Practical Animation of Liquids

Nick Foster*
PDI/DreamWorks

Ronald Fedkiw**
Stanford University

Abstract

We present a general method for modeling and animating liquids. The system is specifically designed for computer animation and handles viscous liquids as they move in a 3D environment and interact with graphics primitives such as parametric curves and moving polygons. We combine an appropriately modified semi-Lagrangian method with a new approach to calculating fluid flow around objects. This allows us to efficiently solve the equations of motion for a liquid while retaining enough detail to obtain realistic looking behavior. The object interaction mechanism is extended to provide control over the liquid's 3D motion. A high quality surface is obtained from the resulting velocity field using a novel adaptive technique for evolving an implicit surface.

Keywords: animation, computational fluid dynamics, implicit surface, level set, liquids, natural phenomena, Navier-Stokes, particles, semi-Lagrangian.

1. Introduction

The desire for improved physics-based animation tools has grown hand in hand with the advances made in computer animation on the whole. It is natural then, that established engineering techniques for simulating and modeling the real world have been modified and applied to computer graphics more frequently over the last few years. One group of methods that have resisted this transition are those used to model the behavior of liquids from the field of computational fluid dynamics (CFD). Not only are such techniques generally complex and computationally intensive, but they are also not readily adaptable to what could be considered the basic requirements of a computer animation system.

One of the key difficulties encountered when using these methods for animation directly characterizes the trade off between simulation and control. Physics-based animations usually rely on direct numerical simulation (DNS) to achieve realism. In engineering terms, this means that initial conditions and boundary conditions are specified and the process is left to run freely with only minor influence on the part of the animator. The majority of engineering techniques for liquid simulation assume this model.

From an animation viewpoint, we are interested in using numerical techniques to obtain behaviors that would be prohibitive to model by hand. At the same time we want control over the global, low frequency motion so we can match it to the behavior we are trying to create. This then becomes the goal when transitioning between engineering and computer animation; preserve as much of the realistic behavior as feasible while allowing for control over motion on both a local and global scale. This has to be achieved without compromising the overall requirement of a visually coherent and realistic look.

* nickf@pdi.com, ** fedkiw@cs.stanford.edu

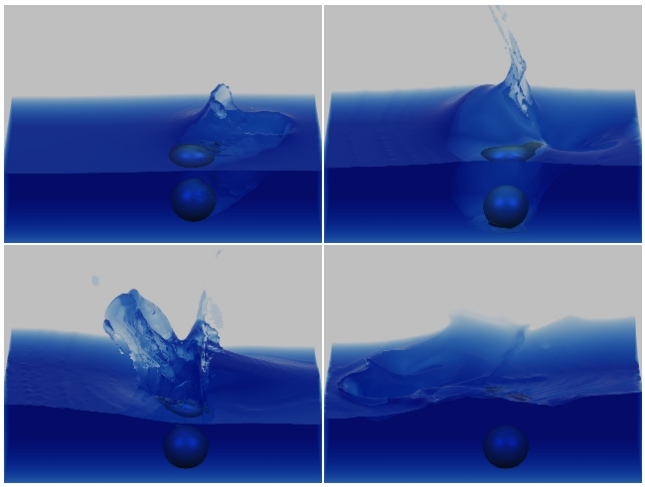

***Figure 1**: A ball splashes into a tank of water.*

This paper specifically addresses these issues for liquid animation. The method presented is for animating viscous liquids ranging from water to thick mud. These liquids can freely mix, move arbitrarily within a fixed three-dimensional grid and interact realistically with stationary or moving polygonal objects. This is achieved for animation by trading off engineering correctness for computational efficiency.

We start with the Navier-Stokes equations for incompressible flow and solve for liquid motion using an adaptation of a semi-Lagrangian method introduced recently to graphics for solving fluid flows [25]. These methods usually result in mass dissipation. While not an issue for gas or smoke, this is visually unacceptable for modeling liquids. We correct for this by tracking the motion of the liquid surface using a novel hybrid combination of inertialess particles and an implicit surface called a level set. The level set prevents mass dissipation while the particles allow the liquid to still splash freely. A useful consequence is that this combined surface can be rendered in a highly believable way.

