Realistic Animation of Liquids

Nick Foster and Dimitri Metaxas

Center for Human Modeling and Simulation,

University of Pennsylvania, Philadelphia, PA 19104

June 3, 1996

Abstract

We present a comprehensive methodology for realistically animating liquid phenomena. Our approach unifies existing computer graphics techniques for simulating fluids and extends them by incorporating more complex behavior. It is based on the Navier-Stokes equations which couple momentum and mass conservation to completely describe fluid motion. Our starting point is an environment containing an arbitrary distribution of fluid, and submerged or semi-submerged obstacles. Velocity and pressure are defined everywhere within this environment, and updated using a set of finite difference expressions. The resulting vector and scalar fields are used to drive a height field equation representing the liquid surface. The nature of the coupling between obstacles in the environment and free variables allows for the simulation of a wide range of effects that were not possible with previous computer-graphics fluid models. Wave effects such as reflection, refraction and diffraction, as well as rotational effects such as eddies, vorticity, and splashing are a natural consequence of solving the system. In addition, the Lagrange equations of motion are used to place buoyant dynamic objects into a scene, and track the position of spray and foam during the animation process. Typical disadvantages to dynamic simulations such as poor scalability and lack of control are addressed by assuming that stationary obstacles align with grid cells during the finite difference discretization, and by appending terms to the Navier-Stokes equations to include forcing functions. Free surfaces in our system are represented as either a collection of massless particles in 2D, or a height field which is suitable for many of the water rendering algorithms presented by researchers in recent years.

Additional Keywords and Phrases: Dynamic Fluid Simulations, Free-surface Flow, Navier-Stokes Equations, Physics-Based Modeling.

1 Introduction

Some of the most breathtaking animations in recent years have been generated by modeling the interaction between light and water. Effects such as caustic shading, reflection, refraction, and internal scattering have been addressed in some detail, with realistic results [12, 15, 17]. One characteristic of this work however, has been that the motion of the water surface is approximated by a non physics-based function. Suggested methods have included parametric functions [4] and sinusoidal phase functions [8, 13]. Two exceptions to this are the papers by Kass and Miller, and Chen and Lobo. Kass and Miller use a fast approximation to the two-dimensional shallow water equations to simulate surface waves in water of varying depth [7]. Their model allows for the reflection and refraction of waves, and takes account of mass transport, but it does not address the full range of three-dimensional motion found in a liquid. Such motion includes rotational and pressure based effects responsible for the much of a fluid's characteristic behavior. They also cannot easily incorporate dynamic objects or buoyant effects into the model, because the velocity of the fluid is known only on the surface, and internal pressure is not calculated at all. Chen and Lobo go further towards a physics-based fluid methodology by solving a simplified form of the Navier-Stokes equations in two dimensions [1]. However, they assume that the fluid has zero depth, and calculate the elevation of the surface solely from the instantaneous pressure. This allows them to perform some interaction between moving objects and the flow field, but restricts the class of problems that can be solved using the method. Notably, obstacle geometry must be two-dimensional, and although the surface height is varied for animation, they treat the fluid as being completely flat during the calculation. Therefore, convective wave effects, mass transport, and submerged obstacles are not covered by their technique.

Comprehensive models of fluid motion do exist, and there are a variety of tools for solving them in the field of Computational Fluid Dynamics (CFD). These methods generally involve direct simulation techniques to get accurate fluid motion. Unfortunately, in any direct simulation technique the temporal resolution is strongly coupled to the spatial resolution. Thus, if the spatial resolution doubles, the temporal resolution must also be doubled so that the solution does not move more than one spatial sample per time step. This gives running times proportional to the fourth power of the resolution, so most of these techniques will scale poorly. Furthermore, an animator needs a fairly clear understanding of the system of equations being solved so that he or she can set initial and boundary conditions to get the desired results. An ideal fluid simulator for graphics applications would apply the correct conditions automatically based on the underlying geometry. CFD methods also resist external control, making it difficult to force a particular motion from a fluid, unless it is a natural consequence of the system. These restrictions are an inherent part of the fluid modeling problem. The question arises whether it is

possible to accurately model realistic fluid motion while keeping within acceptable efficiency bounds for Computer Graphics.

In this paper we present a solution to the Navier-Stokes equations for modeling liquid motion, that satisfies many of an animator's needs. Realism is provided through a finite difference approximation to the incompressible Navier-Stokes equations. This gives rise to a complete pressure and velocity profile of the simulated environment. This profile is then used to determine the behavior of free surfaces, and is loosely coupled to the Lagrange equations of motion to include buoyant rigid objects into a scene. The range of behaviors accounted for include wave effects such as refraction, reflection and diffraction, together with rotational motion such as eddies and vorticity. Furthermore, velocity and pressure are strongly coupled within the model. This means that even the simplest animation exhibits subtle realistic behavior not available using previous computer-graphics fluid models.

Usability has also been a strong motivation for this paper. The Navier-Stokes equations are solved over a coarse, rectangular mesh containing an arbitrary distribution of submerged or semi-submerged obstacles. Boundary conditions for the mesh are generated automatically by constraining the free variables at an obstacle-fluid or air-fluid boundary. This low resolution calculation together with homogeneous boundary conditions leads to a relatively efficient determination of fluid velocity and internal pressure. Detail is achieved by using the velocity field to concentrate attention on regions of interest, i.e., the fluid surface. The surface is represented as either a chain of massless marker particles, or a height field. The markers are carried around the mesh by convection, and can have arbitrary connectivity, accounting for multiple colliding surfaces in a scene.

Consideration is also given to controlling the overall behavior of the fluid. Liquid sources or sinks (known as inflow and outflow boundaries) can be included anywhere in the environment. They allow liquid to flow (or be forced) into a scene, or flow out at a natural rate. A time dependent pressure field may also be applied to the fluid surface. Thus, the effects of a strong wind can be simulated and initial waves be driven realistically. The output from the system is a polygonal surface or height field, both of which can be rendered using many of the techniques presented by researchers in recent years [4, 7, 12, 13, 17].

This paper is organised as follows: We begin by describing the Navier-Stokes equations, which can be used to accurately determine the motion of a viscous liquid in three dimensions. We show how these equations can be solved to give the complete pressure and velocity profile of a flow, and that if care is taken during discretization of the scene, this can be done efficiently and automatically for an arbitrary environment. Section 4 then focuses attention on the liquid free surface. The dynamic position of multiple surfaces can be delineated without restriction by the convection of massless particles. For the special but common case of a flow without overturning, we derive a height field equation that couples

velocity and surface elevation to the Navier-Stokes equations to generate a mesh suitable for spline-based rendering. A method for loosely coupling the Lagrange equations of motion to the flow pressure profile is described in Section 5. This method is used to include bouyant rigid objects into a scene. The complete algorithm for our technique is given in Section 6. Following this, (Section 7) two methods for controlling the fluid motion by constraining velocity and pressure boundary conditions are presented. Examples are given to simulate different speeds of flow and wind driven waves. The paper is concluded with a description of several example animations that have been made using our system, together with a discussion of the procedure followed in each case.

