for a scalar r by finite differences:

$$\frac{\partial r}{\partial x} = \frac{1}{2\delta_x}\left[r(x + \delta_x) - r(x - \delta_x)\right] + O\left(\delta_x^2\right)$$

$$\frac{\partial^2 r}{\partial x^2} = \frac{1}{\delta_x^2}\left[r(x + \delta_x) - 2r(x) + r(x - \delta_x)\right] + O\left(\delta_x^2\right)$$

Transporting coordinates to the grid and discarding the nonlinear terms:

$$\left.\frac{\partial r}{\partial x}\right|_{i,j} \approx \frac{1}{2\delta_x}\left[r_{i+1,j} - r_{i-1,j}\right]$$

$$\left.\frac{\partial^2 r}{\partial x^2}\right|_{i,j} \approx \frac{1}{\delta_x^2}\left[r_{i+1,j} - 2r_{i,j} + r_{i-1,j}\right]$$

Replacing the above formulas in (2), we find an approximate expression for $\partial\vec{u}/\partial t$. We can then use any numerical integration method to evolve the fluid velocity field and other properties.

It is important to note that a better representation for the fluid domain can be obtained by sampling the fluid velocity in a staggered MAC grid pattern [Harlow and Welch 1965; Foster and Metaxas 1997], in which each vector component is stored in the grid edge or face orthogonal to it. This allows for better energy conservation properties and more robustness to the integration scheme. The technique described in this work was implemented on a centered grid, but it is equally suitable to a MAC grid.

3.3 Stable Fluids

The explicit integration of (2) is potentially unstable [Foster and Metaxas 1997]. This has led to the development of an alternative method, known as Stable Fluids, to evolve the velocity field of a fluid in an unconditionally stable manner. In this technique, whose details can be found in [Stam 1999], the Navier-Stokes equations are solved by sequentially applying four operators: external force (F), advection (A), diffusion (D), and projection (P). Each step uses the velocities computed by the previous one to further evolve the field values:

$$\vec{u}_{i,j}^{n+1} = P \circ D \circ A \circ F\left(\vec{u}_{i,j}^{n}\right) \tag{3}$$

The force step does not present any instability problems. Therefore, an explicit Euler integration scheme can be used.

The stable advection step uses an inverse semi-Lagrangian approach. Consider a particle on the center $\vec{x}$ of a grid cell. If this particle were to be transported by the flow, it would reach a position $(\vec{x} + \vec{u}\Delta t)$ (in a first-order approximation), carrying with it all the fluid properties, including its velocity. If we now think backwards, the particle that will reach position $\vec{x}$ must have come from $(\vec{x} - \vec{u}\Delta t)$. This means we can simply backtrace the particle trajectory, interpolate the property values at the computed location, and assign these values as the new ones for the current cell:

$$r(\vec{x}, t + \Delta t) = r(\vec{x} - \vec{u}(\vec{x}, t)\Delta t, t)$$

$$r_{i,j}^{n+1} = r^n\left(\vec{x}_{i,j} - \vec{u}_{i,j}^{n}\Delta t\right) \tag{4}$$

The stable diffusion step also uses an implicit formulation:

$$\left(I - \nu\Delta t\nabla^2\right)\vec{u}_{i,j}^{n+1} = \vec{u}_{i,j}^{n} \tag{5}$$

In (5), we have a Poisson equation, which can be solved by a variety of methods. Stam proposes the robust multigrid technique [Hackbusch 1985], but in this paper we choose the simpler Jacobi iteration method described in [Harris 2004]. This is a slow-convergence iterative technique, in which we repeat a fixed number of times the following update rule:

$$\vec{u}_{i,j} = \frac{\frac{\vec{u}_{i+1,j}+\vec{u}_{i-1,j}}{\delta_x^2} + \frac{\vec{u}_{i,j+1}+\vec{u}_{i,j-1}}{\delta_y^2} + \frac{\vec{u}_{i,j}}{\nu\Delta t}}{\frac{1}{\nu\Delta t} + 2\left(\frac{1}{\delta_x^2} + \frac{1}{\delta_y^2}\right)} \quad (6)$$

Finally, on the projection step, we first calculate the divergence of the velocity field and use it to find the pressure gradient in (7) [Stam 1999]:

$$\nabla\cdot\vec{u} = \nabla^2 p \quad (7)$$

We have once more a Poisson equation, for which the Jacobi iteration update rule is given by:

$$p'_{i,j} = \frac{\frac{p_{i+1,j}+p_{i-1,j}}{\delta_x^2} + \frac{p_{i,j+1}+p_{i,j-1}}{\delta_y^2} - (\nabla\cdot\vec{u})_{i,j}}{2\left(\frac{1}{\delta_x^2} + \frac{1}{\delta_y^2}\right)} \quad (8)$$