The next innovation involves taking account of the effects of moving polygonal objects within the liquid. We develop a new technique that, while not accurate in an engineering sense, satisfies the physics of object/liquid interactions and looks visually realistic. This method is efficient and robust, and we show that it can be adapted to provide low frequency directional control over the liquid volume. This allows us to efficiently

calculate liquid behavior that would be impossible to get by hand, while at the same time allowing us to "dial-in" specific motion components.

When the techniques described above are applied together, the result is a comprehensive system for modeling and animating liquids for computer graphics. The main contributions of the system are a numerical method that takes the minimal computational effort required for visual realism combined with tailor-made methods for handling moving objects and for maintaining a smooth, temporally coherent liquid surface.

2. Previous Work

The behavior of a volume of liquid can be described by a set of equations that were jointly developed by Navier and Stokes in the early eighteen hundreds (see next section). The last fifty years has seen an enormous amount of research by the CFD community into solving these equations for a variety of engineering applications. We direct the interested reader to Abbot and Basco [1] which covers some of the important principles without being too mathematically dense.

Early graphics work concentrated on modeling just the surface of a body of water as a parametric function that could be animated over time to simulate wave transport [12, 22, 23]. Kass and Miller [17] approximated the 2D shallow water equations to get a dynamic height field surface that interacted with a static ground "object". Chen and Lobo [4] extended the height field approach by using the pressure arising from a 2D solution of the Navier-Stokes equations to modulate surface elevation. O'Brien and Hodgins [20] simulated splashing liquids by combining a particle system and height field, while Miller and Pearce [19] used viscous springs between particles to achieve dynamic flow in 3D. Terzopoulos, Platt and Fleischer [27] simulated melting deformable solids using a molecular dynamics approach to simulate the particles in the liquid phase.

Surface or particle based methods are relatively fast, especially in the case of large bodies of water, but they don't address the full range of motion exhibited by liquids. Specifically, they don't take advantage of the realism inherent in a full solution to the Navier-Stokes equations. They are also not easily adapted to include interaction with moving objects. Foster and Metaxas [11] modified an original method by Harlow and Welch [15] (later improved by others, see e.g. [5]) to solve the full equations in 3D with arbitrary static objects and extended it to include simple control mechanisms [9]. Foster and Metaxas also applied a similar technique to model hot gases [10]. Stam [25] replaced their finite difference scheme with a semi-Lagrangian method to achieve significant performance improvements at the cost of increased rotational damping. Yngve et al. used a finite difference scheme to solve the compressible Navier-Stokes equations to model shock wave and convection effects generated by an explosion [28].

3. Method Outline

The Navier-Stokes equations for describing the motion of a liquid consist of two parts. The first, enforces incompressibility by saying that mass should always be conserved, i.e.

$$\nabla \cdot \mathbf{u} = 0 , \qquad (3.1)$$

where $\mathbf{u}$ is the liquid velocity field, and

$$\nabla = \left(\partial/\partial x, \partial/\partial y, \partial/\partial z\right)$$

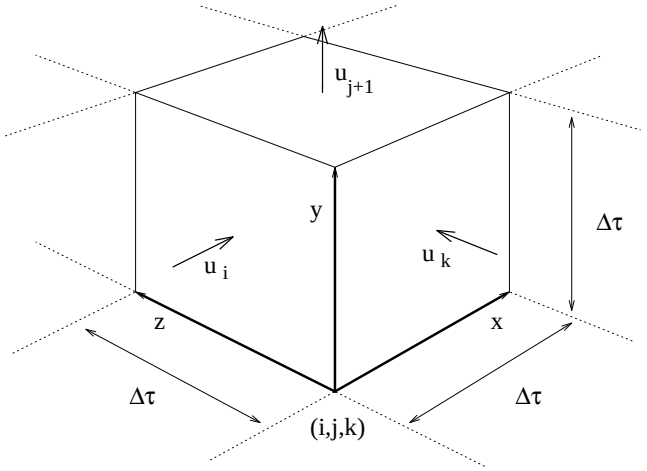

Figure 2: *A single grid cell with three of its six face velocities shown.*

is the gradient operator. The second equation couples the velocity and pressure fields and relates them through the conservation of momentum, i.e.

