These simulated fluids move around randomly while interacting with such everyday objects as a human hand in surprisingly realistic swirls, flows, and vortices.

Interacting with Smoke and Fire in Real Time

Jos Stam

WINDOW BLOWN OUT BY EXPLOSION (INSET SEQUENCE) AND SWIRLING HOT GASES (CREATED BY GARY YNGVE, JAMES O'BRIEN, AND JESSICA HODGINS, GEORGIA INSTITUTE OF TECHNOLOGY)

THE SIMULATION OF FLUIDS IS ONE OF THE MOST challenging problems in a number of engineering fields. Here, I offer a solution in the form of a rapid method I developed for computing the motion and appearance of fluids, allowing users to interact in real time with virtual smoke and fire.

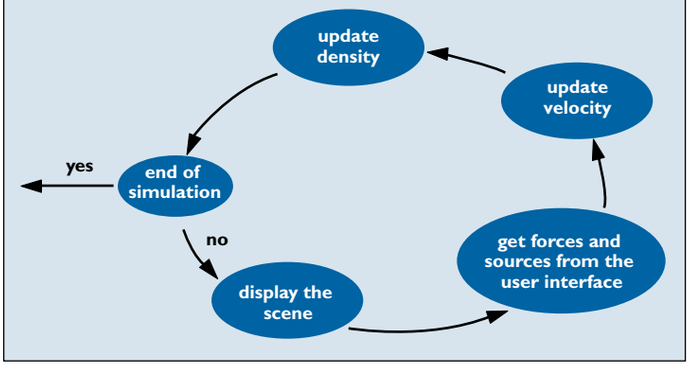
My work is motivated by such computer graphics applications as designing virtual reality games and creating special effects in the entertainment industry. Ideally, a user immersed in a virtual environment should be able to interact with natural phenomena in real time. Imagine an actor traversing a wall of virtual smoke swirling to the movement of an arm. This work will also prove of interest to other fields, including manufacturing design, architecture, and education. Although the related simulations are not strictly accurate, they capture the essential visual characteristics associated with fluids at interactive speeds.

This model is desirable in, say, educational software designed to familiarize students with the basic behavior of fluids. It might assist a designer in evaluating the effect of fluid flows past automobile, train, and airplane bodies in the early stages of shape design. It could also be used in architectural design to estimate the general circulation of air in a building. These potential applications, along with many others, have been suggested by professionals with expertise beyond the field of computer graphics. Although they were aware of the model's inaccuracies, they have been amazed by both the speed and the visual quality of the related fluid demonstrations. So, although my focus here is on computer graphics applications, I expect it to be of use to researchers in many other areas.

The physics of fluids provides the natural framework for these simulations. Fluids have been studied in many disciplines for the last thousand years and are now understood fairly well. Mathematically, their behavior is described by a set of equations known as the Navier-Stokes equations, arrived at more than 150 years ago by the French engineer Claude Navier and the Irish mathematician George Stokes. These equations have been instrumental in the development of my solver.

In fact, the key idea behind it was prompted when I noticed the visual similarity between the Navier-Stokes equations and a simpler set of equations for which I had already developed a rapid solver. Note, however, that my algorithm can be understood without precise knowledge of these equations; wherever I state the equations in the article, I do so only to get you to recognize a particular—and as it turns out critical—similarity between two different ideograms.

Figure 1. Overview of the solver.



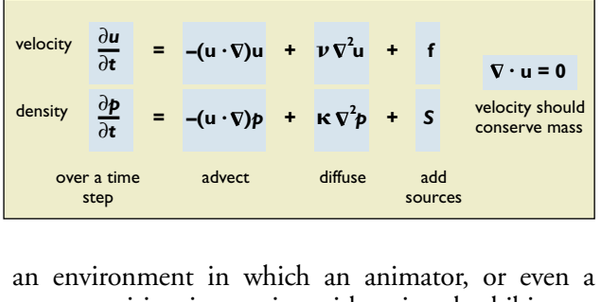
The problem of simulating realistic fluids is equivalent to writing good solvers for the Navier-Stokes equations. Solving them is central to research in areas as diverse as aeronautics, combustion science, biophysics, and mechanical engineering. However, they are also notoriously difficult to solve, due to their nonlinearity, complicating the relationship between causes and effects. Because there is no general analytic solution to them, researchers have made many different simplifying assumptions appropriate to their disciplines. Engineering fields, including aeronautics, can often ignore the complex visual appearance of a fluid, so long as they accurately predict average quantities (such as the maximum stress on an airplane wing).

