Real-Time Fluid Simulation in a Dynamic Virtual Environment

Jim X. Chen
George Mason University

Niels da Vitoria Lobo, Charles E. Hughes, and J. Michael Moshell
University of Central Florida

Solving the 2D Navier-Stokes equations via a computational fluid dynamics method lets us map surfaces into 3D and achieves realistic real-time fluid surface behaviors.

Simulating physically realistic complex fluid behaviors in a distributed interactive simulation (DIS) presents a challenging problem for computer graphics researchers. Such behaviors include driving boats through water, stirring liquids, blending differently colored fluids, mixing insolubles such as oil and water, rain falling and flowing on a terrain, and fluids interacting. These capabilities are useful in computer art, advertising, education, entertainment, and training.

DIS denotes a broad field of simulation research and technology as well as a specific architectural approach, represented by the DIS communications protocol.[1] In this article we use the acronym DIS to designate any simulation conducted by distributed computation whose outputs must respond to changed inputs with the same timeliness the modeled system would exhibit.

Modeling and animating fluids have captured the attention of many graphics researchers. However, no one has achieved general fluid models that are physically realistic and computationally efficient for real-time animation. Fournier and Reeves,[2] Peachey,[3] and T'so and Barsky[4] proposed alternative models based on ocean wave equations. Their approaches avoid solving the differential equations of fluids and suit only ocean waves. Miller and Pearce[5] presented a particle system for animating viscous fluids that represents particles throughout the fluid's volume, incurring significant computational costs. Terzopoulos, Platt, and Fleischer[6] used molecular dynamics to model the process of solids transforming into liquids; their approach is also computationally expensive for obtaining the behaviors we desire.

Kass and Miller[7] animated fluid using simplified shallow water equations. Though they solved the equations in their differential form, Kass and Miller's method is not general enough to model complex phenomena entailing variations in the *Reynolds number*—which controls whether a fluid is laminar or turbulent—or that include moving (self-propelled) objects. Goss[8] used a particle system to model ship wakes in real time, but his approach was not based on physics and does not host any other fluid properties or behaviors besides artificial ship wakes.

Wejchert and Haumann[9] presented a model for inviscid irrotational flow, applicable only to limited situations of objects in a wind field. Stam and Fiume[10] proposed a method that models turbulent wind fields but lacks any physical basis.

To provide a physical foundation for general fluid animation, you must use the Navier-Stokes equations, which embody Newton's second law in fluids, and the governing equations of general fluid flow. Several researchers in computer graphics have acknowledged this.[7,9,10] However, none of the previous methods solved these equations because of the effort involved in deriving a solution method and the time needed to obtain a solution on commonly available workstations. While researchers in computational fluid dynamics (CFD) extensively study computational physical fluids models, their goals are constrained to obtain highly accurate and completely descriptive simulations of fluid behaviors.

Recently, we developed a CFD method using the Navier-Stokes equations.[11] It reduces the time and cost of computing these behaviors from the resolution's cube to the resolution's square and yields results in real time. Of course, the simulation speed depends on the calculated area's size and resolution. On a Silicon Graphics Onyx, the model runs at 20 frames per second with a fluid grid size of 140 by 64, which satisfies many real-time simulation applications. We use "real time" to mean that the frame rate of the physically based modeling and simulation occurs at an interactive rate of human perception. However, our physically based model ignores the fluid's depth and therefore cannot satisfy simulations in a networked virtual environment.

Instead of calculating the fluid behavior through a volume, that is, calculating the 3D Navier-Stokes equations, we computed the full incompressible 2D Navier-Stokes equations.[12] Then we raised the fluid's surface according to the corresponding pressures in the flow field, thus

0272-1716/97/$10.00 © 1997 IEEE

obtaining the surface points' third dimension. We can justify using the pressures to simulate the fluid surface fluctuations because higher pressures at the fluid's base cause taller columns of the surface above due to the fluid's incompressibility. (We discuss the technical justification for this in the section "3D fluid surface animation.") This method reduces the expense from the resolution's cube to the resolution's square without losing the 3D effects and the power of the Navier-Stokes equations. We have demonstrated the simulation of different kinds of fluid flows and their vector fields in real time on commonly available workstations, such as a Silicon Graphics Indigo. We have also simulated floating objects in fluids, moving (self-propelled) objects in fluids, and blending of fluids of different colors in real time by employing the fluid flow velocity and pressure field.

To simulate fluid flow in a dynamic virtual environment, we accounted for the changes of the fluid volume and boundaries when the fluid flows and accumulates. We also extended the general fluid model to let fluid generate and flow freely anywhere on a dynamic terrain surface. We demonstrated this by implementing a bulldozer, which can change the terrain's shape and thus the fluid's constraining boundaries, with the fluid following the modified terrain. With these extensions, the fluid model can integrate into a DIS and offer a reasonably realistic simulated environment.

We did not investigate fluids in a networked environment due to the lack of existing modeling and simulation techniques. Synchronizing physical activities such as fluid flows in a DIS proves important to guarantee fast and accurate simulation. In this article, we introduce a mechanism that uses a uniform time scale—proportional to the clock time and variable time-slicing—to synchronize physical models such as fluids in the networked environment for DIS.

Navier-Stokes equations

Navier-Stokes equations derive from Newton's second law.[13] The following equations are the Navier-Stokes equations for an incompressible flow:

$$\rho\frac{Du}{Dt}=-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+\rho g_x \quad \textbf{(1)}$$

$$\rho\frac{Dv}{Dt}=-\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right)+\rho g_y \quad \textbf{(2)}$$

$$\rho\frac{Dw}{Dt}=-\frac{\partial p}{\partial z}+\mu\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right)+\rho g_z \quad \textbf{(3)}$$

where

$$\frac{D}{Dt}=u\frac{\partial}{\partial x}+v\frac{\partial}{\partial y}+w\frac{\partial}{\partial z}+\frac{\partial}{\partial t},$$

$$u=\frac{dx}{dt},\ v=\frac{dy}{dt},\ w=\frac{dz}{dt},$$

ρ *is* the density, p is the pressure, μ is the viscosity, and g_x, g_y, and g_z are the gravity vectors.

The differential continuity equation for incompressible flow,

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0 \quad \textbf{(4)}$$

is used together with Navier-Stokes equations to determine the relationships between velocities and pressures.