$$\mathbf{u}_t = \nu\nabla \cdot \left(\nabla \mathbf{u}\right) - \left(\mathbf{u} \cdot \nabla\right)\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{g} . \qquad (3.2)$$

This equation models the changes in the velocity field over time due to the effects of viscosity (ν), convection, density (ρ), pressure (p), and gravity ($\mathbf{g}$). By solving (3.1) and (3.2) over time, we can model the behavior of a volume of liquid. The new algorithm we are proposing to do this consists of six straightforward steps.

I. Model the static environment as a voxel grid.

II. Model the liquid volume using a combination of particles and an implicit surface.

Then, for each simulation time step

III. Update the velocity field by solving (3.2) using finite differences combined with a semi-Lagrangian method.

IV. Apply velocity constraints due to moving objects.

V. Enforce incompressibility by solving a linear system built from (3.1).

VI. Update the position of the liquid volume (particles and implicit surface) using the new velocity field.

These steps are described in detail in the following sections. Steps IV and V are presented in reverse order for clarity.

4. Static Environment

Equations (3.1) and (3.2) model a liquid as two coupled dynamic fields, velocity and pressure. The motion of the liquid we are modeling will be determined by evolving these fields over time. We start by representing the environment that we want the liquid to move in as a rectangular grid of voxels with side length $\Delta\tau$. The grid does not have to be rectangular, but the overhead of unused (non-liquid containing) cells will be low and so it is convenient. Each cell has a pressure variable at its center and shares a velocity variable with each of its adjacent neighbors (see figure 2). This velocity is defined at the center of the face shared by the two

neighboring cells and represents the magnitude of the flow normal to that face. This is the classic "staggered" MAC grid [15]. Each cell is then either tagged as being empty (available to be filled with liquid) or filled completely with an impermeable static object. Despite the crude voxelized approximation of both objects and the liquid volume itself, we'll show that we can still obtain and track a smooth, temporally coherent liquid surface.

5. Liquid Representation

The actual distribution of liquid in the environment is represented using an implicit surface. The implicit function is derived from a combination of inertialess particles and a dynamic isocontour. The isocontour provides a smooth surface to regions of liquid that are well resolved compared to our grid, whereas the particles provide detail where the surface starts to splash.

5.1 Particles

Particles are placed (or introduced via a source) into the grid according to some initial liquid distribution. Their positions then evolve over time by simple convection. Particle velocity is computed directly from the velocity grid using tri-linear interpolation and each particle is moved according to the inertialess equation $d\mathbf{x}_p/dt = \mathbf{v}_x$, where $\mathbf{v}_x$ is the fluid velocity at $\mathbf{x}_p$. Particles have a low computational overhead and smoothly integrate the changing liquid velocity field over time. The obvious drawback to using them, however, is that there is no straightforward way to extract a smooth polygonal (or parametric) description of the actual liquid surface. This surface is preferred because we want to render the liquid realistically using traditional computer graphics techniques. It is possible to identify it by connecting all the particles together into triangles, although deducing both the connectivity and set of surface triangles is difficult. In addition, since the particles do not generally form a smooth surface, the resulting polygonal mesh suffers from temporal aliasing as triangles "pop" in or out.