In computer graphics, on the other hand, how a fluid *appears* is of primary interest. A good fluid solver in computer graphics should exhibit all the characteristics of real fluids, such as swirling flows around bodies. Moreover, the speed of the simulator is crucial, since animators want close-to-real-time feedback in a virtual environment. The novel fluid solver I describe here seeks to meet both these requirements. To do so, I departed from conventional wisdom in the computational fluid dynamics literature to design a fluid solver in light of the specific needs of computer graphics. Few models in computer graphics attempt to solve the Navier-Stokes equations directly. Early models considered only those techniques specific to 2D fluids [6, 11]. More recently, Nick Foster of Pacific Data Images and Dimitris Metaxas of the University of Pennsylvania showed that interesting 3D flows can be animated using inaccurate solvers [4] (see Foster and Metaxas's "Modeling Water for Computer Animation" in this section). But because their method requires a small time step, these solvers are relatively slow. My solvers are unconditionally stable and can be advanced over time at any speed. Figure 1 shows the solver's basic structure, which consists of a single loop repeated for each time step.

My approach falls into the class of the so-called "Semi-Lagrangian schemes" introduced in the early 1950s [2]. They are rarely used in engineering applications, because they suffer from too much numerical dissipation; the simulated fluid tends to dampen more rapidly than an actual fluid. This shortcoming is less of a problem in computer graphics applications, especially in an interactive system in which the flow is "kept alive" by an actor applying external forces. In fact, a flow that does not dampen at all might be too chaotic and difficult to control.

As the results demonstrate, I am able to produce nice swirling flows despite the numerical dissipation. Moreover, these new solvers have been integrated into an environment in which an animator, or even a museum visitor interacting with a virtual exhibit, can apply forces to a virtual fluid at interactive frame rates—an effect never achieved before. Additional visual complexity can also be added at little cost by "dressing up" fluid flows with textures.

Figure 2. The Navier-Stokes equations.



Textures are widely used in computer graphics, typically to add a more natural look inexpensively to otherwise smooth surfaces. In the case of my solvers, I use 3D fractal-like texture maps to add visual detail to the densities of smoke moving through the fluids. Similar techniques are well known in computer graphics as "solid texture map-

ping" [3]. The novelty of this approach is to let the textures move naturally along the fluid flow.

A Fluid's Velocity

Fluids are everywhere. Probably their most dramatic manifestation is as a flow of water. The air surrounding us is also a fluid, one whose motion is governed by many external factors (forces), including heat sources, wind fans, and the movement of an actor. The state of a volume of air is specified entirely by its density, temperature, and velocity. Air at rest, such as in a very quiet room, has zero velocity, constant temperature, and constant density. In more interesting environments, these quantities change from point to point and over time. How these quantities change in various situations and environments is the subject of fluid mechanics. I assume that the density and the temperature of the fluid are constant. Constant-density fluids approximate most fluids quite well, while the effects of temperature, including buoyancy, are modeled easily using heat-like forces. An animator needs to describe only how the fluid's velocity changes over time.

The main application of a fluid solver in computer graphics is to use the velocity field to move things around realistically. Examples include smoke, clouds, leaves, and flags. Consequently, in addition to a fluid solver, graphic-effects animators also need a model for the fluid-object interaction. Here, I emphasize the effect of a fluid on "fuzzy" substances like smoke and clouds that are modeled using a density function. For example, clouds are modeled by assigning a density of water droplets to each point in the environment. Where there are no clouds, the density is simply zero. However, the density of such a substance should not be confused with the density of the fluid—air, in this case—we assume to be constant.

The velocity of the fluid and the density of the substance are both defined on either a 2D or a 3D grid, depending on the application. These grids are defined entirely by their position, size, orientation, and resolution in each coordinate, and they have to be large enough to encompass the region of interest in the virtual environment. The solver then updates the grids at time events separated by a fixed time step. Before each update, the solver displays the virtual scene from a particular point of view, either inside the scene or outside looking in. Also, the solver takes input from an interactive user interface to construct force fields and density sources from an actor's movements. Many user interfaces can be devised and added to the solver, though each requires the ability to track the movements of the actor in 3D. In my implementation, the forces and sources are returned to the application in a set of grids. The forces set the fluid in motion and provide the input to the fluid's velocity solver. The sources and the fluid's velocity together control the evolution of the density of the substance.