We can put the Navier-Stokes equations (Equations 1, 2, and 3) and the differential continuity equation (Equation 4) into the following compact vectorial form:

$$\rho = \frac{D\mathbf{V}}{Dt}=-\nabla p+\mu\nabla^2\mathbf{V}+\rho\mathbf{g} \quad \textbf{(5)}$$

$$\nabla\bullet\ \mathbf{V}\ =\ 0 \quad \textbf{(6)}$$

where the gradient vector operator

$$\nabla=\frac{\partial}{\partial x}\mathbf{i}+\frac{\partial}{\partial v}\mathbf{j}+\frac{\partial}{\partial z}\mathbf{k},$$

$\mathbf{V}=u\mathbf{i}+v\mathbf{j}+w\mathbf{k}$, and $\mathbf{g}=g_x\mathbf{i}+g_y\mathbf{j}+g_z\mathbf{k}$. Equation 5 is the governing equation of fluid dynamics. Its solution is complicated by the nonlinear terms on the left-hand side and the requirement of simultaneous solution of velocities and pressures—it represents one of the major computational challenges in fluid mechanics. Navier-Stokes equations without external forces can be written in the dimensionless form

$$\frac{D\mathbf{V}}{Dt}=-\nabla p+\frac{1}{Re}\nabla^2\mathbf{V} \quad \textbf{(7)}$$

where *Re* is the *Reynolds number*. The Reynolds number is a parameter that indicates the fluid's viscosity. If the Reynolds number is relatively small, the fluid is viscous and the flow laminar; if it is large, the fluid tends to be inviscid and the flow turbulent. Therefore, the results of the Navier-Stokes equations with different Reynolds numbers correspond to the behaviors of different kinds of fluids.

Surface implementation

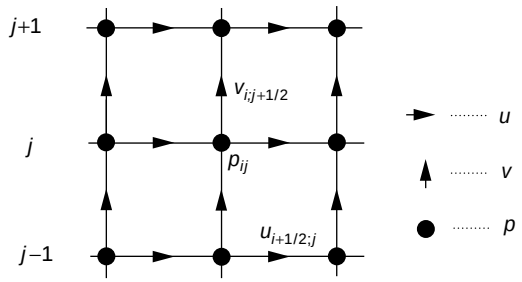
In our approach, we use the corresponding pressures in a 2D fluid flow field to simulate the fluid surface behaviors. Therefore, we only need to compute the 2D Navier-Stokes equations. We present the 2D discretization and computation method below.

Numerical discretization

Several approaches exist in computational fluid dynamics to solve the Navier-Stokes equations.[12] Here we employ a finite-difference solution technique that uses a penalty method,

$$\varepsilon p+\nabla\bullet\mathbf{V}=0,\ \varepsilon>0,\ \varepsilon\to 0 \quad \textbf{(8)}$$

instead of the divergence equation (Equation 6). Temam[14] proved that the solution of Equations 7 and 8 tends toward the solution of the Navier-Stokes equations (Equations 5 and 6 without external forces), when $\varepsilon\to 0$.



1 The staggered marker and cell mesh.

The spatial discretization of Equations 7 and 8 in 2D employs the staggered marker and cell mesh (see Figure 1). We consider an explicit discretization as follows:

$$u^{n+1}_{i+1/2;j} = u^n_{i+1/2;j} + \left(-a^n_{i+1/2;j} - \Delta^1_x p^n_{i+1/2;j} + \frac{1}{Re}\nabla^2_h u^n_{i+1/2;j}\right)\Delta t \quad \textbf{(9)}$$

$$v^{n+1}_{i;j+1/2} = v^n_{i;j+1/2} + \left(-b^n_{i;j+1/2} - \Delta^1_y p^n_{i;j+1/2} + \frac{1}{Re}\nabla^2_h v^n_{i;j+1/2}\right)\Delta t \quad \textbf{(10)}$$

$$p^{n+1}_{i;j} = \frac{\Delta^1_x u^{n+1}_{i;j} + \Delta^1_y v^{n+1}_{i;j}}{\varepsilon} \quad \textbf{(11)}$$

where i, j are fluid-flow field coordinates, n represents the current state, and $n+1$ represents the next state after a time-step of Δt.

The difference operators

$$\Delta^1_x,\ \Delta^1_y,\ \text{and}\ \nabla^2_h$$

are defined by

$$\Delta^1_x f_{l;m} = \frac{1}{\Delta x}(f_{l+1/2;m} - f_{l-1/2;m}) \quad \textbf{(12)}$$

$$\Delta^1_y f_{l;m} = \frac{1}{\Delta y}(f_{l;m+1/2} - f_{l;m-1/2}) \quad \textbf{(13)}$$

$$\nabla^2_h f_{l;m} = \frac{f_{l+1;m} - 2f_{l;m} + f_{l-1;m}}{\Delta x^2} + \frac{f_{l;m+1} - 2f_{l;m} + f_{l;m-1}}{\Delta y^2} \quad \textbf{(14)}$$

The terms

$$a^n_{i+1/2;j} \text{ and } b^n_{i;j+1/2}$$

are the approximations of

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial v} + w\frac{\partial u}{\partial z} \quad \text{and}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}$$

We have

$$a^n_{i+1/2;j} = u^n_{i+1/2;j}\Delta^0_x u^n_{i+1/2;j} + V^n_{i+1/2;j}\Delta^0_y u^n_{i+1/2;j} \quad \textbf{(15)}$$

$$b^n_{i;j+1/2} = U^n_{i;j+1/2}\Delta^0_x v^n_{i;j+1/2} + v^n_{i;j+1/2}\Delta^0_y v^n_{i;j+1/2} \quad \textbf{(16)}$$

where

$$U_{i;j+1/2} = \frac{1}{4}(u_{i+1/2;j} + u_{i+1/2;j+1} + u_{i-1/2;j+1} + u_{i-1/2;j}) \quad \textbf{(17)}$$

$$V_{i+1/2;j} = \frac{1}{4}(v_{i+1;j+1/2} + v_{i;j+1/2} + v_{i;j-1/2} + v_{i+1;j-1/2}) \quad \textbf{(18)}$$

$$\Delta^0_x f_{l;m} = \frac{1}{2\Delta x}(f_{l+1;m} - f_{l-1;m}) \quad \textbf{(19)}$$

$$\Delta^0_y f_{l;m} = \frac{1}{2\Delta y}(f_{l;m+1} - f_{l;m-1}) \quad \textbf{(20)}$$

All these approximations have second-order accuracy. That is, the error involved lies in O ($\Delta x^2 + \Delta y^2$). The algorithm to compute the solution to the Navier-Stokes equations then follows. For the known current state of the velocity vectors and pressures of the fluid flow field

$$\left\{u^n_{i+1/2;j},\ v^n_{i;j+1/2},\ \text{and}\ p^n_{i;j}\right\}$$

the next state

$$\left\{u^{n+1}_{i+1/2;j},\ v^{n+1}_{i;j+1/2},\ \text{and}\ p^{n+1}_{i;j}\right\}$$

after Δt time is calculated by Equations 9, 10, and 11. These equations use functions from Equations 12 to 20.