5.2 Isocontour

An alternative technique for representing the liquid surface is to generate it from an isocontour of an implicit function. The function is defined on a high resolution Eulerian sub-grid that sits inside the Navier-Stokes grid. Let each particle represent the center of an implicitly defined volumetric object (see Bloomenthal et al. [3] for a survey of implicit surfaces). Specifically, an implicit function centered at the particle location $\mathbf{x}_p$ with radius r is given by

$$\phi_p(\mathbf{x}) = \sqrt{(x_i - x_{pi})^2 + (x_j - x_{pj})^2 + (x_k - x_{pk})^2} - r .$$

The surface of that particle is defined as the spherical shell around $\mathbf{x}_p$ where $\phi_p(\mathbf{x})=0$. An implicit function, $\phi(\mathbf{x})$, is then defined over all the particles by taking the value of $\phi_p(\mathbf{x})$ from the particle closest to $\mathbf{x}$. If we sample $\phi(\mathbf{x})$ at each sub-grid point we can use a marching cubes algorithm [18] to tessellate the $\phi(\mathbf{x})=0$ isocontour with polygons. More sophisticated blend functions could be used to create an implicit function, however, we are going to temporally and spatially smooth $\phi(\mathbf{x})$ so it isn't necessary. We refer those interested in wrapping implicit surfaces around particles to the work of Desbrun and Cani-Gascuel [7].

The first step towards smoothing the surface is to normalize ϕ so that $|\phi(\mathbf{x})|$ equals the distance from $\mathbf{x}$ to the closest point on the zero isocontour. The sign of ϕ is set negative inside the liquid and

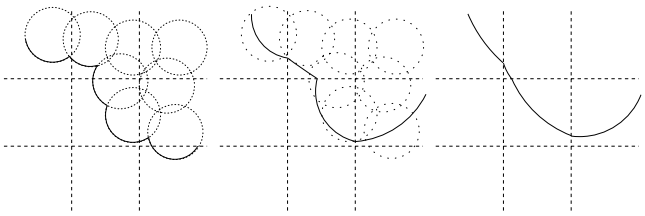

Figure 3: *The isocontour due to the implicit function around the particles, interpolated ϕ values, and smoothed ϕ values, respectively.*

positive outside. This signed distance function can be created quickly using the Fast Marching Method [24] starting from the initial guess of $\phi(\mathbf{x})$ defined by the particles as outlined above.

In order to smooth out ϕ to reduce unnatural "folds" or "corners" in the surface (see figure 3), a smoothing equation of the form

$$\phi_\eta = -S\left(\phi^{\eta=0}\right)\left(\left|\nabla\phi\right| - 1\right), \qquad (5.1)$$

is used to modify values of ϕ close to the $\phi(\mathbf{x})=0$ isocontour. $S(\phi)$ is a smoothed sign function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + \Delta\tau^2}} .$$

If applied for a few relaxation steps in fictitious time η (everything else remains constant), (5.1) smooths out the $\phi(\mathbf{x})=0$ isocontour while maintaining overall shape. Once smoothed, the isocontour can be ray traced directly using a root finding algorithm to identify the zero values of ϕ. A fast root finder can be built easily because at any sub-grid point the value of ϕ explicitly gives the minimum step to take to get closer to the surface. Note that the surface normal is given by $\mathbf{n} = \nabla\phi / \left|\nabla\phi\right|$.

By creating a smooth isocontour for each frame of animation, we get an improved surface representation compared to using particles alone. There are still drawbacks however. A high density of particles is required at the $\phi(\mathbf{x})=0$ isocontour before the surface looks believably flat. Particles are also required throughout the entire liquid volume even when it's clear that they make no contribution to the visible surface. The solution is to create ϕ once using the particles, and then track how it moves using the same velocity field that we're using to move the particles. This leads to a temporally smoothed dynamic isosurface known in the CFD literature as a level set.

5.3 Dynamic Level Set

An obvious way to track the evolution of the surface of a volume of liquid would be to attach particles directly to the surface in its initial position and then just move them around in the velocity field. This would require adding extra particles when the surface becomes too sparsely resolved, and removing them as the surface folds, or "splashes" back over itself. An alternative method which is intuitively similar, but that doesn't use particles, was developed by Osher and Sethian [21] and is called the level set method.