2 Navier-Stokes Equations

The motion of a fluid at any point within a flow is completely described by a set of non-linear equations known as the momentum or Navier-Stokes equations. In three dimensions, for an incompressible fluid such as water, these equations can be written as

$$\begin{aligned}
\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} &= -\frac{\partial p}{\partial x} + g_x + \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) \\
\frac{\partial v}{\partial t} + \frac{\partial vu}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} &= -\frac{\partial p}{\partial y} + g_y + \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) \\
\frac{\partial w}{\partial t} + \frac{\partial wu}{\partial x} + \frac{\partial wv}{\partial y} + \frac{\partial w^2}{\partial z} &= -\frac{\partial p}{\partial z} + g_z + \nu(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}),
\end{aligned} \tag{1}$$

where u, v, w are velocities in the x, y, z directions respectively, p is the local pressure, g, gravity, and ν is the kinematic viscosity of the fluid. Although they may seem daunting at first sight, these equations have very humble origins. They are derived from Newton's Second Law which states that momentum is always conserved. The Navier-Stokes equations simply account for all momentum exchange possibilities within a fluid. Specifically, the terms on the left hand side of the equations account for changes in velocity due to local fluid acceleration and convection. The right hand terms take account of acceleration due to the force of gravity (or any body force g), acceleration due to the local pressure gradient, ∇p, and drag due to the kinematic viscosity, ν, or thickness of the fluid. Together with appropriate boundary conditions and the constraint that not only momentum, but also mass should be conserved (see Section 3.1) , the Navier-Stokes equations can be used to accurately simulate fluid phenomena.

3 Solving the Navier-Stokes equations

Despite the complexity of a system of differential equations such as (1), it is possible to solve it in an intuitive way, using standard analysis tools [3]. The first step is to discretize both the equations and the environment that we want to model. There are a number of ways to do this, but it is important to keep four things in mind:

- In a typical graphics application involving liquids, there are likely to be numerous boundaries between the liquid and other objects, and between the liquid and the surrounding medium. Computation cost can be minimized if such interfaces are homogeneously incorporated into the model instead of being treated as special cases.
- Generality is everything. Users of the system need to be able to specify environment geometry quickly, and without referring to the underlying equations for the correct boundary conditions.
- It must be possible to apply some external control to the system so that the animator can accurately specify how the liquid will behave.
- The range of motion that can be animated using the technique should include the set of effects available with existing computer-graphics methods, and extend it by adding new, interesting, and useful behavior.

With some thought, a good discretization that provides a solution to the first of these constraints also provides solutions to the other three. In the following sections we present a numerical solution to the Navier-Stokes equations. The technique combines a low resolution 3D calculation to determine velocity and pressure fields within the liquid, with a height field equation that is used to precisely track the position of a free surface. At all times during the computation, boundary conditions due to solid obstacles and the fluid surface are homogeneous, and their application is transparent to the user.

3.1 Discretization

We solve (1) across the entire environment. Solid obstacles and the atmosphere are treated as fluid, but with special properties that remain constrained throughout the calculation. The computation domain is first divided into a fixed rectangular grid aligned with a Cartesian coordinate system. The u, v, and w velocities are defined at the centers of each face of a cell and referenced locally (see Figure 1) while pressure, p , is defined in the center of each cell. Note from the figure that $v_{i,j-1/2,k} \equiv v_{i,(j-1)+1/2,k}$. At the start of the calculation the contents of each cell are determined. A cell may either contain a *Solid*

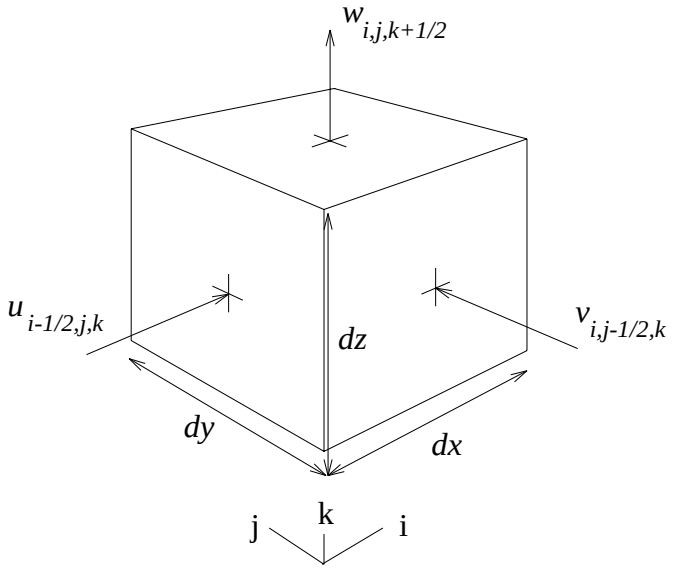


Figure 1: Location of staggered velocity components on a typical cell (i, j, k).

obstacle, be *Full* of fluid, be a *Surface* cell on the boundary between the liquid and surrounding medium, or be *Empty*. In all four cases, the velocity and pressure fields are defined everywhere.

This discretization leads to an explicit finite difference approximation of (1) in the form [6]

$$\begin{aligned}\tilde{u}_{i+1/2,j,k} = u_{i+1/2,j,k} + \delta t\{&(1/\delta x)[(u_{i,j,k})^2 - (u_{i+1,j,k})^2] \\ &+(1/\delta y)[(uv)_{i+1/2,j-1/2,k} - (uv)_{i+1/2,j+1/2,k}] \\ &+(1/\delta z)[(uw)_{i+1/2,j,k-1/2} - (uw)_{i+1/2,j,k+1/2}] + g_x \\ &+(1/\delta x)(p_{i,j,k} - p_{i+1,j,k}) + (\nu/\delta x^2)(u_{i+3/2,j,k} \\ &-2u_{i+1/2,j,k} + u_{i-1/2,j,k}) + (\nu/\delta y^2)(u_{i+1/2,j+1,k} \\ &-2u_{i+1/2,j,k} + u_{i+1/2,j-1,k}) + (\nu/\delta z^2)(u_{i+1/2,j,k+1} \\ &-2u_{i+1/2,j,k} + u_{i+1/2,j,k-1})\},\end{aligned} \tag{2}$$

for each velocity component u,v, and w of cell i, j, k. Although this system of equations is complex, the solution process is straightforward. In choosing an explicit formulation, we have made a tradeoff between ease of implementation and the maximum stable timestep for numerical integration. However, by approximating the environment using a relatively coarse grid (see Section 3.2), the condition that [3]

$$1 > max[u\frac{\delta t}{\delta x}, v\frac{\delta t}{\delta y}, w\frac{\delta t}{\delta z}] \tag{3}$$

be satisfied everywhere for the solution to be stable, can be met with a timestep that is still sufficiently large to animate motion efficiently (see Section 8).

To move the solution forward, velocities and pressures from the previous iteration are taken directly from individual cells and plugged into (2) to give the new velocities for the current iteration ($\tilde{u}$,$\tilde{v}$,$\tilde{w}$). In some cases, velocities are required that do not lie on cell faces, in which case they are averaged over the nearest available values, e.g., $u_{i,j,k} = \frac{1}{2}(u_{i+1/2,j,k} + u_{i-1/2,j,k})$, and the square of a quantity, e.g., u^2 at (i, j, k), is the square of the average, $(u_{i,j,k})^2$.