Numerical stability considerations

The stability of the numerical computation depends on many factors. Rather than present a lengthy mathematical analysis of stability conditions, we later present (in the section "Application examples") a number of choices of convergence parameters, each user adjustable even during the course of a simulation. In a simulation, you can adjust some parameters to the safe zone or dynamically update these parameters to achieve stability and other special effects.

For example, we can simulate different kinds of fluids by changing the Reynolds number and the fluid behaves differently. However, a higher Reynolds number in the calculation might result in numerical divergence, while a lower Reynolds number will result in numerical convergence. We can specify a very high Reynolds number to achieve some specific turbulent behavior but reduce the Reynolds number before the numerical calculation diverges. This way, we can use a simple numerical method to achieve behaviors not possible for a constant parameter. Although this is not an accurate fluid computation, it suffices for fluid animation for certain phenomena.

Here we simply point out that the numerical calculations tend to stabilize if we choose smaller *dt* and *Re*

and larger dx, dy, and ε values. The trade-off is that a smaller dt will result in a slower simulation; a smaller Re will result in quieter fluid behavior (tending to laminar); and a larger dx, dy, and ε will result in greater errors in the simulation.

3D fluid surface animation

After calculating the 2D fluid flow field's velocity vectors and pressures, we can draw a current frame of velocity field. For a given grid (i, j) in the flow field,

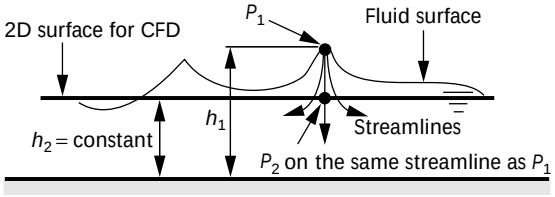
$$u_{i,j} = \frac{u_{i+1/2,j} + u_{i-1/2,j}}{2} \quad \textbf{(21)}$$

$$v_{i,j} = \frac{v_{i,j+1/2} + v_{i,j-1/2}}{2} \quad \textbf{(22)}$$

we can draw a velocity vector from (i, j) to $(i + u_{i,j}, j + v_{i,j})$. By raising the grid (i, j) in the third dimension to the scale of $p_{i,j}$, we can draw the fluid velocity field in a 3D surface. Using the Bernoulli equation—a basic fluid mechanics equation satisfied along two points on the same streamline—justifies the pressures to simulate the fluid surface height:[13]

$$\frac{V_1^2}{2} + gh_1 + \frac{p_1}{\rho} = \frac{V_2^2}{2} + gh_2 + \frac{p_2}{\rho} \quad \textbf{(23)}$$

where V_1 and V_2 are the velocities at point 1 (P_1) and point 2 (P_2) respectively, g is the gravity, h_1 and h_2 are the heights, p_1 and p_2 are the pressures, and ρ is the density of the fluid (see Figure 2). To simplify the analysis, we consider the velocities at the two points to be equal, that is, $V_1 = V_2$. We assume P_1 lies on the surface of the fluid, and P_2 lies on the 2D surface used to calculate the 2D Navier-Stokes equations as shown in Figure 2. We can see that the fluid's surface has a constant pressure



2 Relationships between pressure and height.

(that is, the pressure of the atmosphere, p_1 = constant) and the height of the 2D surface is constant (h_2 = constant). Therefore, for our purpose, we can simplify Equation 23 as follows:

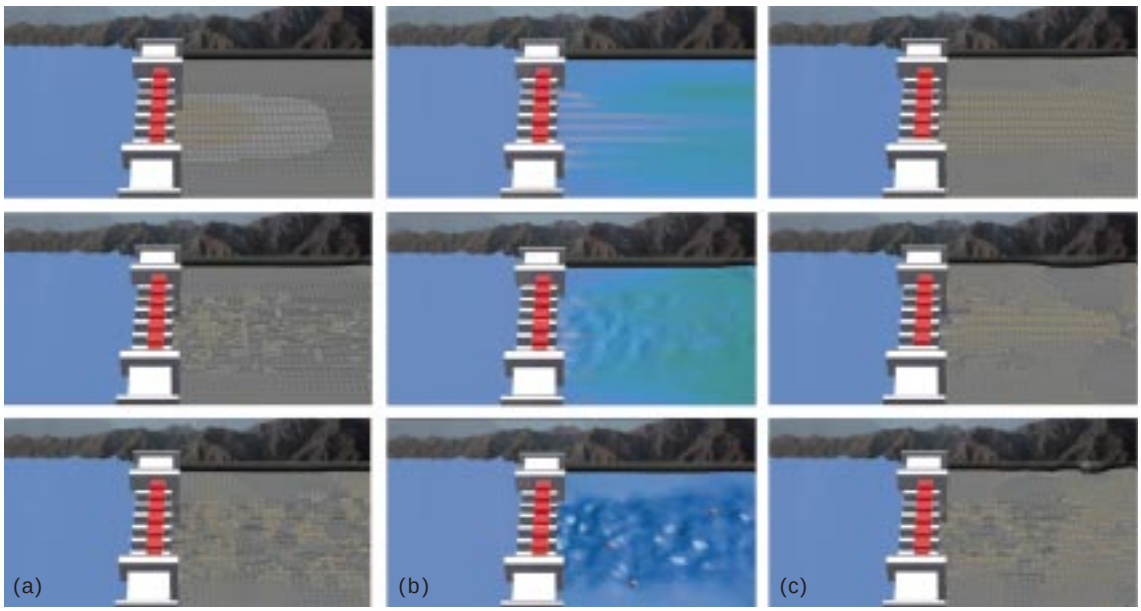
$$h_1 = \frac{p_2}{g\rho} + h_2 - \frac{P_1}{gp} \quad \textbf{(24)}$$

Since g, ρ, p_1, and h_2 are all constants, we can see that the height of h_1 is proportional to the pressure p_2. In particular, when $p_2 = p_1$, $h_1 = h_2$.