We want to evolve ϕ directly over time using the liquid velocity field $\mathbf{u}$. We have a smooth surface but need to conform, visually at least, to the physics of liquids. It has been shown [21] that the

equation to update ϕ under these circumstances has the following structure,

$$\phi_t + \mathbf{u} \cdot \nabla \phi = 0 \,. \qquad (5.2)$$

Using (5.2), the surface position is evolved over time by tracking $\phi(\mathbf{x})=0$. The $(\mathbf{u} \cdot \nabla \phi)$ term is a convection term similar to the $(\mathbf{u} \cdot \nabla)\mathbf{u}$ term in (3.2) implying that we could use a semi-Lagrangian method to solve this equation. However, since this equation represents the mass evolution of our liquid, the semi-Lagrangian method tends to be too inaccurate. Instead we use a higher order upwind differencing procedure [1] on the $(\mathbf{u} \cdot \nabla \phi)$ term. Fedkiw et al. [8] used this methodology to track a fluid surface and give explicit details on solving (5.2). This method can suffer from severe volume loss especially on the relatively coarse grids commonly used in computer graphics. This is clearly visible when regions of liquid break away during splashing and then disappear because they are too small to be resolved by the level set. We require visual coherency for this to be a useful graphics technique and so the level set method needs to be modified to preserve volume.

5.4 Hybrid Surface Model

Particle evolution is a fully Lagrangian approach to the mass motion while level set evolution is a fully Eulerian approach. Since they tend to have complementary strengths and weakness, a combined approach gives superior results under a wider variety of situations. Level set evolution suffers from volume loss near detailed features while particle evolution suffers from visual artifacts in the surface when the number of particles is small. Conversely, the level set is always smooth, and particles retain detail regardless of flow complexity. Therefore we suggest a novel combination of the two approaches.

At each time step we evolve the particles and the level set ϕ forward in time. Next, we use the updated value of the level set function to decide how to treat each particle. If a particle is more than a few grid cells away from, and inside the surface, as indicated by the locally interpolated value of ϕ, then that particle is deleted. This increases efficiency since particles are only needed locally near the surface of the liquid as opposed to throughout the entire liquid volume. In addition, for cells close to $\phi(\mathbf{x})=0$ that are sparsely populated, extra particles can be introduced "within" the isocontour. Thus, for a bounded number of particles, we get improved surface resolution.

Next, for each particle near the surface, the locally interpolated curvature of the interface, calculated as

$$k = \nabla \cdot \left(\nabla\phi / \left|\nabla\phi\right| \right),$$

is used to indicate whether or not the surface is smooth. Smooth regions have low curvature and the particles are ignored allowing the level set function to give a very smooth representation of the liquid surface. On the other hand, regions of high curvature indicate splashing. In these regions, the particles are a better indicator of the rough surface topology. Particles in these regions are allowed to modify the local values of ϕ. At grid points where the implicit basis function for the particle would give a smaller value of ϕ (i.e. a particle is "poking" out of the zero level set), this smaller value is used to replace the value obtained from the time evolution of ϕ.

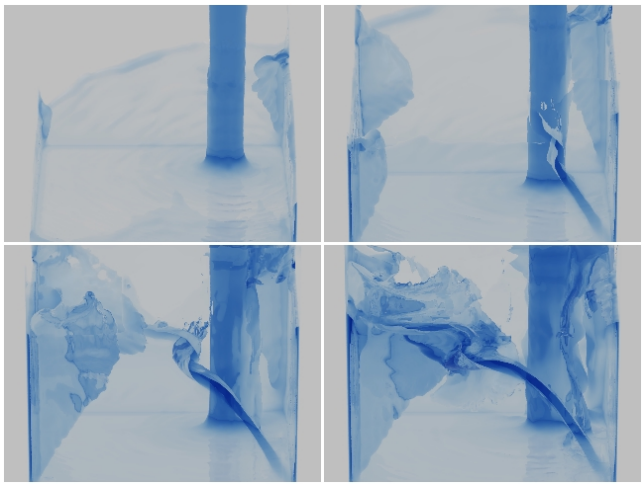

Figure 4: *Water pours into a container causing a complex surface to develop.*

Even with the tight coupling between the particles and the level set, some particles will escape the inside of the liquid layer since the grid is too coarse to represent them individually. These particles can be rendered directly as small liquid drops. In addition, these stray particles could be used as control particles to indicate the presence of fine spray or mist.