Once the pressure has been found, we project the velocities into a divergence-free field by applying:

$$\vec{u}' = \vec{u} - \nabla p \quad (9)$$

After the velocities have been integrated, we can use them to evolve scalar quantities through the fluid, such as density and temperature. This evolution can be modeled by:

$$\frac{\partial r}{\partial t} = -(\vec{u}\cdot\nabla)\, r + \kappa_d\nabla^2 r - \kappa_e r + S \quad (10)$$

In this equation, κ_d and κ_e are configurable coefficients. Note also that (2) and (10) are strikingly similar. Indeed, the same integration method can be applied here by using sequential operators for source (S), advection (A), diffusion (D), and extinction (E):

$$r_{i,j}^{n+1} = E\circ D\circ A\circ S\left(r_{i,j}^{n}\right) \quad (11)$$

In (11), the extinction term can be integrated directly:

$$r_{i,j}^{n+1} = (1+\kappa_e\Delta t)^{-1}\, r_{i,j}^{n} \quad (12)$$

Finally, we can also compute how particles are transported by the flow. This is done by forward integration of each particle position using the fluid velocities.

As a final remark, it should be noted that, since the calculation of derivatives for a cell requires accessing its direct neighbors, we need an extra layer of cells around the fluid domain to impose some boundary conditions. A good discussion of this topic can be found in [Foster and Metaxas 1997].

4 Parameterized Stable Simulation

The main goal of our work is to be able to use a simple uniform grid to drive a fluid simulation on an arbitrarily-shaped structured grid. The idea is to perform the integration of the reduced Navier-Stokes equations directly in the uniform grid's parameter (s,t) space, instead of in the (x,y) coordinates of the more complex world-space domain discretization. Figure 2 shows the relationship between the two coordinate frames.

We start by presenting the Jacobian matrix concept and how it can be used to convert some of the fluid properties between the world and grid spaces. Then, we examine some additional techniques and mathematical adaptations that must be made to each step of the Stable Fluids solver in order to achieve our goal. Throughout the section, we use the notation f_x to denote the derivative of f in the x direction, and $f_{(x,y)}$ to denote the value of f in the (x,y) coordinate space.

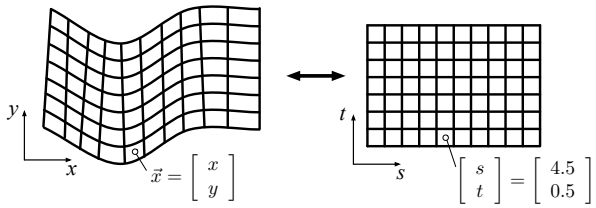

Figure 2: *Relationship between world (x,y) and grid (s,t) coordinates.*

4.1 Jacobian matrix and coordinate systems

Looking back to Equations (1) and (2), we note that the fluid behavior is described only in terms of velocities and other field derivatives, not depending directly on any positional information. Therefore, all we need is to relate these properties in the world and grid spaces. For that purpose, we can express the derivatives f_s and f_t as follows:

$$f_s = f_x x_s + f_y y_s \qquad f_t = f_x x_t + f_y y_t \quad (13)$$

Similarly, we can express f_x and f_y as:

$$f_x = f_s s_x + f_t t_x \qquad f_y = f_s s_y + f_t t_y \quad (14)$$

In matrix form:

$$\begin{pmatrix} f_s \\ f_t \end{pmatrix} = \begin{pmatrix} x_s & y_s \\ x_t & y_t \end{pmatrix} \begin{pmatrix} f_x \\ f_y \end{pmatrix} \Rightarrow \nabla f_{(s,t)} = J\nabla f_{(x,y)} \quad (13b)$$

$$\begin{pmatrix} f_x \\ f_y \end{pmatrix} = \begin{pmatrix} s_x & t_x \\ s_y & t_y \end{pmatrix} \begin{pmatrix} f_s \\ f_t \end{pmatrix} \Rightarrow \nabla f_{(x,y)} = J^{-1}\nabla f_{(s,t)} \quad (14b)$$

Using the definitions above for the Jacobian matrix J and its inverse, it can also be easily shown that:

$$\vec{u}_{(s,t)} = (J^{-1})^T \vec{u}_{(x,y)} \quad (15)$$

$$\vec{u}_{(x,y)} = J^T \vec{u}_{(s,t)} \quad (16)$$

Both the Jacobian matrix J and its inverse can be precomputed by the fluid solver. In this work, we calculate the Jacobian matrix terms on each cell by finite differences with the positions of neighboring cell centers, and then store them just like any other fluid property.

It is important to note that, since the Jacobian is not necessarily constant over each cell, it should also be interpolated when needed. However, throughout the algorithm, most of the times it will be queried exactly at a cell center, in which case we can avoid this extra cost.

Another remark is that, as we will see in Section 4.3, we will also need the second-order terms s_{xx}, s_{yy}, t_{xx} and t_{yy}. Once we have the inverse Jacobian matrix sampled over the grid, these terms can also be precalculated by considering each term of J^{-1} as a scalar field f and using Equation (14) again, along with finite differences.