The new velocities are labeled with a tilde because the direct application of (2) does not ensure physical correctness. Due to the rectangular discretization of the environment, each individual cell may not explicitly satisfy the criteria that mass be conserved and that the fluid is incompressible. Also, the new pressure field needs to be determined. These constraints are satisfied simultaneously by solving the mass conservation, or continuity equation [3],

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{4}$$

which essentially says that the net fluid flow into or out of a cell is zero.

Consider a cell i, j, k. The divergence of fluid within the cell, or "missing mass", is given by [11]

$$\begin{aligned} D_{i,j,k} &= -((1/\delta x)(u_{i+1/2,j,k} - u_{i-1/2,j,k}) \\ &+ (1/\delta y)(v_{i,j+1/2,k} - v_{i,j-1/2,k}) \\ &+ (1/\delta z)(w_{i,j,k+1/2} - w_{i,j,k-1/2})). \end{aligned} \tag{5}$$

Notice that this is a finite difference approximation to the continuity equation. A positive D therefore, represents an influx of fluid, and in the real world would correspond to an increase in cell pressure and subsequent increase in fluid outflow from the cell. Similarly, a negative D lowers internal pressure and increases inflow from neighboring cells. Thus if the change in cell pressure is scaled according to the divergence in the cell, and the face cell velocities are adjusted according to the change in pressure, the cell can be made to satisfy (4). The change in pressure for a cell is

$$\delta p = \beta D, \tag{6}$$

where β is given by [11]

$$\beta = \beta_0/2\delta t(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2}), \tag{7}$$

and β_0 is a relaxation coefficient within the range [1,2]. The cell face velocities are then updated

according to δp such that

$$\begin{aligned}
u_{i+1/2,j,k} &= u_{i+1/2,j,k} + (\delta t/\delta x)\delta p, \\
u_{i-1/2,j,k} &= u_{i-1/2,j,k} - (\delta t/\delta x)\delta p, \\
v_{i,j+1/2,k} &= v_{i,j+1/2,k} + (\delta t/\delta y)\delta p, \\
v_{i,j-1/2,k} &= v_{i,j-1/2,k} - (\delta t/\delta y)\delta p, \\
w_{i,j,k+1/2} &= w_{i,j,k+1/2} + (\delta t/\delta z)\delta p, \\
w_{i,j,k-1/2} &= w_{i,j,k-1/2} - (\delta t/\delta z)\delta p,
\end{aligned} \tag{8}$$

and the cell pressure is updated according to

$$\tilde{p}_{i,j,k} = p_{i,j,k} + \delta p. \tag{9}$$

Use of the above equations satisfies (4) for a single cell, but neighboring cells may now have a non-zero divergence [11]. In order for the whole mesh to simultaneously satisfy (4), the pressure and velocities are first adjusted using (6)$\cdots$(8) for every cell in the grid. This procedure is then repeated until all cells in the flow field have a divergence less than some prescribed small ϵ. With a β_0 of 1.7 and ϵ of 0.0001, the examples shown in this paper converged in 3–6 sweeps on average. Once convergence is achieved, the fluid is considered to be locally incompressible and the velocity and pressure fields are complete for buoyant object inclusion and the start of the next cycle.

3.2 Boundary Conditions

The boundary conditions for our model are set automatically once the contents of each cell in the mesh have been determined. They are also homogeneous. That means that once they have been set, the Navier-Stokes equations can be applied blindly without determining exactly where surfaces or obstacles lie. This makes for cheap computation because boundary conditions need only be checked once at the beginning of an iteration, rather than for the velocity component calculation of each cell. A boundary is an interface between the fluid and a solid obstacle, or between the fluid and atmosphere, or a point at which fluid flows into or out of the system. In all cases, generalizing assumptions about the shape of static obstacles, and the position of free surfaces can greatly reduce the amount of work that we have to do, without compromising accuracy or realism.

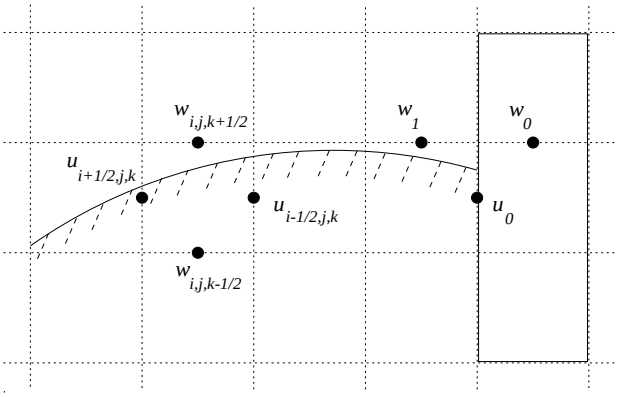


Figure 2: Setting boundary conditions on the free surface and across obstacle boundaries.

3.2.1 Stationary Obstacles

Consider Figure 2, which shows an obstacle and a free surface. We assume that the walls of an obstacle are always co-incident with the face of a computational cell. It then becomes a trivial process to set correct solid obstacle boundary conditions, that is, velocity and pressure for use in the finite difference expressions. For example, the component of fluid velocity normal to the face of a non-permeable obstacle is zero. Because obstacle and cell faces are coincident, the normal velocities are set directly ($u_0 = 0$ in the figure). In the case of a non-slip obstacle which exerts a drag on the fluid, the tangential velocity at the boundary is also zero. This is set indirectly by making the tangential cell face velocity inside the boundary cell equal and opposite to that outside in the fluid ($w_0 = -w_1$). Finally, the pressure in the boundary cell, which is also needed for the finite difference calculation, is set equal to the pressure in the adjacent fluid cell, preventing any acceleration across the boundary.

Another useful type of obstacle is a free-slip boundary. The treatment of pressure and velocity is the same as for a non-slip boundary except that the inner tangential velocity is set equal to that outside in the fluid ($w_0 = w_1$). A free-slip boundary can be thought of as a plane of symmetry for motion tangential to it, thus it provides a convenient way to bound a flow field. [1]

3.2.2 Inflow and Outflow

Fluid can easily flow into or out of the system by virtue of inflow or outflow boundary cells. For inflow, the required input velocity is set on the cell faces and held fixed throughout the calculation. In the case of an outflow boundary, velocities are initially set equal to the tentative velocity field in adjacent fluid

[1] All of the examples shown in this paper are bounded by a layer of free-slip boundary cells so that sides of a scene which are *open* do not effect the flow.

cells and then allowed to relax without constraint during the pressure iteration step. This ensures that fluid can flow freely out of the system without causing any upstream artifacts.

3.2.3 Free Surface

Boundary conditions also need to be set on the free surface. When (2) is applied to a surface cell, velocities and pressures are needed from adjacent empty cells. We assume that for most applications, if the wavelength of any disturbance is longer than a few inches, forces due to surface tension will be negligible. We then relax the constraint that we need to know exactly where in a cell the surface lies. Thus, if any part of a free surface passes through a cell, that cell is labeled as a *Surface* cell, and the equation of continuity (4) is used to set boundary velocities. Consider a two dimensional surface cell which is surrounded on three sides by cells containing fluid. The velocity on the remaining surface side is set so that the divergence D of the fluid in the cell is zero. So referring to Figure 2,

$$w_{i,j,k+1/2} \quad = \quad w_{i,j,k-1/2} - (\delta z/\delta x)(u_{i+1/2,j,k} - u_{i-1/2,j,k}). \tag{10}$$

If the cell had two sides which face an empty cell, we require that $\partial u/\partial x$ and $\partial w/\partial z$ both vanish separately, that is that each open side velocity equals the velocity of the side of the cell opposite it. This also satisfies (4). Finally, for the case in which three sides are open, the side opposite the fluid carries the velocity of that side, while the remaining two sides follow freely the effects of the body force and do not otherwise change. A three dimensional surface cell has velocity components set in an analogous fashion, leading to 64 distinct *Empty–Fluid* configurations. The pressure in a surface cell is set to the applied atmospheric pressure or forcing pressure function (see Section 7.2).