6. Updating the Velocity Field

We have a representation of the graphics environment and a way of tracking the surface of a volume of liquid. We can now apply (3.2) to the existing velocity field to advance it through an Euler-integration time step Δt. The equation is solved in three stages. First we compute Δt using the CFL condition (see Appendix A). Next, we update the convective component, i.e. $(\mathbf{u} \cdot \nabla)\mathbf{u}$, using a first order semi-Lagrangian method, as per Courant et al. [6] and by Stam [25]. We use the same formulation as Stam and refer readers to his description. Standard central differencing is then used on the viscous terms of (3.2) as described by Foster and Metaxas [11]. The results from this and the preceding calculation are added together to get an updated (though not mass conserving) velocity field for time $t+\Delta t$.

Semi-Lagrangian methods allow us to take large time steps without regard for the sometimes overly restrictive CFL condition [26]. Unfortunately, these large time steps come at the cost of added dissipation. This is visually acceptable for gases such as smoke where it appears realistic. For liquids however, mass dissipation ruins the visual effect. Therefore, even though we use a semi-Lagrangian method to update (3.2), the time step for evolving the particles and the level set still needs to be limited according to a plausible CFL condition. Updating the surface position isn't particularly expensive computationally, and so we alternate between a large time step for updating the Navier-Stokes equations and a series of small time steps (that add up to the large time step) for the particles and the level set. Our experience suggests that the velocity field time step can only be a few (around five) times bigger than that dictated by the usual CFL criterion. However, even this gives tremendous computational savings, since enforcing incompressibility (step V, discussed in section 8) is the most expensive part of the algorithm.

We caution the reader that using a particle only evolution with the semi-Lagrangian method introduces noise into the surface, and that using a level set only evolution with the semi-Lagrangian method gives noticeable volume loss. The key to making the semi-Lagrangian method work for liquids is the mixed Eulerian-Lagrangian formulation that uses *both* particles and level sets to evolve the surface position over time.

7. Boundary Conditions

When solving (3.2) within the grid, we need to specify pressure and velocity values for certain cells. We want stationary object cells to resist liquid motion and cells that represent the boundary between air and liquid to behave appropriately.

7.1.1 Non-liquid Cells

Cells in the grid that don't contain particles and aren't contained within the isosurface are either considered empty (open air) or are part of an object. If a cell is empty, its pressure is set to atmospheric pressure, and the velocity on each of its faces shared with another empty cell is set to zero. This assumes that air dynamics has a negligible effect. An object cell, on the other hand, can have velocities and pressures set using many different combinations to approximate liquid flowing into or out of the environment, or to approximate different object material properties. Foster and Metaxas [10] summarize and discuss methods to do this.

7.1.2 Liquid Surface

Other grid cells that require special attention are those that contain part of the $\phi(\mathbf{x})=0$ isocontour. Such cells represent what we know about the location of the liquid surface within the grid. The movement of the isocontour will determine how the surface evolves, but we need to set velocities on faces between empty and liquid cells so that normal and tangential stresses are zero. Intuitively, we need to make sure that the "air" doesn't mix with or inhibit the motion of the liquid, while allowing it to flow freely into empty cells. This is done by explicitly enforcing incompressibility within each cell that contains part of the liquid surface. Velocities adjacent to a liquid filled cell are left alone, whereas the others are set directly so (3.1) is satisfied for that cell. The pressure in a surface cell is set to atmospheric pressure.

8. Conservation of Mass

The velocity field generated after evolving the Navier-Stokes equations (steps III and IV) has rotation and convection components that are governed by (3.2) (excluding the pressure term). However, (3.1), conservation of mass, is only satisfied in surface cells where we have explicitly enforced it. The best we can do to preserve mass within our grid is to ensure that the incompressibility condition is satisfied for every grid cell (at least to some tolerance). Foster and Metaxas [11] achieved this using a technique called Successive Over Relaxation.