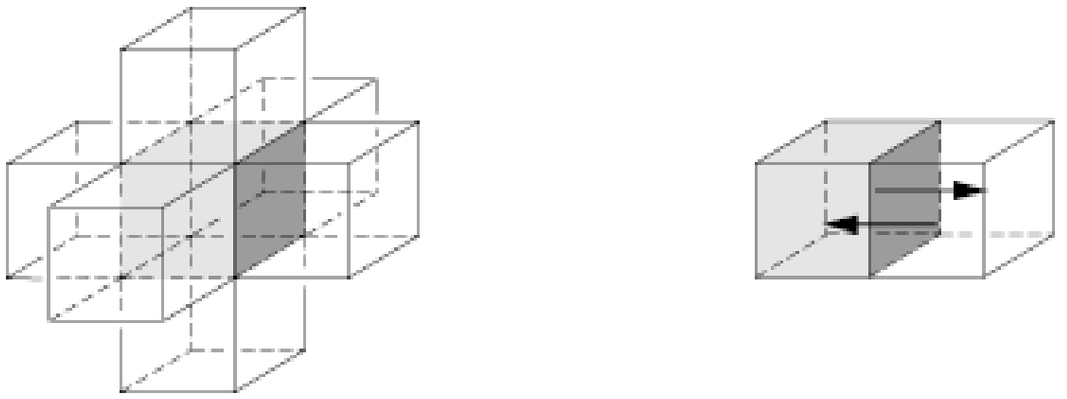
These solvers are based on a precise mathematical formulation of the evolution of a fluid—the Navier-Stokes equations in Figure 2. The equation on the top of the figure models the evolution of the fluid's velocity; the one on the bottom describes the evolution of the density of a substance immersed in the fluid. The main reason I show these equations is to point out how visually similar they are. This similarity is apparent even to someone completely unfamiliar with the equations. It gave me the insight I needed to go on to invent the solvers. In fact, I first developed a solver for a density moving through a static velocity field and then realized the method could be extended to evolve the velocity field as well.

This approach is best explained by first describing the density solver. (For a formal treatment of the related mathematics, see [10], as well as [1], which was instrumental in the development of the solver and for the mathematical structure of the Navier-Stokes equations, and [8] for a more hands-on approach to solving the equations.)

The equation at the bottom of Figure 2 is simply a precise mathematical formulation of the physics that rule the evolution of density over time. It states that over time the density changes for the three reasons embodied in the three terms on the right side of the equation. The first term states the density should flow naturally with the fluid's velocity; the second means the density diffuses over time, mainly due to the small turbulences of the velocity field not modeled explicitly; and the last states the density increases due to external sources. These sources are provided by the user interface and correlated to certain motions of the actor. For example, the actor could light up a virtual cigarette with the tip corresponding to a source of smoke density. The solver resolves each term sequentially. The source term is the easiest to resolve in the following way: multiply the source grid (provided by the user interface) by the time step and add it to the grid storing the densities. This method increases the density precisely in the areas where there are sources.

The most straightforward way to handle diffusion is to consider exchanges only between adjacent cells (see Figure 3). During the diffusion step, each cell transfers a percentage of its density to its six closest neighboring cells. Simultaneously, the cell receives a percentage of its neighbor's density. This percentage is inversely proportional to the area of the cell's face and proportional to the time step and the diffusion rate. Although this method is very easy to implement, it

Figure 3. A simple but unstable solution for the diffusion step. Cells exchange densities with only their six nearest neighboring cells (left), depending on the area of a face of the grid cell (right).



doesn't always work. Problems arise when the cell size is too small or when the time step or the diffusion rate is too high. In these situations, a cell may have to give away more than 100% of its density—an impossibility. Consequently, the densities stored in the grid can become negative or grow without bounds. In other words, instability takes over, rendering the simulation useless. This method fails because the exchange of density requires a larger neighborhood of cells.