4 Tracking fluid position

We have described a method for solving the full Navier-Stokes equations over a finite difference mesh. From the mesh we want to generate a smooth and accurate representation of the actual fluid surface position. We also want to track the motion of such a surface over time, so that we can adjust the contents of the mesh accordingly (i.e., *Full*, *Surface*, or *Empty*). Finally, to avoid aliasing, the resolution of the surface should not be restricted by the coarse resolution of the mesh. With these goals in mind three methods of surface identification have been developed, each of which is useful for a particular class of liquid phenomena.

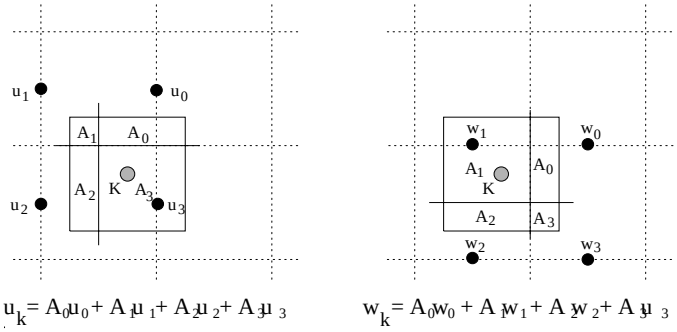


Figure 3: Area weighting interpolation scheme for determining local fluid velocity for a marker k.

4.1 Marker Particles

The simplest and most functional way to track fluid position in 2D is to convect massless marker particles with local fluid velocity. In this way particles are continuously introduced at inflow boundaries and removed if they cross an outflow boundary, and can splash and flow freely. A particle's new position is found using an area weighting interpolation over the four nearest cell velocities (See Figure 3) and multiplying the resultant velocity by the current timestep. The finite difference mesh is then labeled as follows:

- A cell containing no particles is *Empty*.
- A cell containing at least one particle that is adjacent to an *Empty* cell is a *Surface* cell.
- A cell containing at least one particle that is not a *Surface* cell is a *Full* cell.

The use of marker particles can highlight the full range of internal fluid motion such as rotation and splashing at a greater resolution than the finite difference mesh. It is important to note that the particles do not represent a mass of fluid. They are used to define the position of the surface only, and have no effect on the calculation. Frames from two dimensional animations using marker particles are shown in Figure 4. Figure 4(b) shows that an initial pulse of water has struck the sides of a concrete tank and has been projected up into the air. This jet eventually overturns and crashes back down into the growing pool. Marker particles are ideal for animating violent phenomena such as overturning waves because they define the position of the fluid exactly, regardless of how complex the surface has become. Figure 4 is discussed in more detail in Section 8.

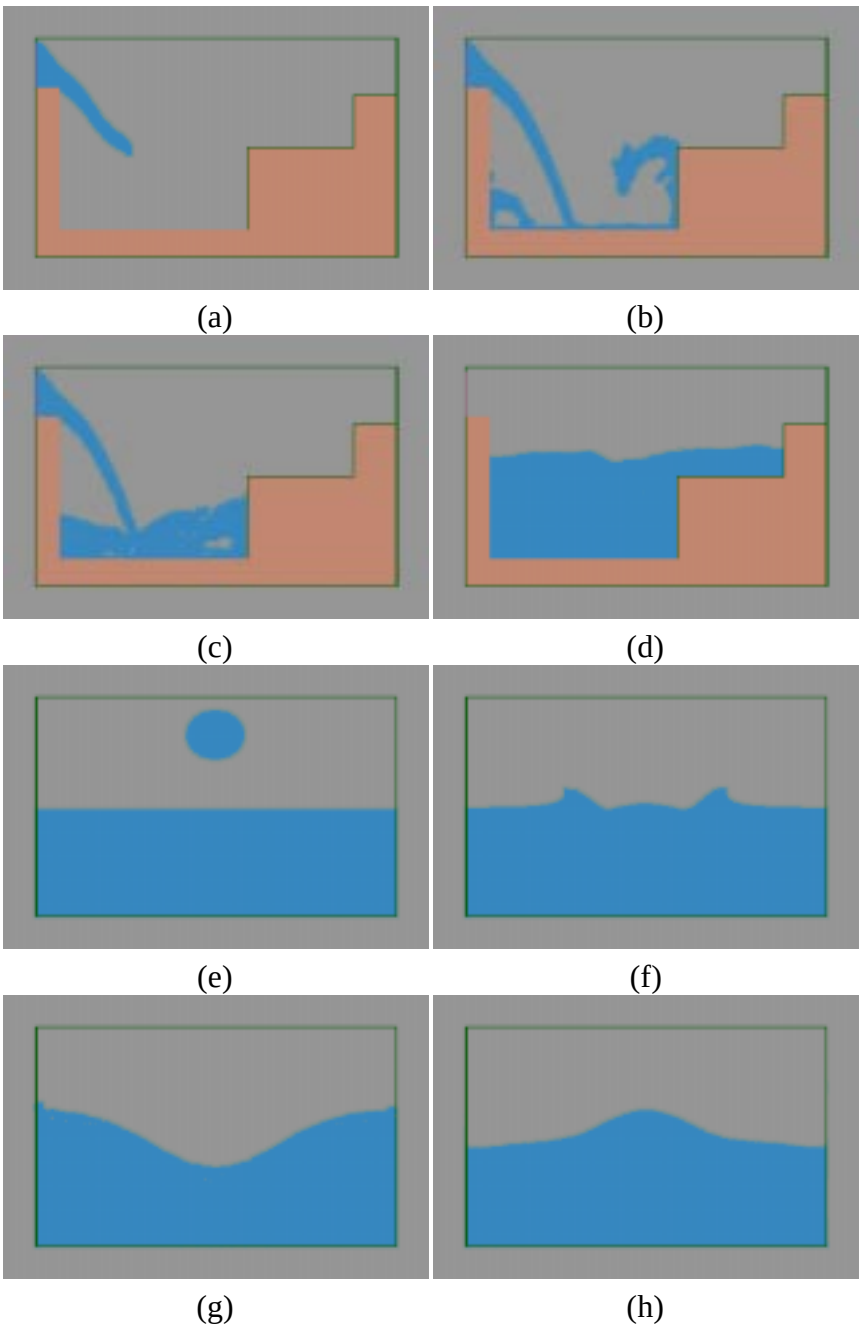


Figure 4: Frames from two dimensional animations making use of marker particles. A jet of water splashes into a concrete tank (a-d). A drop of water splashes into a shallow pool (e-h).