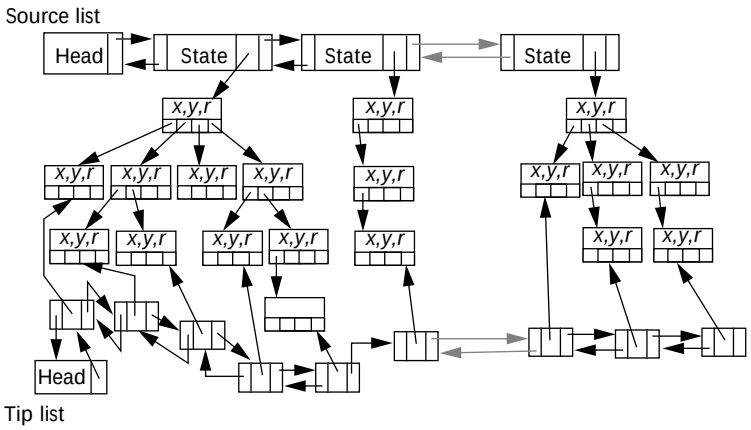
Therefore, as the real-time calculations and drawing progress, we can animate the velocity vectors of the points on the fluid's surface. For example, given a channel flow with a boundary condition (specified later in the section "Applications examples") for a dam on a river, we can have frames of the animation of the velocity field as in Figure 3a, where yellow indicates levels above zero, white indicates those equal to zero, and blue indicates those below zero. By shading and lighting the surface of the flow field, we obtain frames of the same simulation of the channel flow as shown in Figure 3b. So for a fluid flow, we have obtained not only the simulation of the fluid surface, but also the velocity field on the surface, thus providing the velocities of all visible points on the fluid. This information is important because it applies to simulating floating objects with the speed of the fluid.

Volume conservation and fluid flow

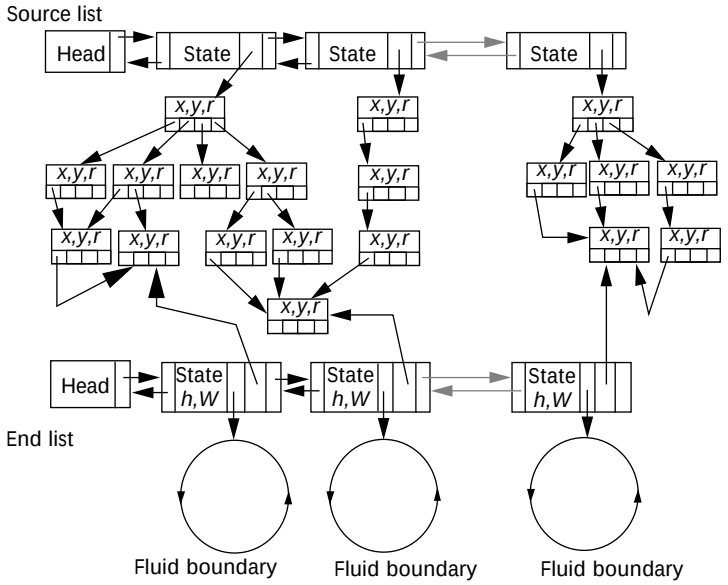
The pressure-height 3D surface behavior does not include the fluid volume and fluid flow boundary



3 Fluid flows with different *Re* numbers (*Re* = 1, *Re* = 8, *Re* = 300 from top to bottom, respectively) and boundary conditions. (a) Velocity vector field frames. (b) Same fluid flow frames with surface shaded. (c) Same fluid flow frames with the right boundary blocked.



4 Fluid source list structure.



5 Fluid end list structure.

manages the fluid sources, the tip list manages the fluid flow from sources to the destinations where these flows accumulate, and the end list manages the fluid accumulation. Fluid flow points on the terrain connect these structures. Elevation posts represent the terrain—each post corresponds to the location of a point. The connected fluid flow points form a path of fluid flow. Following a fluid source in the source list, we can step through the fluid flow points to the locations in the tip list where the fluid expands or to the locations in the end list where the fluid accumulates.

The source list contains fluid source pointers and state information. A fluid source pointer provides the location (x, y) and the rate of flow (r) of a fluid source on a terrain (see Figure 4). Fluids generate from sources in the source list.

The tip list, a temporary list, connects all points that are currently at the tips of the fluid flow (Figure 4). It lets us calculate the fluid's leading edge until it arrives at the endpoints where it will accumulate. Flow rates are calculated and carried along the fluid flow with the tips. Notice that those fluid points between a fluid source and a fluid tip are neither in the source list nor in the tip list. They are points along the path of fluid flow from the source to the tip. When a fluid tip expands to a point where the fluid ceases to flow because it is lower than all its surrounding points, we reclassify the tip as an endpoint where fluid accumulates. The endpoint joins the end list, which contains locations where fluid accumulates (see Figure 5).

The end list contains fluid states, fluid surface heights (h), fluid surface areas (W), pointers to fluid boundaries, and pointers to fluid endpoints (x, y) and flow rates (r). Fluid state information tells whether the current fluid volume is growing. Fluid surface height lets us draw the surface at its corresponding height. Flow rate and surface area together determine the fluid height changes for any particular point in time. The fluid boundary, or perimeter, links to a list of the fluid surface's boundary points (see Figure 6). The structure is convenient for DIS because we only need to send the fluid heights and boundaries across the network to describe the same fluid areas and depths on different simulators. Note, the source list, the tip list, and the end list coexist in a fluid flow and terrain environment, although to simplify the explanation we did not draw them together.

changes, making it difficult to simulate flows that depend on the simulated environment. In many applications, such as simulations involving a dynamic terrain, fluid flow interacts with the environment. For example, a bulldozer could break a dam, causing water to flow from the breach and over soil that changes continually. Therefore, fluid conservation and the interaction of the flowing fluid with its environment are important issues to address.

We now introduce a method that calculates the fluid flow and volume conservation necessary for complementing the physical fluid surface behavior simulation. Hence, our general fluid model allows fluid to generate anywhere on a terrain surface, and the flow follows the environment's 3D contours. It allows changing boundaries and accounts for the fluid's depth. With these capabilities, the fluid model integrates into the dynamic terrain within a DIS. Our approach has many advantages—it does not confine fluid boundaries, it conserves fluid volume, and it synchronizes fluid behaviors in the distributed interactive simulation.

Data structure

Three main structures exist in this generalized fluid model: the *source list*, *tip list*, and *end list*. The source list

Fluid flow volume conservation calculation

Fluid flow calculation includes adding fluid at source

points, expanding fluid at tip points, and accumulating fluid at endpoints.

Update a source. When a user specifies a source and a rate of flow, a new fluid source node is created in the source list and in the tip list. Initially, the fluid source state is set to active. When a fluid source is stopped or removed, that is, when no more fluid flows from the current fluid source, we set the state of the source to inactive (state equals 0). Then, we iteratively trace the fluid flow from the inactive source points to the endpoints, removing all the fluid flow points along the path. Note that if two sources flow together at some intermediate point, our data structure maintains two separate entries, one associated with each source. In this process, we remove only the entry associated with the inactive source. Thus we can have many user input fluid sources generating fluids and eliminate many dried fluid sources.