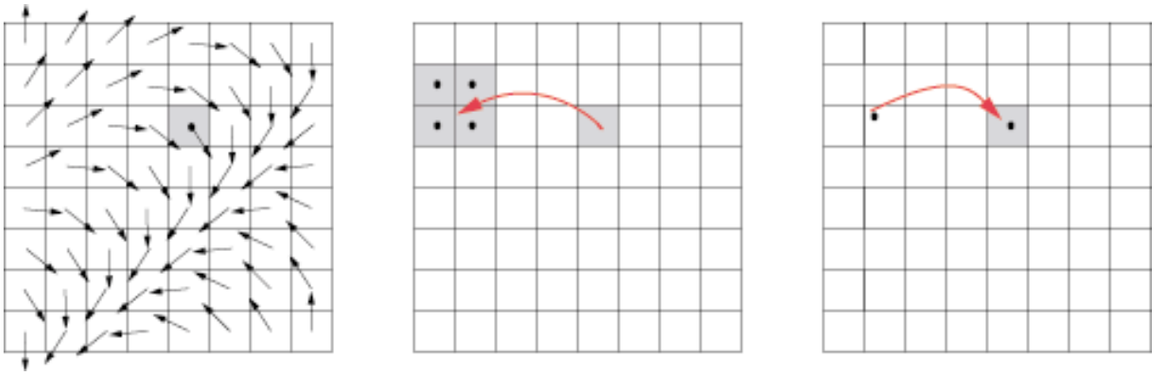
Any technique that performs only explicit updates between adjacent cells will fail. To obtain a stable solver, I use a different approach, known as "implicit time stepping." The basic idea is to reverse the diffusion step in time. The solver seeks a set of new densities that, when diffused backward in time, yield the initial set of densities. Finding these new densities requires the solution of a system of linear equations at every time step. This system is spare and can be solved efficiently using one of the many existing techniques in the literature [7]. I opted for an iterative "conjugate gradient" method. In practice, it turns out that solving a linear system with a large implicit time step costs the application far less computational overhead than taking many small explicit time steps to avoid instabilities. This implicit technique is quite standard.

The main application of a fluid solver in computer graphics is to use the velocity field to move things around realistically .

The most challenging step in the density solver is the so-called "advection step," which corresponds to the first term on the right side of the density equation in Figure 2. This term states that the density must flow along the velocity of the fluid. This flow-velocity calculation is similar to the diffusion step in that densities are transferred between cells. In the advection step, the transfer of densities is greatest in the direction of the velocity. Consequently, the solver resolves it using a similar implicit time-stepping method. But it turns out that there is an alternative solution to this problem that easily extends to a solver for the fluid's velocity. The key observation is that the advection step is solved simply if the density is represented as a set of particles, each carrying a certain amount of density. In this case, the solver would then move each particle through the velocity field, carrying the density along that field.

The next idea is both simple and elegant. The solver constructs a new grid of densities by computing the initial positions of the particles that, over a single time step, end up exactly in the centers of the grid cells. The solver then samples the density at these initial positions through linear interpolation from the density grid of the previous time step. These interpo-

Figure 4. Advection step. For each grid cell, a particle is traced backward in time through the velocity field. A density is interpolated from its four neighbors; the new density is then transferred to the grid cell.



lated densities are then transferred to the corresponding cells in which the particles end up. Figure 4 shows how this is implemented in 2D. First, the solver traces a particle starting at the center of a grid cell backward in time through the fluid's velocity. It then interpolates the density from its four neighboring cells. Finally, it sets the new density of the cell to the interpolated density. These three steps are performed automatically by the solver for each grid cell. The method is stable because it uses linear interpolation; the new values are never larger than the maximum density of the previous time step. Consequently, the density values remain bounded no matter how large the time step.

A Stable Solver

Return now to Figure 2 and its equations for a fluid's velocity and the density of a substance immersed in the fluid. The evolution of velocity and density over a time step is similar. Note that the two farthest-right terms in the equations are almost identical; over a time step, the velocity both diffuses and reacts to external sources. The sources correspond to a force field provided by the user interface as a grid of vectors. As in the case of density, the solver multiplies the force grid by the time step and adds it to the velocity. The diffusion step is resolved using the same implicit time-stepping technique described earlier for density. The difference is that the solver now has to solve three diffusion steps—one for each component of the velocity. The rate of diffusion corresponds to the viscosity of the fluid. Varying the viscosity thus allows a wide spectrum of fluids to be modeled, ranging from highly viscous glue-like substances to low-viscosity turbulent flows.