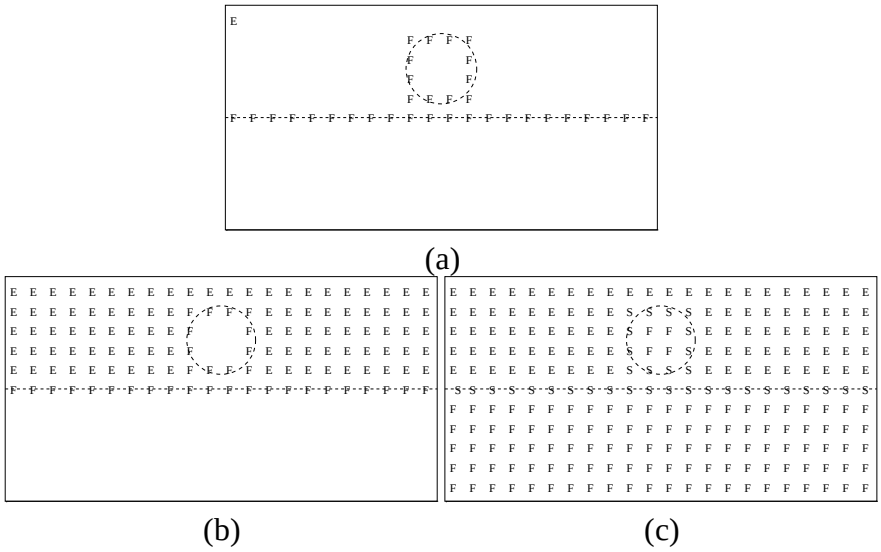


Figure 5: Region growing used to identify cell contents when using free surface particles. Initial state showing *Surface* cells S (a). After growing *Empty* cells E (b). Final state after growing *Full* cells F (c).

4.2 Free Surface Particles

Marker particles can also be used to precisely delineate any free-surfaces in a scene. Instead of appearing within every cell containing fluid, a grid of markers is placed along the boundaries between fluid and obstacles or air. This grid is convected with local velocity as before. However, the number, distribution, and connectivity of particles are allowed to change dynamically as the position of the surface changes. The rules for removing and adding particles are simple. If two particles become too close together, delete both of them and connect their neighbors. If two particles become too far apart, insert a new particle on the link between them. This ensures that the surface always remains continuous and that colliding surfaces are smoothly connected. In two dimensions this method is particularly useful because it is fast and can easily account for multiple surfaces.

Cell configuration in 2D for each cycle is determined using a region growing algorithm. Firstly, all the cells that contain a surface marker are set to *Full* and all other cells are flagged as *Unknown* (See Figure 5(a)). An *Empty* region is then grown from a cell in the mesh, usually the corner above the origin, which is known to be empty of particles at all times during the computation. Growing advances from this starting cell using a 4-connected search. In this way, any *Unknown* cells that are adjacent to an existing *Empty* cell are also set to *Empty*. When there are no more adjacent *Empty*-*Unknown* pairs (Figure 5(b)), a second region is grown by setting *Unknown* cells that are adjacent to a *Full* cell to *Full*. This process is repeated, alternating between *Empty* and *Full* regions, always growing until a boundary or *Full* cell is hit. Finally, the cells that contain the original surface markers are set to *Surface* (Figure 5(c)).

4.3 Height Field

Liquid in the real world often has a surface that is single valued. Examples of this are puddles, rivers, or the ocean (as long as there are no overturning waves). For such cases the position of the surface can be calculated without using marker particles because we no longer need to track the complex geometry caused by overturning. We define the surface height along the y axis, at the center of each vertical column of cells in the three-dimensional mesh. The change in local surface elevation at each timestep is determined by the local fluid velocity, that is, by the vertical component of the fluid motion plus the horizontal convection of the surface elevation from adjacent cell columns,

$$\frac{\partial h}{\partial t} = w - u(\frac{\partial h}{\partial x}) - v(\frac{\partial h}{\partial y}), \tag{11}$$

where h is the surface height. This equation can be approximated by a finite difference expression [11]

$$\begin{aligned} h_{i,j}^{t+\delta t} = \; & h_{i,j}^{t} + \delta t\{\overline{w}_{i,j,k}^{t+\delta t} \\ & + \frac{(h_{i-1,j}^{t} - h_{i+1,j}^{t})}{4\delta x}(u_{i+1/2,j,k}^{t+\delta t} + u_{i-1/2,j,k}^{t+\delta t}) \\ & + \frac{(h_{i,j-1}^{t} - h_{i,j+1}^{t})}{4\delta y}(v_{i,j+1/2,k}^{t+\delta t} + v_{i,j-1/2,k}^{t+\delta t})\}. \end{aligned} \tag{12}$$

This expression is used to update the position of the height field once the velocity and pressure fields have been calculated. It is important to note that despite superficial similarities to the method used by Kass *et al.* in [7], the height field equation is very different. Here, surface elevation is driven by the underlying fluid velocity. Therefore, velocity or pressure disturbances anywhere in the fluid volume can affect the surface (see Examples). Cell configuration for the height field approach is trivial. Cells crossed by the height field are *Surface* cells while those above it are *Empty*, and those below it are *Full*.

For dramatic effects such as crashing waves or splashing, the height field can be combined with the marker particles. Whenever the vertical velocity of the surface is greater than some positive threshold, a set of particles are introduced just below the surface and the local fluid velocity is used to set their initial velocity. The particles are then removed from the Navier-Stokes calculation and affected only by gravity, wind, and air resistance. For more realism there is a small probability that airborne particles will separate over time, giving the appearance of dispersing spray. Foam is introduced whenever spray collides with the height field. Foam has a random life span proportional to the size of its parent spray particle. There is interesting discussion of topics related to the use of particle systems for fluid animation in [4, 5, 10].

5 Buoyancy

Rigid dynamic objects can be included in a scene using the velocities and pressures calculated using the Navier-Stokes equations. Specifically, we assume that each rigid object is discretized and consists of a set of nodes n_i. For each model surface node n_i which is within the fluid, the force acting on this node is calculated based on the following formula

$$\mathbf{f}_{n_i} = -\nabla p_i dV_i + m_i \mathbf{g}, \tag{13}$$

where dV_i is a volume associated with the submerged node of the object and ∇p_i is the gradient vector of the pressure. Each component of ∇p_i is computed in discrete form as

$$(\nabla p_i)_{x_j} = \frac{p_{n_i} - p_{n_i,x_j}}{\delta x_j}, \qquad j = 1, 2, 3\ , \tag{14}$$

where p_{n_i} is the pressure in the cell containing n_i, and p_{n_i,x_j} is the pressure in the previous cell in the x_j direction. Also, **g** is the gravitational acceleration, and m_i is the nodal mass assuming lumped masses. The total force on the object due to the fluid motion and gravity is given in discrete form by

$$\mathbf{f}_{fluid} = \sum_i \mathbf{f}_{n_i}. \tag{15}$$

Based on the total force acting on each node, we compute the generalized external forces $\mathbf{f}_q$ (total force and torque acting on the object) as demonstrated in [9] and we compute its motion based on the Lagrange equations of motion

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} = \mathbf{f}_q + \mathbf{g}_q, \tag{16}$$

where **M** and **D** are the object's generalized mass and damping matrices, **q** are the model translational and rotational degrees of freedom, and $\mathbf{g}_q$ are the generalized coriolis and centrifugal forces. The mass matrix, **M**, is derived directly from the object in question [9], and is unaffected by the fluid model. The damping matrix, **D**, also has the same form as in [9], but with the damping coefficients adjusted proportional to the relative velocity between a node, n_i, and the local fluid.