4.2 Advection step and particle tracking

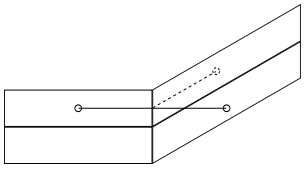

Figure 3: *Effect of particle tracking for a velocity in the x direction. Dashed line: trajectory without tracking. Continuous line: trajectory with velocity recalculation in grid coordinates.*

In the advection step of the solver, we first query the flow velocity at the center of each grid cell. We then use the Jacobian matrix to transform this velocity to grid coordinates, where the advection is to be performed. The backtracking of the Lagrangian particle can be done by an integrator of any desired order. After reaching the final location, the value of the advected field can be interpolated and copied back to the original cell.

It is important to note that, with higher-order integrators or when the time step allows the particle to go through several grid cells at once, it is necessary to track the particle and find out when it moves from a cell to one of its neighbors. This is often the case in fluid animation and requires a very simple verification in the underlying grid coordinates. On each transition, since the Jacobian matrix can be different for each cell, the velocity in grid coordinates should be recalculated from the remaining time step movement in world coordinates. Figure 3 illustrates how this can affect the behavior of the fluid.

4.3 Diffusion step and Jacobi iterations

For the diffusion step, the form of the Jacobi iterations must be derived from scratch. We start with Stam's implicit equation [Stam 1999], with the time step replaced by h to avoid confusion with the t coordinate, and using $\vec{u}'$ to denote the next value of $\vec{u}$ after the current iteration:

$$\frac{\vec{u}' - \vec{u}}{\nu h} = (\vec{u}_{xx} + \vec{u}_{yy}) \tag{17}$$

Using (14) twice to expand the derivatives of the Laplacian:

$$\begin{aligned} &= (\vec{u}_x)_s\, s_x + (\vec{u}_x)_t\, t_x + (\vec{u}_y)_s\, s_y + (\vec{u}_y)_t\, t_y \\ &= (\vec{u}_s s_x + \vec{u}_t t_x)_s\, s_x + (\vec{u}_s s_x + \vec{u}_t t_x)_t\, t_x + \\ &\quad (\vec{u}_s s_y + \vec{u}_t t_y)_s\, s_y + (\vec{u}_s s_y + \vec{u}_t t_y)_t\, t_y \end{aligned}$$

Hence, by the chain rule:

$$\begin{aligned} &= (\vec{u}_{ss} s_x + \vec{u}_s s_{xs} + \vec{u}_{ts} t_x + \vec{u}_t t_{xs})\, s_x + \\ &\quad (\vec{u}_{st} s_x + \vec{u}_s s_{xt} + \vec{u}_{tt} t_x + \vec{u}_t t_{xt})\, t_x + \\ &\quad (\vec{u}_{ss} s_y + \vec{u}_s s_{ys} + \vec{u}_{ts} t_y + \vec{u}_t t_{ys})\, s_y + \\ &\quad (\vec{u}_{st} s_y + \vec{u}_s s_{yt} + \vec{u}_{tt} t_y + \vec{u}_t t_{yt})\, t_y \end{aligned}$$

Combining terms and reversely applying (14) for s_{xx}, s_{yy}, t_{xx}, t_{yy} yields:

$$\begin{aligned} &= \vec{u}_{ss}\left(s_x^2 + s_y^2\right) + \vec{u}_{tt}\left(t_x^2 + t_y^2\right) + \\ &\quad \vec{u}_s\left(s_{xx} + s_{yy}\right) + \vec{u}_t\left(t_{xx} + t_{yy}\right) + 2\vec{u}_{st}\left(s_x t_x + s_y t_y\right) \end{aligned}$$

Approximating this equation by finite differences in the underlying uniform grid, where $\delta_s = \delta_t = 1$, and plugging back into (17), we can finally find the iteration rule for $\vec{u}$:

$$\begin{aligned} \vec{u}'_{i,j}\left[\beta + 2\left(k_s + k_t\right)\right] = \alpha f + \\ \left(\vec{u}_{i+1} + \vec{u}_{i-1}\right) k_s + \left(\vec{u}_{i+1} - \vec{u}_{i-1}\right) k_{ss} + \\ \left(\vec{u}_{j+1} + \vec{u}_{j-1}\right) k_t + \left(\vec{u}_{j+1} - \vec{u}_{j-1}\right) k_{tt} + \\ \left(\vec{u}_{i+1,j+1} + \vec{u}_{i-1,j-1} - \vec{u}_{i+1,j-1} - \vec{u}_{i-1,j+1}\right) k_{st} \end{aligned} \tag{18}$$

where:

$$\begin{aligned} k_s &= s_x^2 + s_y^2 & k_t &= t_x^2 + t_y^2 \\ k_{st} &= \left(s_x t_x + s_y t_y\right)/2 & \alpha &= 1/\nu h \\ k_{ss} &= \left(s_{xx} + s_{yy}\right)/2 & \beta &= 1/\nu h \\ k_{tt} &= \left(t_{xx} + t_{yy}\right)/2 & f &= \vec{u}_{i,j} \end{aligned} \tag{19}$$

Note that k_s, k_t, k_{st}, k_{ss} and k_{tt} can all be precalculated. Nevertheless, the iteration becomes somewhat expensive due to the need to access all eight cell neighbors. This cost is also present in [Stam 2003].