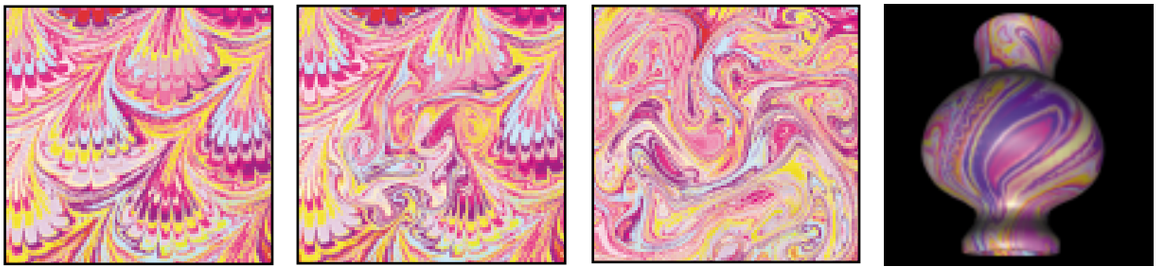
The advection step is handled in the same way. In the density case, the equation term states that the velocity is moving the density. However, in this—the velocity—case, we can interpret the advection step as the "velocity moving itself." It can be solved exactly as in the density case. Simply trace a particle from the center of each grid cell backward in time through the velocity field. Then interpolate the velocity at that location and transfer it to the grid cell. (Note that the advection procedure is exactly like the one in Figure 4.) It can be shown mathematically that this algorithm does indeed resolve the nonlinear advection term. The advantage of this method over others is that it is stable and easy to implement.

So far, I have ignored another physical constraint: that every fluid has to conserve mass, meaning the flow into a cell should be equal to the flow coming out of the cell. However, after the three previous steps—adding forces, diffusion due to viscosity, and advection—this is rarely the case. To force the velocity to conserve mass, it has to be "corrected" by subtracting a vector field from it. The "corrective field" satisfies a standard equation known as the Poisson equation using the same techniques employed for the diffusion term. Mass conservation is important for producing realistic visual results, since it forces the flow to swirl and form vortices, greatly enhancing the flow's realism.

Animations in 2D and 3D

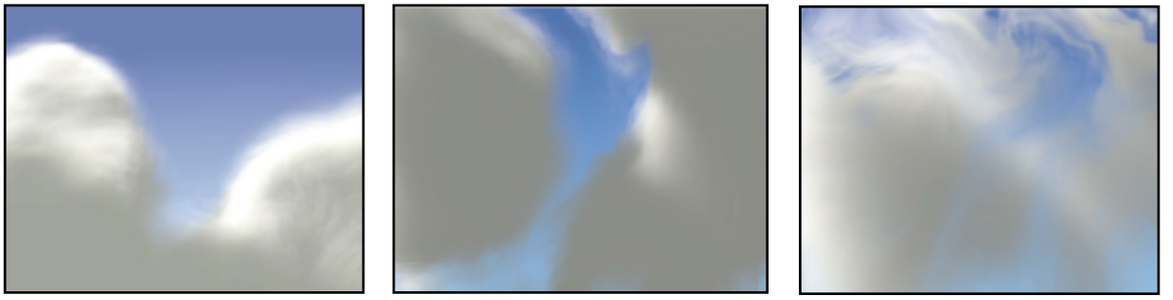
I have implemented two versions, one for 2D and one for 3D animations (see Figure 5). Most of these images are from sequences created interactively. In the 2D solver, the animator interacts with the fluid through a standard input device like a mouse. Forces and new densities are then painted into the simulation at the cursor's location. (The visual complexity of these simulations was greatly enhanced by multi-

Figure 5. Sample applications of the solver in 2D and 3D.



(a) Sequence created interactively with the 2D solver.
The results can be used to texture map a virtual object (right).

(b) Example of 2D flow around bodies immersed in a fluid.



(c) 3D simulation of clouds animated and rendered interactively.

(d) Using the Zcam (left), an actor interacts with the 3D solver in real time.

plying the density by a texture map; the texture can be any image.) When the entire grid is filled with a constant density and the fluid is at rest, the image is undistorted, as shown at the left of Figure 5a.

When external forces are applied, the image becomes warped and flows naturally along the fluid's velocity, also shown in Figure 5a. To achieve this effect, I supplemented the density field with two new scalar fields corresponding to the two texture coordinates. Initially, these texture coordinates varied linearly from 0 to 1 across the grid. During the fluid simulation, the density solver is applied to these texture coordinates as well, though without diffusion or sources. Figure 5a also demonstrates that 2D flows can be mapped to a virtual object. The solver gracefully handles flows around obstacles painted into the scene as well, so they swirl and flow naturally past these boundaries, as shown in Figure 5b.