In order to handle collisions of the floating objects with static obstacles, we also apply the techniques developed in [9] for collision detection and collision force computation.

The floating objects that we used in the examples are small compared to the mesh size and therefore it is possible to make the simplifying assumption that they do not effect the water flow. Thus, they act

like large marker particles moving and rotating according to local forces. For the objects to influence the motion of the fluid, more sophisticated techniques need to be employed.

6 Summary of the Navier-Stokes Algorithm

The complete algorithm for solving the Navier-Stokes equations and tracking the fluid surface can be summarized in the following steps;

1. Define obstacles and starting fluid configuration, and place dynamic objects.
2. Set initial pressure and velocity conditions.
3. Determine cell contents depending on the method used to track the surface.
4. Set up boundary conditions for the free surface and obstacle cells.
5. Compute $\tilde{u}, \tilde{v}, \tilde{w}$ for all *Full* cells.
6. Perform the pressure iteration for all *Full* cells.
7. Recalculate boundary velocities for *Surface* cells.
8. Update the position of the surface and objects.
9. Go to step 3.

7 Control

An important part of the animation process is specifying how objects in a scene will move. Doing this for a fluid surface is difficult because the governing equations (1) are strongly coupled and non-linear. Large scale behavior of the system can be controlled by altering various constants such as gravity and viscosity, but it is difficult to specify a motion then solve backwards to find the correct boundary conditions to cause it. However, there are two places in our algorithm where coercion can be applied to the fluid. These can easily be exploited to yield effective methods for controlling the fluid surface.

7.1 Inflow and outflow velocities

A time dependent function can be used to determine the rate at which fluid is pumped into a scene or the rate at which it is allowed to exit, producing a variety of effects. For example, a broken dam initially generates a high input rate, but tails off exponentially as the water level drops. Also, for animating a river scene, a varying inflow and outflow rate will simulate different classes (speed, turbulence) of water flow without requiring any changes to the environment model.

7.2 Surface pressure history

Perhaps the most natural way to specify surface behavior is to model nature. As wind blows across a liquid surface, small, low pressure vortices induce a local change in surface elevation. This in turn, disturbs the airflow over the surface, changing the pressure. Gravity then provides a restoration force for the initial perturbation which results in oscillation. Thus, over time, the process is amplified and a wind driven wave is born. A similar effect can be achieved in a shorter time by applying a forcing pressure history to the free surface during the Navier-Stokes computation. This may be constant, time dependent, or depend on the present height of the surface. For example, in two dimensions, interesting waves can be developed using the forcing function

$$p_{applied}(z) = \frac{A + Bcos(Cz - \omega t)}{\delta t}, \tag{17}$$

where $p_{applied}$ is the pressure within a ***Surface*** cell, $A/\delta t$ is the mean pressure, and B and C are constants derived from the desired wave motion. From Figure 6, if $2L$ is the wavelength of the oscillation and D is the mean depth of the fluid, then,

$$B = a\sqrt{-\frac{gD}{C}}, \tag{18}$$

and

$$C = \frac{\pi}{L}, \tag{19}$$

where a is the wave amplitude, and g is gravity. Such a function is used to set the applied pressure boundary condition on the free surface (See section 3.2.3).

8 Examples

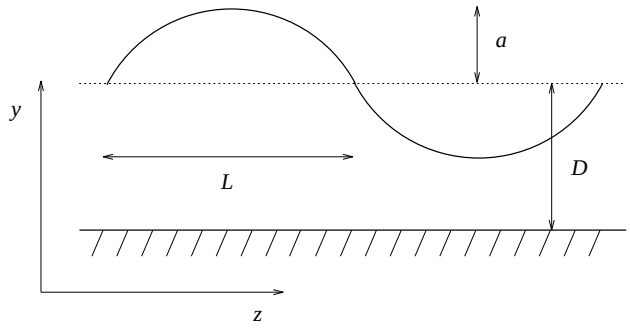


Figure 6: Deriving constants for an applied pressure function.

Figure 7: Starting configuration for 2D marble soup animation.

We present a number of examples to show different aspects of the system described in this paper. Running times are given for a Silicon Graphics Crimson R4000. They do not vary linearly with the size of each problem because other factors, such as the total number of *Full* cells present, or the speed of the flow, make a larger contribution to the amount of CPU time required.

The first example (Figure 4(a-d)), is a two dimensional animation of a water jet splashing into a concrete tank. The water motion was calculated over a 30x40 grid of cells, and marker particles were used to delineate fluid position. Two input rates were specified; water inflow and particle inflow. The jet had a velocity of 0.8 ms^{-1} and new particles were introduced at the inflow boundary at a rate of 500 particles per second. It is important to note that the only overhead associated with the marker particles is the cost of moving and displaying them. A relatively sparse distribution of particles was used in this case to clearly show that the model can account for colliding surfaces, overturning waves, and arbitrary splashing. A later frame from the animation shows that after the jet is turned off the vortices in the tank slow down and the surface starts to settle (Figure 4(d)). This animation ran for 4500 iterations in just over sixteen minutes. The same grid size (30x40) was used again to animate a splashing drop (Figure 4(e-h)). Figures 4(e) and (f) show the starting configuration of the drop and its initial impact with the surface. The waves caused by the collision travel out to the sides of the pool (Figure 4 (g)), and are reflected back to give the characteristic fluid rebound at the epicenter of the splash. Particle density was set at 25 per cell. This animation ran for 2800 iterations in twelve minutes, slightly slower than the water

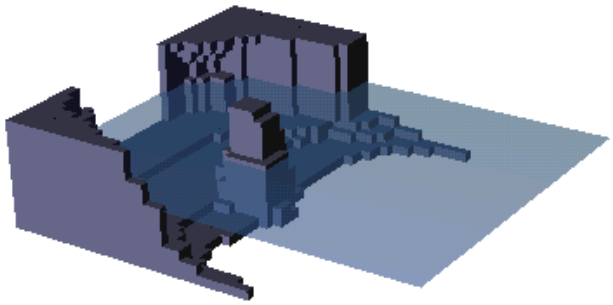


Figure 8: Calculation environment for the Moonlight Cove.

jet example above because the average number of *Full* cells per iteration was higher.

If the scene geometry is rotationally symmetric, computationally cheap two dimensional calculations can be made using linked chains of markers. The resulting profiles can then be used to define a surface of revolution. While this method is not a physically accurate way of generating three dimensional motion, it does give some simple effects which are good enough for graphics applications. Figure 9 shows two frames from an animation of a rigid marble dropping into a bowl of thick soup. The actual calculation was performed in two dimensions by setting the z axis resolution to 1. The curved side of the bowl was approximated as a series of steps, and a semicircular drop of liquid was aligned along the y axis (see Figure 7) to represent the marble. The drop was given an initial velocity of -0.2 ms^{-1}, and the viscosity of the soup was set relatively high (0.003). Two chains of particles were used to represent the free surfaces in the scene, 50 for the drop and 150 for the soup. The calculation was run for two thousand iterations at a resolution of 15x30x1 taking twelve minutes. The scene was rendered directly from the positions of the markers. Each chain was used to define the profile of a surface of revolution, which was smoothed using a series of bicubic splines. Finally, the marble and other objects were added, and the whole scene rendered using Pixar's PhotoRealistic RenderMan [16]. The liquid surface was colored using a straightforward environment map, taking account of Fresnels's law [2] to calculate the fraction of light reflected toward the camera, or transmitted to the bottom of the bowl.