4.4 Projection step

The projection step starts by calculating the divergence of the velocity field. This can be done in a very straightforward manner by using (14).

The Jacobi iterations to obtain the pressure field should be transformed in a way analogous to that of the diffusion step. Indeed, (18) can be applied directly if we replace $\vec{u}$ and $\vec{u}'$ with p and p', respectively, and use the following parameter values:

$$\alpha = -1,\ \beta = 0,\ f = \nabla \cdot \vec{u}$$

4.5 Transport step

The transport of particles by the fluid flow is achieved by integrating their positions forward in time. This process follows the same rules and adaptations described for the backtracking during advection.

It should be noted that particle coordinates must be stored directly in grid coordinates, since there is no straightforward way to convert a position from world coordinates to them (that would probably require solving a nonlinear system of two equations). The opposite conversion, on the other hand, is possible by interpolating grid node positions at the particle coordinates, an operation that will be required if the particles are to be rendered later.

4.6 Boundary conditions

In our simulation, we chose to describe border and obstacle conditions in terms of the velocities tangent and orthogonal to the boundary. A wall cell, for example, should ensure that the orthogonal velocity entering or leaving it is zero, while letting the tangent velocity vary according to its roughness. This means that boundary conditions are applied to velocities in grid coordinates, which are always aligned with the corresponding surface. On the other hand, since velocities must be stored back in world coordinates, the Jacobian matrix must be used both ways in the process. Furthermore, if a boundary indicates an entrance or exit of fluid, its required velocity magnitude should also be ensured in world coordinates, despite its direction depending on the grid orientation. For example, if a fluid enters the domain through a diagonal boundary orthogonal to the grid coordinate s, that's the direction the velocity should be enforced in. If the entrance speed is u, however, this exact magnitude should be enforced in the world (x, y) frame.

5 Implementation

Our implementation is strongly based on the work by Harris [Harris 2004], making extensive use of the GPU and standard OpenGL textures and shaders.

As mentioned earlier, we consider the fluid domain in the world (x, y) space to be mapped to an underlying uniform grid with (s, t) coordinates. Each cell is identified by an $[i, j]^T$ index, has unit size in grid space, and center $\vec{x} = [x, y]^T$ in world space. The grid geometry is given by a simple input array of its node coordinates.

We use floating-point textures to store every quantity sampled throughout the fluid domain. That includes fluid properties like velocity (in world space) and pressure, secondary properties like divergence and curl, scalar quantities like the density of some colored inks for visualization, external influences like applied forces and ink sources, and all precalculated grid properties, such as the Jacobian matrices, their inverses, and the k terms in (19).

In order to control the behavior of the fluid near boundaries, we adjust the pressure and velocity for each border cell based on its corresponding internal neighbor. For the cells in the entire lower boundary of the grid, for example, this neighbor is the cell directly above it. As explained in [Foster and Metaxas 1997], pressure values should usually be copied unaltered from the interior to the border cells to keep the effect on the flow minimal. The velocity, on the other hand, can be controlled to achieve different effects.

We use two textures to control boundary conditions. The first one stores, for each grid cell that represents a boundary, the offset in grid coordinates necessary to find the corresponding internal neighbor (cells inside the fluid domain and inside obstacles receive special values instead to indicate that). The second stores a tangential scale factor and an orthogonal absolute offset for the velocity. This allows a border cell to have an arbitrary orthogonal velocity entering or leaving the fluid domain, while its tangent velocity is calculated as a scalar times the tangent velocity of the internal neighbor cell, resulting in either a more laminar or more turbulent flow.

Finally, we use vertex buffer objects to hold grid node and center coordinates in world space, as well as node texture coordinates in grid space. These are used for various visualization modes of the grid, fluid properties, inks and particles. The particles themselves could have their positions stored in buffer objects as well, but our current implementation uses another texture for that, allowing a maximum number of particles equal to the number of cells in the grid.

Our fluid simulation evolves by using GLSL shaders, one for each step of the solver. The implementation follows the work by Harris [Harris 2004], with the adaptations from Section 4. Additional shaders are also used for Lagrangian particle tracking, bilinear interpolation (when GPU does not filter floating-point textures), and various visualization modes.

6 Results and Discussion

In this section, we present example grids, describe validation experiments and discuss the obtained results in terms of correctness and performance. All non-regular grids were simulated as riverbeds, with the fluid entering from the left and exiting on the right with the appropriate speed to ensure mass conservation.