A more efficient method for enforcing incompressibility comes from solving the linear system of equations given by using the Laplacian operator to couple local pressure changes to the divergence in each cell. Specifically, this gives

$$\nabla^2 p = \rho \nabla \cdot \mathbf{u} / \Delta t \,, \qquad (8.1)$$

where $\nabla^2 p$ is the spatial variation (Laplacian) of the pressure and $\mathbf{u}$ is the velocity field obtained after solving (3.2). Applied at the center of a cell, (8.1) can be discretized as

$$\sum_{n=\{ijk\}} (p_{n+1} + p_{n-1}) - 6p = \rho \frac{\Delta \tau}{\Delta t} \sum_{n=\{ijk\}} (u_{n+1} - u_n), \qquad (8.2)$$

where $p_{n \pm 1}$ is the pressure from the cell ahead (+) or behind (-) in the n direction, and the u values are taken directly from the grid cell faces. Using (8.2), we form a linear system AP = b where P is the vector of unknown pressures needed to make the velocity field divergence free, b is the RHS of (8.2), and A has a regular but sparse structure. The diagonal coefficients of A, a_{ii}, are equal to the negative number of liquid cells adjacent to $cell_i$ (e.g., -6 for a fully "submerged" cell) while the off diagonal elements are simply $a_{ij}=a_{ji}=1$ for all liquid $cells_j$ adjacent to $cell_i$.

Conveniently, the system described above is symmetric and positive definite (as long as there is at least one surface cell as part of each volume). Static object and empty cells don't disrupt this structure. In that case pressure and velocity terms can disappear from both sides of (8.2), but the system remains symmetric. Because of this, it can be solved quickly and efficiently using a Preconditioned Conjugate Gradient (PCG) method. Further efficiency gains can be made by using an Incomplete Choleski preconditioner to accelerate convergence. There is a wealth of literature available regarding PCG techniques and we recommend any of the standard implementations, see Barret et al. [2] for some basic templates. Once the new pressures have been determined, the velocities in each cell are updated according to

$$u_{\{ijk\}}^{t+\Delta t} = u_{\{ijk\}} - \frac{\Delta t}{\rho \Delta \tau} (p_n - p_{n-1})$$

The resulting velocity field conserves mass (is divergence free) and satisfies the Navier-Stokes equations.

9. Moving Objects

Previous techniques proposed for liquid animation could deal with static objects that have roughly the same resolution as the grid, but they have difficulty dealing with moving objects. Unfortunately, the CFD literature has little to offer to help resolve the effects of moving objects on a liquid in terms of animation. There are sophisticated methods available for handling such interactions, but they typically require highly resolved computational grids or a grid mechanism that can adapt itself to the moving object surface. Neither approach is particularly well suited to animation because of the additional time complexity involved. Therefore, we propose the following method for handling interactions between moving objects and the liquid.

Consider an object (or part of an object) moving within a cell that contains liquid. There are two basic conditions that we want to enforce with respect to the computational grid, and an additional condition with respect to the surface tracking method. These are

1. Liquid should not flow into the object. At any point of contact, the relative velocity between the liquid and object along the object's surface normal should be greater than or equal to zero.
2. Tangential to the surface, the liquid should flow freely without interference.
3. Neither the particles nor the level set surface should pass through any part of the surface of the object.

The last of these is relatively straightforward. We know where the polygons that comprise the object surface are and in what direction they are moving. We simply move the particles so that they are always outside the surface of the object. As long as we accurately take account of the velocity field within the grid then the isocontour will remain in the correct position relative to the object.

To prevent liquid from flowing into the object we directly set the component of liquid velocity normal to the object. We know the object surface normal, $\mathbf{n_s}$, and can calculate the liquid velocity relative to that surface, $\mathbf{v_r}$, in a given cell. If $\mathbf{v_r}.\mathbf{n_s} < 0$ then liquid is flowing through the surface. In such cases we manipulate $\mathbf{u}$ in the cell so that $\mathbf{v_r}.\mathbf{n_s} = 0$ leaving the tangential ("slip") part of the velocity unchanged.