The soup example clearly shows some of the advantages of our model. The liquid drop for the calculation is the same size as the marble object, so after impact the mean surface level has risen correctly. Also, the coupling between pressure and velocity develops as a non-linear oscillation which continues long after the wave due to the collision has subsided. Previous computer graphics fluid models would have accounted for the surface wave, but not for the accompanying pressure wave which is responsible for most of the final motion.

The first full 3D example is an animation titled *Moonlight Cove* (Figure 10). A 50x15x40 mesh was used to finely resolve the effect of two large ocean waves crashing into a shallow cove. Submerged rocks, and an irregular sea bottom, focus the waves into the center of the cove, causing a number of interesting features on the water surface. The wave becomes steeper as the water depth decreases, and eddies and pressure waves appear to the left of, and behind the initial obstacle (Figure 10(b)).

Setting up the scene was straightforward and proceeded in two stages. First, a voxel based editor was used to define the initial distribution of rocks and water (Figure 8). The last plane of cells opposite the cove were then designated as inflow cells, with inflow velocity defined as

$$u = u + a\omega^2 cos\omega t, \tag{20}$$

where a was the desired wave amplitude and ω the desired wave frequency. The calculation was run for $2/\omega$ seconds, then the inflow cells were changed to outflow and water allowed to leave the system at its natural rate. This approach resulted in two full waves while allowing the added water volume to flow back out of the scene once the waves had been reflected. The animation took two and a half hours to complete and ran for 20,000 iterations.

RenderMan was also used to render this example. Two spline meshes were used; one generated from the surface height field, and another from the distribution of boundary cells. The water surface was rendered as a glass-like object with small disturbances generated using the long crested wave model suggested in [14]. Detail in the rocks was provided using a displacement map and suitable noise function on the spline surface.

A second 3D example is shown in Figure 11. A pressure wave caused by an opening sluice gate is forced along a channel. The wave travels over a submerged pyramid, is reflected through ninety degrees, and finally crashes over a semi-submerged rock. The importance of calculating pressure and velocity throughout the whole volume is clear from the pictures. Figure 11(a) shows the trailing wave that builds up behind the submerged obstacle as water flows around it. This trailing wave survives until the original wave crashes over the semi-submerged obstacle (Figure 11(b)) and flows out of the system. The mean water level has also risen because of the inflow from the sluice gate. This example was calculated over a 40x12x40 grid, took an hour to compute, and ran for 8,000 iterations. For speed, standard Silicon Graphics hardware routines were used for rendering in this case.

The frames in Figure 12 show screen shots from an animation involving buoyant objects. Water flows into a closed container carrying soda cans along with it. When the flow is turned off, the cans gather at the far corner of the container because the walls in this example were set as non-slip so the tangential fluid velocity is zero. This simulates the effect that objects tend to gather in stagnant parts

of a flow. The water motion was precomputed in thirty minutes over a 30x10x20 grid. The soda cans were added later using an interactive editor which takes a precomputed velocity and pressure field, and calculates the forces on an object within the mesh. In this way, many different shapes and sizes of object can be experimented with, without having to re-do the fluid calculation.

9 Conclusions

We have presented a comprehensive method for animating liquid phenomena. A direct simulation technique is used to solve the Navier-Stokes equations in two or three dimensions yielding a range of behavior unavailable with previous computer graphics fluids models. The method does come with a computational cost, which, like other volumetric techniques, scales proportional to the fourth power of the spatial resolution. However, by careful discretization of the environment, the most expensive part of the computation can be made at a low resolution. Detail generation is then achieved by directly calculating the position of a height field representing the fluid surface, or tracking the changing connectivity of surface marker particles convected with local fluid velocity. The model allows for some novel control techniques that can be used to generate a variety of interesting effects, and is suitable as a front end to many of the more inspiring water rendering algorithms available.

References

[1] Chen, J., and Lobo, N., (1995) "Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations," Graphical Models and Image Processing, March 1995, pp. 107–116.

[2] Cook, R., and Torrance, K., (1982) "A Reflectance Model for Computer Graphics," ACM Transactions on Graphics, 1(1), pp. 7–24.

[3] Fletcher, C.A.J., (1990) *Computational Techniques for Fluid Dynamics*, Springer Verlag, Sydney, 1990.

[4] Fournier, A. and Reeves, W., (1986) "A Simple Model of Ocean Waves," Proceedings of SIGGRAPH '86, in Computer Graphics, 20(3), pp. 75–84.

[5] Goss, M.E., (1990) "A Real-time Particle System for Display of Ship Wakes," IEEE Computer Graphics and Applications, 10(3), pp. 30-35.

[6] Harlow, F.H., and Welch, J.E., (1965) “Numerical Calculation of Time-Dependent Viscous Incompressible Flow,” Phys. Fluids, 8, pp. 2182–2189.

[7] Kass, M., and Miller, G., (1990) “Rapid, stable fluid dynamics for computer graphics,” Proceedings of SIGGRAPH ’90, in Computer Graphics, 24(3), pp. 49–57.

[8] Max, N., (1981) “Vectorized procedural models for natural terrain: Waves and islands in the sunset,” Proceedings of SIGGRAPH ’81, in Computer Graphics, 15(3), pp. 317–324.

[9] Metaxas, D., and Terzopoulos, D., (1992) “Dynamic Deformation of Solid Primitives with Constraints,” Proceedings of SIGGRAPH ’92, in Computer Graphics, 26(3), pp. 309–312.

[10] Miller, G., and Pearce, A., (1989) “Globular Dynamics: A Connected Particle System for Animating Viscous Fluids,” Computers and Graphics, 13(3), 1989, pp. 305–309.

[11] Nichols, B.D., (1973) “Calculating Three Dimensional Free Surface Flows in the Vicinity of Submerged and Exposed Structures”, J. Comp. Phys., 12, pp. 234–245.

[12] Nishita, T., and Nakamae, E., (1994) “Method of Displaying Optical Effects Within Water: Using the Accumulation Buffer,” Proceedings of SIGGRAPH ’94, (July 1994), pp. 24–29.