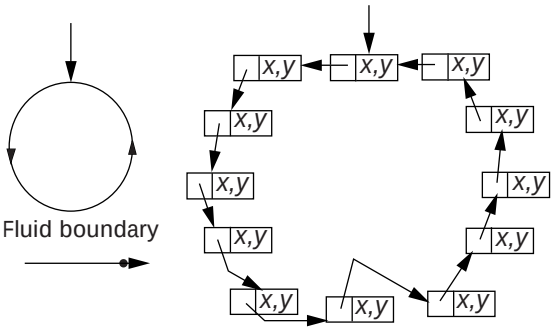
Flow at the tips. Fluid tips in the tip list are processed by expanding to new locations, that is, new tips. When this occurs, the current tips delink from the tip list and become inner points of the fluid flow. Every fluid tip is a temporary structure, used only once. We use it to find the next positions to which the fluid flows from its current position according to the environment (that is, the shape of the terrain, which, in this case, consists of elevation posts). A tip searches its surrounding points (namely, elevation posts), and the next positions are those points lower than the current point. Dividing the fluid flow rate by the number of the next positions yields the flow rate passed on to these next positions. If this position is a local lowest point on the terrain, then no next position exists. In this case, fluid will start to accumulate, and this point joins the end list. Otherwise, new fluid tips are generated as children of the current tip and added to the current tip list. The current fluid tip is removed from the tip list, and becomes an inner point of the fluid flow.

Accumulate at the ends. The points in the end list are points where fluid accumulates. Processing the end list updates the fluid surface heights, surface areas, surface boundaries, and mergers of fluid boundaries.

We have said that the fluid flow rate travels with fluid flow points to the ends. Given the fluid flow rate r at an endpoint, the fluid volume (V) to add to the current location at an iteration is

$$V = r \bullet \Delta t \qquad \textbf{(25)}$$

If the elevation of the environment around the current fluid surface approximates the height of the current fluid surface, the fluid will expand to its surrounding area at the rate corresponding to the volume V. Assuming that without raising the fluid surface, each elevation post (point) takes a unit of fluid volume and the surrounding area has P points to receive a portion of V, then the fluid will expand to P^* ($P^* \le V$) points. The fluid surface area W will update as $W = W + P^*$. The neighboring points of the current fluid area are randomly chosen. If $P < V$, then $P^* = P$, else $P^* = V \le P$. If the surrounding points of current fluid surface lie above the fluid surface, the fluid surface will rise according to the added volume and surface area. The surface height rise Δh is calculated by



6 Fluid boundary structure.

$$\Delta h = \frac{V - P^*}{W} \qquad \textbf{(26)}$$

When the fluid has a boundary point lower than the current fluid surface height by a preset threshold, we reclassify the current boundary point as a new fluid source by adding it to the source list and the tip list with the current flow rate. Then the current fluid stops expanding.

When two fluid surfaces start to intersect with each other, we merge these two surfaces into one fluid surface. You can see that when two surfaces start to intersect, their surface heights are close to each other. Rather than calculating boundary intersections, we use a marker method to detect the intersection. Calculating intersections between fluid surfaces proves expensive. When fluid expands to a post, it marks the current post. When fluid expands to an already marked post, it detects an intersection, and the current post is used to find the intersected surface.

The calculations above produce the fluid's footprint for dynamically changing terrain. Our 2D Navier-Stokes system works within this footprint to achieve a realistic looking simulation. The above data structures can be used in a DIS. We can simulate a fluid on one host, sending data in the end list to networked simulators. The end list provides the fluid height and surface boundaries. This information is sufficient to reproduce similar fluid surface appearances on networked simulators. Therefore, only the fluid end lists on network simulators need to be updated, keeping network traffic to a minimum. We will further explain the method of synchronizing physically based modeling in a DIS later, in the section "Synchronization in DIS."

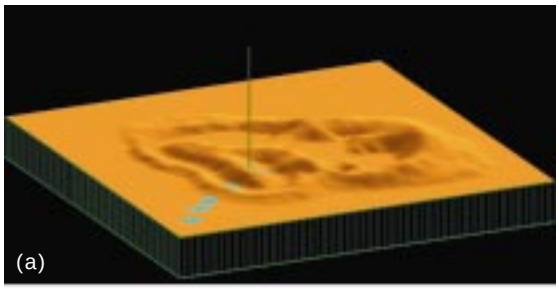
Application examples

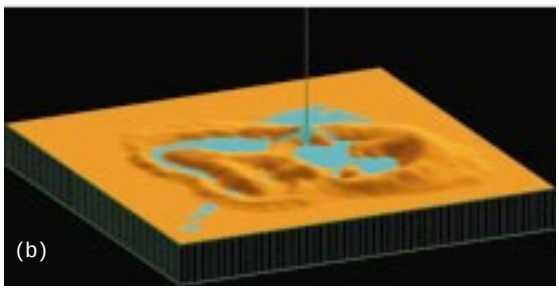
In all the examples given here, unless otherwise specified the initial values of the Reynolds number $Re = 300$, grid size in the x and y directions is 1 meter, time slice $\Delta t = 0.001$ second, penalty parameter $\varepsilon = 0.005$, and fluid field size is $X \times Y$ where $60 \le X \le 120$ and $60 \le Y \le 120$.

Fixed or movable boundary conditions

A flow is internal if the flow field resides inside a channel, pipe, or any other external boundaries. It is external if the flow field lies outside a tower, bridge, or any other internal boundaries. A fluid flow can be both internal

7 Movable boundary conditions of a (a) slow sail boat and (b) fast speed boat.





8 Changeable boundaries: (a) a user movable fluid source and (b) fluid emanating from the source.

and external if it has both internal and external boundaries. The fluid boundary changes determine the fluid area, but the fluid boundary condition changes determine the fluid behavior. Therefore, the boundary conditions prove important in achieving the required fluid flow simulation. The boundary conditions can be fixed or moving—they are inputs to the computation.