The 3D solver needs a user interface allowing an actor or animator to specify 3D forces and sources. I wrote a system aimed specifically at animators who wish to choreograph special effects involving fluids and fuzzy densities, including smoke. In this environment, forces and sources are specified using 3D locators correlated with 2D input devices like a mouse. Interactive changes to the input allow animators to rapidly fine-tune a particular effect.

To display the density field, the solver takes advantage of specialized computer graphics hardware. The cloud simulation in Figure 5c was rendered in real time using the 3D hardware texture capabilities of an SGI Octane workstation. Notice that the solver also handles the effects of self-shadowing, through which a density casts shadows on other parts of its virtual environment and on itself. As in the solver's 2D implementation, an animator can add realistic detail to the density by moving 3D textures through the air fluid flow as well. I also experimented with the volumePro graphics card from Mitsubishi Electric Research Laboratory, allowing large volumes to be rendered in real time on a standard PC [9].

Figure 5d is a result of an ongoing collaboration with 3dvSystems, an Israeli company that has developed a camera called the Zcam that records both image and depth simultaneously in real time (www.3dvsystems.com). In the figure, the closest point to the camera is used as the moving location of sources in a fluid simulation. It shows an actor interacting with the fluid solver, using the tip of his finger to add densities and stir up the fluid.

Conclusion

Development of this fluid solver shows that real-time interaction with complex physical phenomena in a virtual environment is increasingly feasible. Its results were achieved not by augmenting the power of a particular application system's hardware but by designing algorithms in light of the special needs of computer graphics—speed, stability, and good visual results. This approach may allow other phenomena to be simulated rapidly as well. For example, other models for simulating liquids, such as water [5], may be accelerated using the ideas I've outlined here. I also wish to extend this model to other general substances, including clays and plastics. Modeling these materials rapidly would have a wide range of applications in entertainment, industrial design, and architecture. The equations themselves are more complicated, but I am convinced that a generalization of the technique will prove to be effective. C

References

1. Chorin, A. and Marsden, J. A mathematical introduction to fluid dynamics. *Texts in Applied Mathematics 4, 2d Ed.* Springer-Verlag, New York, 1990.
2. Courant, R., Isaacson, E., and Rees, M. On the solution of nonlinear hyperbolic differential equations by finite differences. *Commun. Pure and Appl. Mathem. 5 (*1952), 243–255.
3. Ebert, D., Musgrave, K., Peachy, D., Perlin, K., and Worley, S. *Texturing and Modeling: A Procedural Approach.* Academic Press Professional, 1994.
4. Foster, N. and Metaxas, D. Modeling the motion of a hot, turbulent gas. In *Computer Graphics Proceedings Annual Conference Series* (Los Angeles, Aug. 3–8, 1997), 181–188.
5. Foster, N. and Metaxas, D. Realistic animation of liquids. *Graph. Mod. Image Process. 58,* 5 (1996), 471–483.
6. Gamito, M., Lopes P., and Gomes, M. Two-dimensional simulation of gaseous phenomena using vortex particles. In *Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation* (Maastricht, The Netherlands, Aug. 28–Sept. 1). Springer-Verlag, Vienna, Austria, 1995, 3–15.
7. Golub, G. and van Loan, C. *Matrix Computations, 3rd Ed.* The Johns Hopkins University Press, Baltimore, Md., 1996.
8. Griebel M., Dornseifer T., and Neunhoeffer, T. *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. Society for Industrial and Applied Mathematics, Philadelphia, 1998.
9. Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., and Seiler, L. The VolumePro real-time ray-casting system. In *Computer Graphics Proceedings Annual Conference Series* (Los Angeles, Aug. 3–8, 1999), 251–260.
10. Stam, J. Stable fluids. *In Computer Graphics Proceedings Annual Conference Series* (Los Angeles, Aug. 3–8, 1999), 121–127.
11. Yaeger, L. and Upson, C. Combining physical and visual simulation: Creation of the planet Jupiter for the film 2010. In *Proceedings of SIGGRAPH'86* (Dallas, Tex., Aug. 13–18). ACM Press, New York, 1986, 85–93.

Jos Stam (jstam@aw.sgi.com) is a research scientist at Alias|Wavefront in Seattle, Wash.