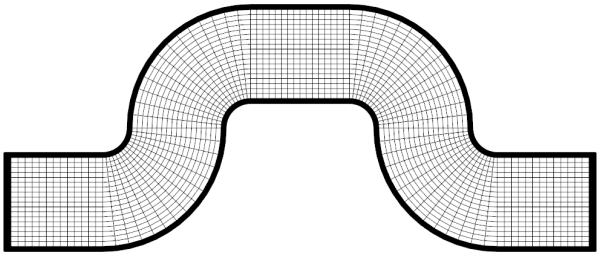

Figure 4: *Grid geometry for a curved path.*

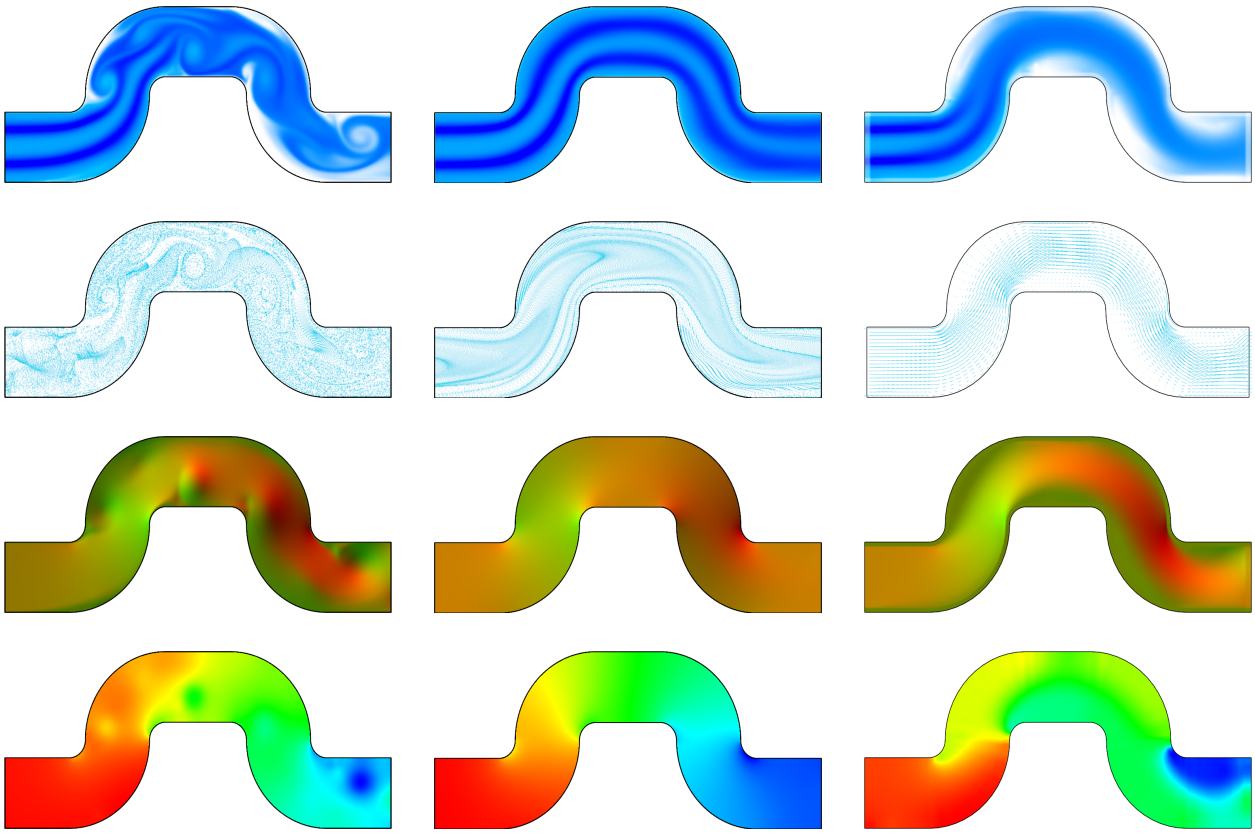

Figure 5: *Results obtained for the curved path geometry. From top to bottom: blue ink advected by the flow, massless particles transported by it (or velocity vectors in the third column), fluid velocity in the x (red) and y (green) directions, and relative pressure (low as blue, high as red). From left to right: refined (500×100) grid with rough boundaries, refined grid with smooth boundaries and added fluid viscosity, and coarse (100×20) grid with rough boundaries.*

6.1 Validation on regular grid

The most trivial validation of our method is to apply it to a standard regular grid. The resulting fluid behavior should be identical to the one obtained by the original Stable Fluids algorithm. Mathematically, the Jacobian matrix for a regular grid with cell size $[\delta_x, \delta_y]^T$ is given by (20). It can be verified that plugging this value into the equations derived in Section 4 does reduce them to the original versions. Experimentally, the predicted behavior was also verified in all of our tests.

$$J = \begin{pmatrix} x_s & y_s \\ x_t & y_t \end{pmatrix} = \begin{pmatrix} \delta_x & 0 \\ 0 & \delta_y \end{pmatrix} \tag{20}$$

6.2 Curved path

The grid shown in Figure 4 was designed to test the fluid's behavior flowing through a path with curved boundaries.