For example, consider the initial conditions of the fluid as $u_{i+1/2;j} = 5$, $v_{i;j+1/2} = 0$, and $p_{i;j} = 0$ for all $i = 0, 1,\ldots, X-1$ and $j = 0, 1, \ldots, Y-1$. If we add a fixed external boundary condition representing the edges and boundaries of a channel as $u_{i+1/2;0} = 5$, $v_{i:1/2} = 0$, $u_{i+1/2;Y-1} = 5$, $v_{i;Y-3/2} = 0$, $u_{1/2;j} = 5 + 2*(Y-j)*j*(j \bmod 3$ and $j > Y/3$ and $j < 2*Y/3)/Y$, $v_{0;j+1/2} = 0$, $u_{X-3/2;j} = (u_{X-3/2;j} + u_{X-1/2;j})/2$, and $v_{X-1;j+1/2} = (v_{X-1;j+1/2} + v_{X-2;j+1/2})/2$ for all $i = 0, 1,\ldots, X-1$, and $j = 0, 1, \ldots, Y-1$, we get frames of the simulation shown in Figures 3a and 3b (dam flow). If we change the above boundary condition to $u_{X-3/2;j} = (u_{X-3/2;j} + u_{X-1/2;j})/4$, and $v_{X-1;j+1/2} = (v_{X-1;j+1/2} + v_{X-2;j+1/2})/4$, we get frames of the simulation shown in Figure 3c (dam flow with vortices).

We can iteratively modify the boundary conditions to change the fluid behaviors. Therefore, depending on the applications, the boundary conditions can be fixed, movable, not modified, or modified at each step. For example, by specifying a small internal boundary area inside the fluid flow field and moving the area and its boundary conditions in the fluid field, we can achieve the behavior of a boat moving inside the fluid. Providing the initial and boundary conditions of a channel flow as $u_{i+1/2;j} = 30 + (Y-j)*j/Y$, $v_{i;j+1/2} = 0$, and $p_{i;j} = 0$, $u_{i+1/2;0} = v_{i;1/2} = u_{i+1/2;Y-1} = v_{i;Y-3/2} = 0$, $u_{1/2;j} = (u_{1/2;j} + u_{3/2;j})/2$, $v_{0;j+1/2} = 0$, $u_{X-3/2;j} = (u_{X-3/2;j} + u_{X-1/2;j})/2$, and $v_{X-1;j+1/2} = (v_{X-1;j+1/2} + v_{X-2;j+1/2})/2$ for all $i = 0, 1, \ldots, X-1$ and $j = 0, 1, \ldots, Y-1$, then such a moving boat can be represented as $u_{i+1/2;j}$ = WakeSpeed, $u_{i-1/2;j+1}$ = WakeSpeed, $u_{i-1/2;j-1}$ = WakeSpeed where $(i; j)$ is the current location of the moving boat, which in our example repeatedly changes along the path from $(0; Y/2)$, $(1; Y/2)$, . . ., to $(X-1; Y/2)$ and where WakeSpeed equals the current boat speed. These conditions result in the animation frames shown in Figure 7. These conditions amount to setting the fluid velocities in the internal boundary area in correspondence with the movement of the boat. If the internal boundary area moves slowly, the wake behind the area is very small, producing the effect of a sail boat in the fluid as shown in Figure 7a (WakeSpeed = 50). Compare this with the result in Figure 7b, in which the area moves fast, causing a large and turbulent wake simulating the effect of a speed boat (WakeSpeed = 70). We can put several internal movable boundary conditions together into the fluid to achieve the animation of different boats in the same fluid. Later, we'll discuss the blending of fluids of different colors and the drifting of objects shown in Figure 7b.

Fixed or changeable boundaries

The fluid boundary can also be fixed or changeable. Note that the change of the fluid boundaries differs from the change of fluid boundary conditions. The fluid boundary changes determine the fluid footprint on a terrain; the fluid boundary condition changes determine the fluid surface behavior. The channel flow in Figure 3 is an example of a fixed boundary. The fluid flow field has a fixed given area, while the general flow on a terrain as shown in Figure 8 has changeable external boundaries. Users can control the movement of the green straw in Figure 8, which specifies the location of the current fluid source.

Viscous or inviscid fluid flow

A fluid's Reynolds number is inversely proportional to its viscosity. Thus we can simulate different fluid flows by changing the Reynolds number. If the Reynolds number is small, the fluid is viscous and the flow laminar. If the Reynolds number is large, the fluid tends to be inviscid and the fluid more turbulent. For example, Figure 3 depicts the same fluid flow situation with $Re = 1$, $Re = 10$, and $Re = 300$ from top to bottom, respectively.

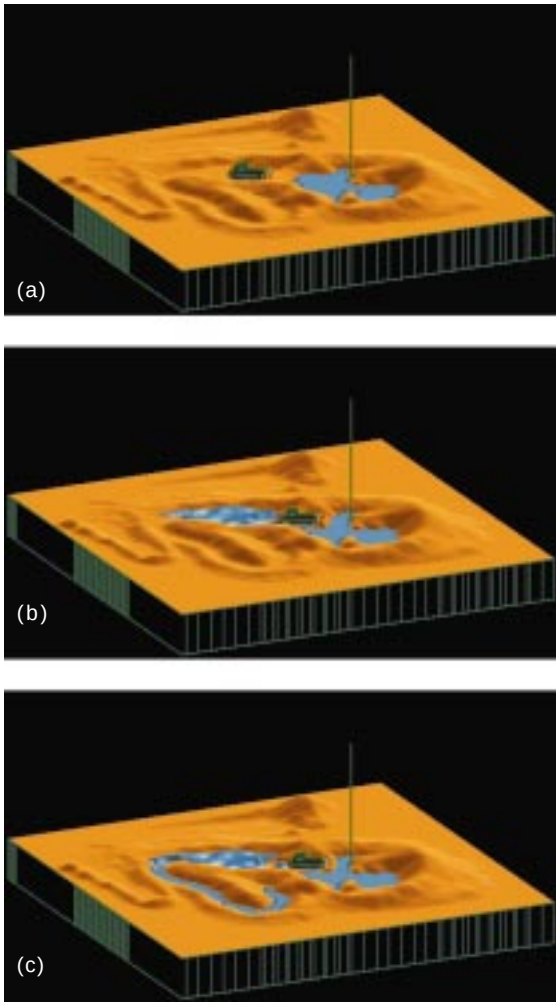
Floating objects and color elements in the velocity field

The fluid velocity field provides a convenient tool to simulate objects floating in the fluid. Floating objects will flow in the fluid field with the velocities at their current locations. When an object reaches a new position, its velocity changes to the velocity at that position.

A simple method mixes fluids of different colors, producing a realistic look of color blending, as follows. We consider fluids of different colors to contain elements of different colors that can mix their colors when they flow to the same grid position in the flow field. Each element updates its color and the grid position's color to which it travels by averaging the grid color and all elements at the same grid position. The color elements float in the fluid with the velocities at their current locations. Figures 3b and 7b display results from this approach.