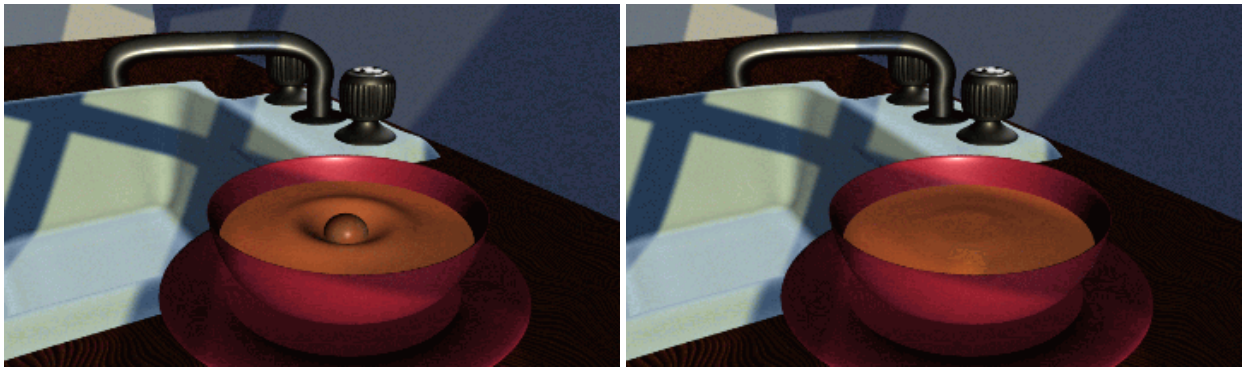
[13] Peachy, D., (1986) “Modeling Waves and Surf,” Proceedings of SIGGRAPH ’86, in Computer Graphics, 20(3), pp. 65–74.

[14] Schachter, B., (1980) “Long crested wave models,” Computer Graphics and Image Processing, 12, (February 1980) , pp. 187–201.

[15] Ts’o, P., and Barsky, B., (1987) “Modeling and rendering waves: Wave-Tracing using Beta-Splines and Reflective and Refractive Texture Mapping,” ACM Transactions on Graphics, 6(3), pp. 191–214.

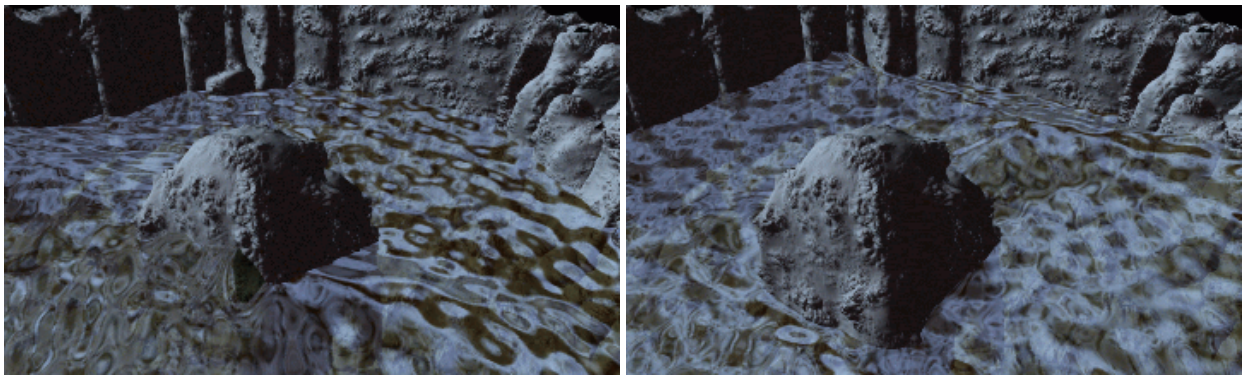
[16] Upstill, S., (1990) *The RenderMan Companion*, Addison Wesley, New York, 1990.

[17] Watt, M., (1990) “Light-Water Interaction using Backward Beam Tracing,” Proceedings of SIGGRAPH ’90, in Computer Graphics 24, pp. 377–385.



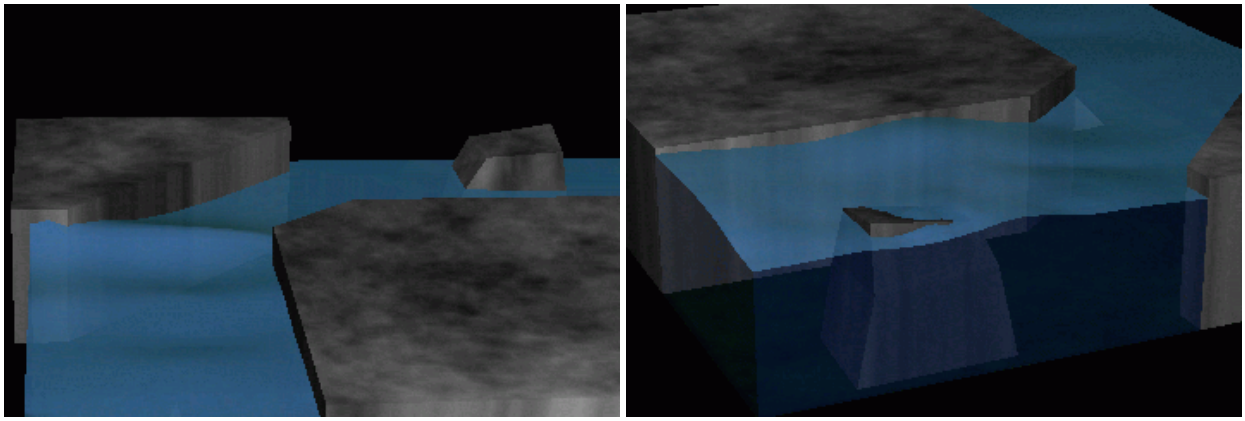
(a) (b)

Figure 9: A marble dropping into a bowl of thick soup. Initial collision (a). Oscillation due to coupling between pressure and velocity (b).



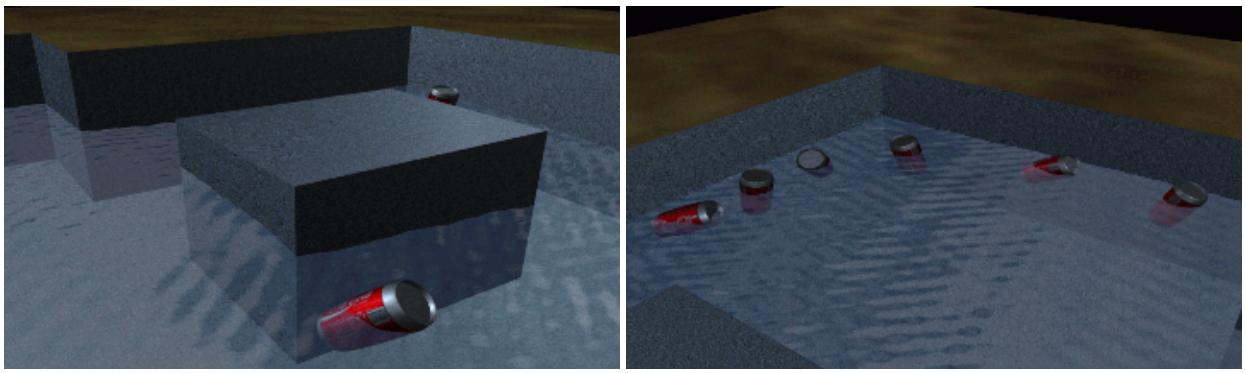
(a) (b)

Figure 10: Moonlight Cove. Two ocean waves crash into a shallow cove. Pressure and velocity effects throughout the water volume manifest themselves at the surface (a,b).



(a) (b)

Figure 11: A pressure wave travels over a submerged obstacle (a), is reflected by ninety degrees and crashes over a semi-submerged rock (b).



(a) (b)

Figure 12: Dynamic objects. Soda cans are carried along with the incoming water, colliding with obstacles (a), and getting caught in local eddies (b).