Some of the simulation results are presented in Figure 5, which shows very convincing flows. In the left column, we can see that the rough borders and the fluid's inertia cause the formation of vortexes just around and past sharp concave corners, where they are intuitively expected. Moreover, from the last row, we can also confirm that these areas correctly have a lower pressure than their surroundings. Directing our attention to the middle column, we can see a well-behaved viscous laminar flow inside a smooth boundary. Note that sharp corners still produce a small pressure decrease. Finally, the right column shows how the overall fluid behavior is the same on a lower-resolution grid, although vortexes and other fine details tend to disappear.

6.3 Constricted path

The next grid, presented in Figure 6, resembles a Venturi tube. It was designed to test the fluid's behavior when flowing through a constriction.

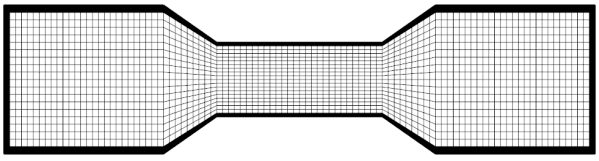

Figure 6: *Grid geometry for a constricted path.*

The results of this simulation are presented in Figure 7. For a laminar flow, a higher velocity is expected inside the constricted region to keep the mass flow per area constant. Additionally, from the hydrodynamics theory, the fluid inside the constriction should also exhibit a lower pressure. These properties can be verified

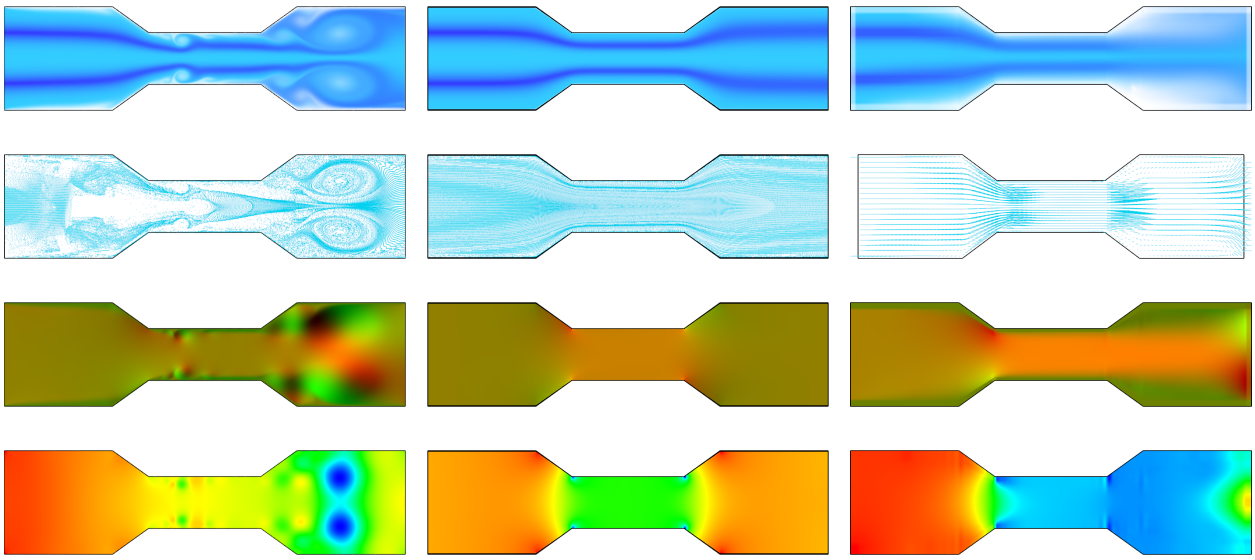

Figure 7: *Results obtained for the constricted path geometry. From top to bottom: blue ink advected by the flow, massless particles transported by it (or velocity vectors in the third column), fluid velocity in the x (red) and y (green) directions, and relative pressure (low as blue, high as red). From left to right: refined* (1000×100) *grid with rough boundaries, refined grid with smooth boundaries and added fluid viscosity, and coarse* (100×20) *grid with rough boundaries.*

in the bottom two rows of the middle column, where turbulences are minimal. In our experiments, we also numerically confirmed the ratio of the velocity increase. For example, for a half-width section, the fluid correctly flows at approximately double the full-width section speed. In contrast, it is interesting to note the different behavior observed in the left and right columns. When the fluid has no viscosity at all, it tends to rush right past the constriction, tunneling through the thicker section with little loss of velocity. Combined with the rough boundaries (left column), large vortexes are then generated, as shown in the first two rows.

6.4 River-like path

The last test grid geometry was designed as a representation of a natural river, possibly with internal obstacles like rocks and small islands. The basic grid is presented in Figure 8.

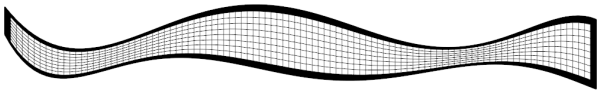

Figure 8: *Grid geometry for a river-like path.*

The river-like geometry was also altered in two different ways: first, we included a circular obstacle in the beginning of the widest section; second, we forked and rejoined the path at some points. The resulting boundary shapes and their respective simulation results can be seen in Figure 9.