Fluid in a dynamic virtual environment

We use a soil slippage model to show how the boundaries change and how to conserve volume in a dynamic virtual environment. Figure 9a shows the fluid flow in a dynamic terrain in which a bulldozer will break the dam. Figures 9b and 9c show the fluid flow after this happens. A user can place fluid sources at any point on the terrain. The fluid will emanate from these points and follow the environment. We calculated and animated the soil slippage and fluid flow in real time.



9 (a) A bulldozer moving toward a dam, (b) the bulldozer just after breaking the dam, and (c) fluid flow after the bulldozer breaks the dam.

Synchronization in DIS

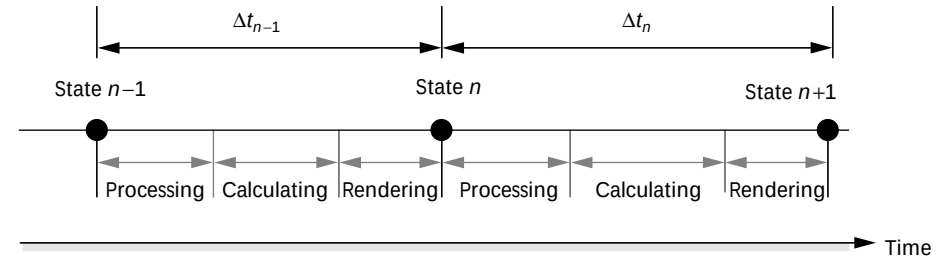
An entity is an object such as a bulldozer, boat, piece of changing terrain, or even a piece of fluid surface simulated in a DIS. Each entity has a local variable (*lastTime*) that records the last time this entity updated its state. When an entity begins to update its state, it can read the simulator's clock to get the current time (*currentTime*) and subtract the *lastTime* to determine the period between the current time and the last time the state was updated. This period equals the time slice passed; its value, together with the old state, determines the new state. At the same time, *lastTime* updates to *currentTime*.

Each entity proceeds at its own pace—synchronized by our clock's uniform time scale. The whole system is synchronized in the sense that, at a certain instance, all entities have advanced approximately the same amount of time. For a simulator with a fast simulation cycle, the time slice tends to be small and the fidelity high; for a simulator with a slow simulation cycle, the time slice tends to be large and the fidelity low. This synchronization ensures that all entities use the same time scale.

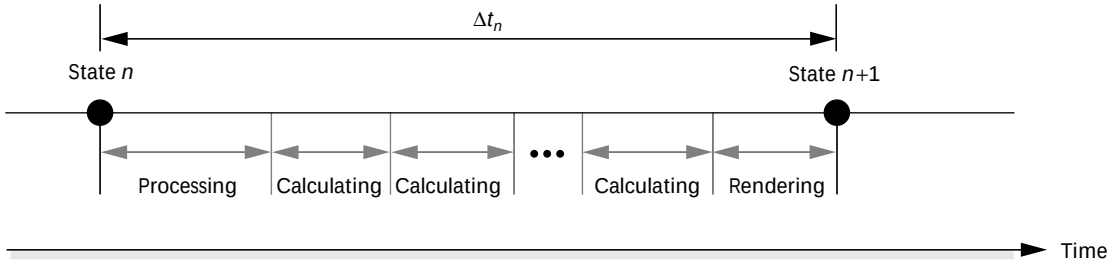
When an entity updates its corresponding entities on a different host, it sends its current state to them. They update their states by the received state and the network delay. When an entity receives a message, it computes the network delay between the time the message was sent and when it was received. When the sender sends a message including the current time information of the sender's clock, the receiver decides the network delay of the received message by looking at its own clock and the time information sent with the received message. For example, if we know that the time difference between simulator 1 and simulator 2 is exactly three seconds (simulator 1 is three seconds earlier than simulator 2), then if simulator 2 sends a message to simulator 1 at 12:00:04 p.m. (on simulator 2's clock) and simulator 1 receives the message at 12:00:03 p.m. (of simulator 1's clock), we know the network delay equals two seconds.

Two major advantages of this synchronization mechanism exist. One is that it needs very little network information. Synchronization does not result from sending messages across the network. The other advantage is that you can predict activities and record time elapses of events. For example, if we know the speed of a vehicle, then we can predict its position ahead of the simulation at an instance. Thus, the simulated environments provide a better perception of real-world time.

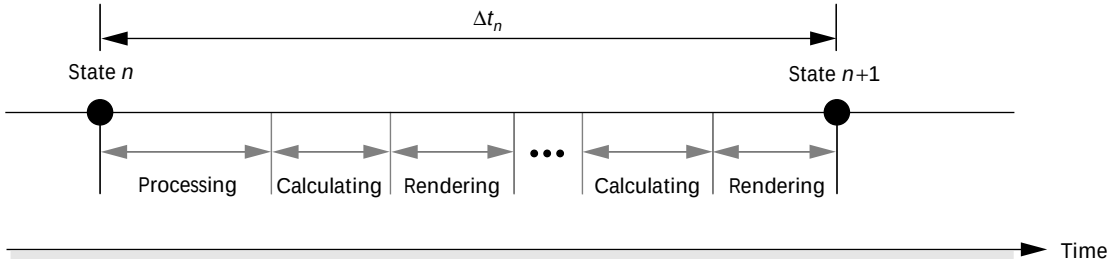
Existing efforts in physically based modeling have used different approaches and physical laws (such as Newton's second laws, energy equations, Lagrange equations, and Navier-Stokes equations) to achieve modeling and simulation. All these physical laws are functions of time. The numerical simulations of these physical laws include calculating current state from last state and the time passed between last and current states. In a standalone simulation, Δt_n is often a chosen constant allowing the same numerical stability for all n values. In a real-time DIS, three major activities exist between states



10 Activities between simulation states.



11 Physical models needing a small time slice.



12 Physical models needing a small time slice but evolving fast.

of simulation as shown in Figure 10: processing (network) messages and other information, calculating next states of the (physical) entities, and rendering the calculated entities graphically. To achieve real-time simulation, all the activities cannot be slower than certain thresholds.

The network synchronization method introduced earlier synchronizes the network activities and minimizes the network traffic in DIS. However, it could happen that the varying time slices between states are so big that the numerical computation for physically based modeling diverges. As we know in physically based modeling and simulation, the time slice must remain smaller than a certain criterion to retain numerical stability and to limit the numerical offset error. For example, the numerical calculations of the Navier-Stokes equations need small time slices to retain numerical stability.