It is important to note that our model does not support actual holes in the grid representation, since there is always a full regular grid underneath. Instead, to achieve the kind of effect described here, we must deform the cells in appropriate grid areas (usually rectangles in the grid space) so that they approximate the shape of the desired internal obstacles. Then, the outermost layer cells must be marked as walls and have boundary conditions associated with them. In Figure 9, these apparently missing obstacle cells are simply hidden to provide a better visual appearance.

As for the simulation results, in the figure's left column we can observe a low pressure zone in the most constricted section of the river and some interesting vortexes formed as a result of the boundary's shape and roughness. In the middle column, we can clearly notice the perturbation of the flow created by the obstacle, as some vortexes are generated behind it. Finally, the right column shows how the flow is forked and rejoined in a natural-looking way.

6.5 Performance

We measure the performance of our proposed method and compare it with that obtained by a standard Stable Fluids solver [Stam 1999] implemented in an analogous manner.

The most computationally intensive steps of our integrator are the Jacobi iterations, which therefore impose a bound on our efficiency. Fortunately, for animation purposes, the number of iterations need not be very high: 20 to 50 should be enough for the viscosity and diffusion steps, and 40 to 80 provide good results for the pressure computation [Harris 2004].

Table 1 shows the frame rates we achieved for the simulation of fluids over regular grids using our proposed method and the original Stable Fluids solver. We enabled the full simulation of the fluid, along with a set a ink densities and particle positions, using a fixed 20 it-

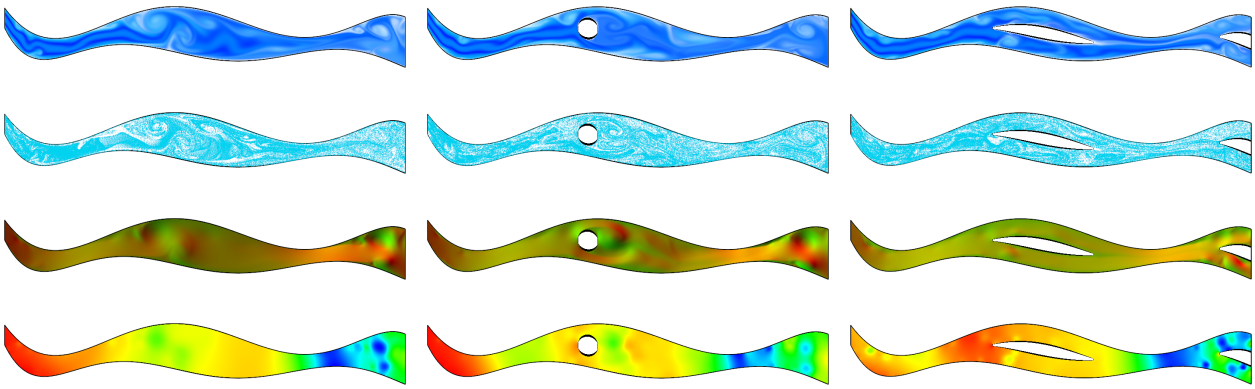

Figure 9: *Results obtained for the river-like path geometry with refined* (1000×100) *grids and rough boundaries. From top to bottom: blue ink advected by the flow, massless particles transported by it, fluid velocity in the* x *(red) and* y *(green) directions, and relative pressure (low as blue, high as red). From left to right: free path, circular obstacle, and forked path.*

erations for the viscosity and diffusion steps. The frame rates also include the rendering of ink alone. All measurements were performed on a machine with an Intel Core 2 Quad Q8400 2.66GHz processor, 4GB of RAM and a GeForce 9600 GT graphics card.

Table 1: *Performance measurements comparing our proposed method with the original Stable Fluids solver. The parentheses indicate the grid dimensions.*

Pressure Iterations	Our Method (300×300)	Stable Fluids (300×300)	Our Method (300×130)
10	39 fps	79 fps	80 fps
25	24 fps	46 fps	48 fps
50	19 fps	38 fps	39 fps
100	14 fps	28 fps	28 fps

As can be seen in the Table 1, our method has approximately doubled the computational cost of the original Stable Fluids solver for the same grid size. However, the extra freedom of the parameterization can eliminate the need to oversample the fluid domain near its boundaries. Therefore, if the total number of cells required by the parameterized grid is less than half of that needed for an equivalent regular grid, we can actually achieve better performance than the standard algorithm.

7 Conclusion

In this paper, we extended the work on Stable Fluids by Stam [Stam 1999] to animate fluids confined in two-dimensional domains with arbitrarily-shaped boundaries. We employed a simple uniform grid to drive the simulation, in parameter space, of a structured discretization of these domains. In order to accomplish this, the Jacobian matrices that relate world and parameter spaces were derived using finite differences. As a result, our approach was able to deliver efficient and convincing flow simulation on complex domains, as in the demonstrated examples of winding riverbeds with obstacles and forks.