Here we provide a time differential solution for this problem. Instead of using a big time interval Δt_n, we can differentiate it into smaller time slices satisfying the numerical calculation requirements of the physical equations. For example, if we know that Δt satisfies these requirements, we still use it to calculate the physical equations regardless of the varying time slice between simulation states, yet we can also synchronize all the physical models' activities in the network. When we get Δt_n larger than Δt, we can divide Δt_n into a number of Δt values and calculate the activities of the physical phenomena $\lfloor \Delta t_n / \Delta t \rfloor$ times. The residue of the time division can be added to the next simulation period. If Δt_n proves smaller than Δt, we can consider it as a residue time and add it to the next simulation period.

Figure 11 illustrates this process. Notice the difference between the time slice used to calculate the state changes and the time slice between state changes.

In cases where the time slice is small but the physical phenomena evolve so fast that the animation appears jumpy (temporally aliased) after a series of calculations, we can render every frame after each numerical calculation (see Figure 12). This situation occurs when we simulate fast particle movements in a DIS.

Discussion and conclusion

Although our fluid model examples are simple and limited, they can serve as a testbed to simulate many more fluid phenomena by changing the boundary conditions, adding interactions between fluids and objects, mixing fluids of different properties, and adding fluids at different locations in the dynamic virtual environment.

However, our fluid model does have some limitations. Specifically, the 2D solution with the addition of pressures is not physically equivalent to the solution of 3D Navier-Stokes equations, which precisely describes the fluid behavior. For example, the boat wakes don't show differential wave curvature. The images look rigid compared to real water. Our model cannot be used for any accurate engineering purpose. However, it is useful in applications where the fluid effects are more important than the accuracies of the computational results. In a simulated virtual environment for real-time pilot train-

ing, for example, the real-time fluid behaviors will greatly enhance the realities of the training environment. The boat wakes can provide pilots with useful visual cues even without accurate results.

Some other problems remain. For example, the simulation may not stay stable after a long period of time if the Reynolds number is too high. Also, our fluid model does not account for fluid jump and fluid refraction. In subsequent work we plan to model the interactions between floating objects and fluid and address some of the problems.

We are currently working on improving our fluid model and using it for interactive distance learning and a networked training environment. ■

Acknowledgments

We are grateful to Xin Li for providing suggestions and for his dynamic terrain soil slippage model code; Michelle Sartor, Art Cortes, and Curt Lisle for providing comments and suggestions; the people at the Visual Systems Lab (VSL) for their support; and the VSL of the Institute for Simulation and Training (IST) for providing the facilities for our initial research.

References

1. Institute for Simulation and Training, *The DIS Vision: A Map to the Future of Distributed Simulation*, Version 1, Univ. of Central Florida, Orlando, Fla., May 1994.
2. A. Fournier and W.T. Reeves, "A Simple Model of Ocean Waves," *Comp. Graphics,* Vol. 20, No. 4, Aug. 1986, pp. 75-84.
3. D.R. Peachey, "Modeling Waves and Surf," *Comp. Graphics,* Vol. 20, No. 4, Aug. 1986, pp. 65-74.
4. P.Y. T'so and B.A. Barsky, "Modeling and Rendering Waves: Wave-Tracing Using Beta-Splines and Reflective and Refractive Texture Mapping," *ACM Trans. on Graphics*, Vol. 6, No. 3, July 1987, pp. 191-214.
5. G. Miller and A. Pearce, "Globular Dynamics: A Connected Particle System for Animating Viscous Fluids," *Computers and Graphics,* Vol. 13, No. 3, 1989, pp. 305-309.
6. D. Terzopoulos, J. Platt, and K. Fleischer, "Heating and Melting Deformable Models (From Goop to Glop)," *Proc. Graphics Interface,* Canadian Information Processing Society, Toronto, June 1989, pp. 219-226.
7. M. Kass and G. Miller, "Rapid, Stable Fluid Dynamics for Computer Graphics," *Computer Graphics,* Vol. 24, No. 4, Aug. 1990, pp. 49-55.
8. M.E. Goss, "A Real-time Particle System for Display of Ship Wakes," *IEEE Computer Graphics and Applications*, Vol. 10, No. 3, May 1990, pp. 30-35.
9. J. Wejchert and D. Haumann, "Animation Aerodynamics," *Computer Graphics,* Vol. 25, No. 4, July 1991, pp. 19-22.
10. J. Stam and E. Fiume, "Turbulent Wind Fields for Gaseous Phenomena," *Proc. Computer Graphics*, Addison Wesley, New York, Aug. 1993, pp. 369-376.
11. J.X. Chen and N.V. Lobo, "Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations," *Graphical Models and Image Processing,* Vol. 57, No. 2, Mar. 1995, pp. 107-116.
12. R. Peyret and T.D. Taylor, *Computational Methods for Fluid Flow*, Springer-Verlag, New York, 1985.
13. W.R. Fox and A.T. McDonald, *Introduction to Fluid Mechanics,* 4th edition, John Wiley and Sons, New York, 1992.
14. R. Temam, *Bulletin de la Societé Mathématique de France*, Gauthier-Villars, Paris, 1968, pp. 115-152.

Jim X. Chen *is an assistant professor of computer science at George Mason University in Fairfax, Virginia. His research interests include physically based modeling, real-time simulation, and scientific visualization in graphics. He received a PhD in computer science from the University of Central Florida in 1995, and MS and BS degrees in computer science from Southwest Jiaotong University, China, in 1986 and 1983, respectively. He is a member of the IEEE Computer Society and ACM.*

Niels da Vitoria Lobo *is an assistant professor in the Department of Computer Science at the University of Central Florida in Orlando. His research interests are in vision and physical modeling for graphics. He received a BS at Dalhousie University in Canada in 1982, and an MS and PhD at the University of Toronto in 1985 and 1992, respectively. He is a member of the IEEE Computer Society.*

Charles E. Hughes *is a professor of computer science at the University of Central Florida in Orlando. His research interests are in distributed interactive simulation, constraint logic programming, parallel processing, and object-oriented paradigms. He received a PhD and MS in computer science from Pennsylvania State University, and a BA in mathematics from Northeastern University. He is a member of the IEEE, IEEE Computer Society, and ACM.*

J. Michael Moshell *is a professor of computer science at the University of Central Florida in Orlando. He founded the Visual Systems Laboratory of the Institute for Simulation and Training at the University of Central Florida and currently serves as its chief scientist. His research interests are in applying simulation and virtual environments to training and education. He received a BS in physics from Georgia Institute of Technology in 1968 and a PhD in computer science from Ohio State University in 1975. He is senior editor of* Presence. *He is a member of the IEEE and ACM.*

Contact Chen at George Mason University, Department of Computer Science, MS 4A5, Fairfax, Va., 22030-4444, e-mail jchen@cs.gmu.edu.