It is interesting to note that, despite having a higher computational cost per-cell than the original Stable Fluids solver, our proposed method can produce convincing results for curved-boundary domains with a relatively small number of cells, which greatly compensates that cost and allows for quite fast simulations. Moreover, with an approach based on regular grids, the fluid behavior should not be depicted as accurately, even with a large number of cells or adaptive refinement schemes.

The proposed method can be easily extended to 3D. As efficiency may be a constraint for complex 3D domains, we plan to investigate the use of a CUDA-based implementation and the replacement of the Jacobi iteration method by a more efficient approach for solving the Poisson equations, either a multigrid method [McAdams et al. 2010] or a conjugate-gradient method with an incomplete Cholesky preconditioner [Golub and Van Loan 1996].

Acknowledgments

We would like to thank CNPQ and Faperj for the funding received during this project.

References

ARIS, R. 1989. *Vectors, tensors, and the basic equations of fluid mechanics*. Dover books on engineering. Dover Publications.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. In *ACM SIGGRAPH 2007 papers*, ACM, SIGGRAPH '07.

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

CHENTANEZ, N., FELDMAN, B. E., LABELLE, F., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2007. Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 219–228.

CHORIN, A., AND MARSDEN, J. 1993. *A mathematical introduction to fluid mechanics*. Springer-Verlag.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. In

Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM, SIGGRAPH '02, 736–744.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '01, 15–22.

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. In *ACM SIGGRAPH 2005 Papers*, ACM, SIGGRAPH '05, 904–909.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '01, 23–30.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, 181–188.

GOLUB, G. H., AND VAN LOAN, C. F. 1996. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. In *ACM SIGGRAPH 2005 Papers*, ACM, SIGGRAPH '05, 973–981.

HACKBUSCH, W. 1985. *Multi-grid methods and applications*. Springer series in computational mathematics. Springer.

HARLOW, F. H., AND WELCH, J. E. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. In *The Physics of Fluids 8*, 2182–2189.

HARRIS, M. J. 2004. *Fast Fluid Dynamics Simulation on the GPU*. GPU Gems. Pearson Higher Education, ch. 38.

HONG, J.-M., AND KIM, C.-H. 2005. Discontinuous fluids. In *ACM SIGGRAPH 2005 Papers*, ACM, SIGGRAPH '05, 915–920.

HORVATH, C., AND GEIGER, W. 2009. Directable, high-resolution simulation of fire on the gpu. In *ACM SIGGRAPH 2009 papers*, ACM, SIGGRAPH '09, 41:1–41:8.

JOHANSEN, H., AND COLELLA, P. 1998. A cartesian grid embedded boundary method for poisson's equation on irregular domains. *Journal of Computational Physics 147*, 1, 60 – 85.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. In *ACM SIGGRAPH 2006 Papers*, ACM, SIGGRAPH '06, 820–825.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers*, ACM, SIGGRAPH '04, 457–462.

LOSASSO, F., IRVING, G., GUENDELMAN, E., AND FEDKIW, R. 2006. Melting and burning solids into liquids and gases. *IEEE Transactions on Visualization and Computer Graphics 12*, 343–352.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. In *ACM SIGGRAPH 2006 Papers*, ACM, SIGGRAPH '06, 812–819.

MCADAMS, A., SIFAKIS, E., AND TERAN, J. 2010. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 65–74.

MULLEN, P., CRANE, K., PAVLOV, D., TONG, Y., AND DESBRUN, M. 2009. Energy-preserving integrators for fluid animation. In *ACM SIGGRAPH 2009 papers*, ACM, SIGGRAPH '09, 38:1–38:8.

RASMUSSEN, N., ENRIGHT, D., NGUYEN, D., MARINO, S., SUMNER, N., GEIGER, W., HOON, S., AND FEDKIW, R. 2004. Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 193–202.

ROBINSON-MOSHER, A., SHINAR, T., GRETARSSON, J., SU, J., AND FEDKIW, R. 2008. Two-way coupling of fluids to rigid and deformable solids and shells. In *ACM SIGGRAPH 2008 papers*, ACM, SIGGRAPH '08, 46:1–46:9.

ROBLE, D., ZAFAR, N. B., AND FALT, H. 2005. Cartesian grid fluid simulation with irregular boundary voxels. In *ACM SIGGRAPH 2005 Sketches*, ACM, SIGGRAPH '05.

SHAH, M., COHEN, J. M., PATEL, S., LEE, P., AND PIGHIN, F. 2004. Extended galilean invariance for adaptive fluid simulation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, 213–221.

SHI, L., AND YU, Y. 2002. Visual smoke simulation with adaptive octree refinement. Tech. rep., University of Illinois.

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

STAM, J., AND EUGENE, F. 1995. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, 129–136.

STAM, J. 1998. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '98, 395–404.

STAM, J. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, 121–128.

STAM, J. 2003. Flows on surfaces of arbitrary topology. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, 724–731.