These velocities need to be applied without introducing visual artifacts into the flow. The following method solves for both normal and tangential velocity components. It's relatively intuitive, and it seems to work well in practice. The steps are

1. As a boundary condition, any cell within a solid object has its velocities set to that of the moving object.
2. The velocity field is updated using (3.2). No special consideration is given to cells containing an object, i.e. they are all allowed to change freely as if they contain liquid.
3. Each cell that intersects an object surface gets the component of the object velocity along its normal set explicitly as outlined above.
4. Cells internal to the object have their velocities set back to the object velocity.
5. During the mass conservation step (section 8) the velocity for any grid cell that intersects the object is held fixed.

The result of this approach is that liquid is both pushed along by an object while being allowed to flow freely around it, causing realistic-looking behavior in the mean time. Obviously it's only possible to accurately account for one polygon face per grid cell. Objects that are more detailed with respect to the grid can still be handled by determining an average object surface normal and velocity for each intersecting cell, but grid resolution remains the limiting factor.

10. Control

Animation is all about control. Having things behave according to some arbitrary aesthetic is the goal of most production software. The difficulty is in providing this level of control over a physics-based animation while still maintaining a realistic behavioral component. The nature of the governing equations of motion of liquids means that they will always swirl, mix, and splash unless the applied forces are identical everywhere. This necessarily limits the level of control that we can have over the final motion and comes with the territory of non-linear simulation.

Gates [13] has shown that mass conserving flow fields can be blended with calculated fields to get good non-dynamic results. The Navier-Stokes equations allow for the body force term, $\mathbf{g}$, to be manipulated directly [9] much like a traditional particle system. Forces aren't always a very intuitive way of getting motion that we want however. The moving object mechanism on the other hand, is well suited to this. Instead of moving polygons, we can explicitly set velocities anywhere in the grid by introducing "fake" surfaces (a single point even) that have normals and velocities pointing in the direction that we want the liquid to go. Setting the normal *and* tangential velocities in individual cells is also possible if it is done before the mass conservation calculation. This allows the solver to smooth out any lack of physical correctness in applied velocities before passing them into (3.2).

As a brief example, consider a set of 3D parametric space curves that define the desired path for the liquid to follow. We instance a set of points along each curve giving each point a parametric coordinate φ_p. A point's spatial position is then given simply by the curve definition, i.e. $\mathbf{x}_p=F(\varphi_p)$. The velocity of the point can then be described as

$$\mathbf{v}_p = C(t)\left(dF(\varphi_p)/d\varphi_p\right)$$

where $C(t)$ is a monotonic key framed scaling function. $C(t)$ is also used to update φ_p over time according to $d\varphi_p/dt = C(t)$. The "fake" surface normal of the point is then simply $\mathbf{n}_p = \mathbf{v}_p/|\mathbf{v}_p|$. By manipulating $\mathbf{x}_p$, $\mathbf{v}_p$, and $\mathbf{n}_p$ over time, we can "trap" small pockets of liquid and control them directly. The governing equations then make sure that neighboring liquid attempts to follow along.

This basic approach can be adapted to surfaces or even volumetric functions as long as they vary smoothly. While still not giving perfect direct control over the liquid motion, when combined with force fields it is good enough to make it a useful animation tool.

11. Results

The animation system described in the preceding sections was used to generate all of the examples in this paper. The basic Navier-Stokes solver and implicit surface are demonstrated by the container-filling example in figure 4. The combination of particles and level set make sure that the resulting surface stays smooth and behaves in a physically believable way. The splashing object examples in figures 1, 5 and 6 show close interaction between the liquid and moving objects. They also show how the hybrid surface can handle extreme splashing without either the particles or level set being apparent. The particles play a large role in both cases by allowing the liquid to "splash" at a higher resolution than would be possible with the level set alone. All of these images were rendered using a ray-tracing algorithm that marches through the implicit surface grid as outlined in section 5.

The final example, figure 7, makes use of just a spherical implicit function around each particle. It shows the interaction between a thick (high viscosity) liquid and a hand animated character. The character surface is sampled at each grid cell and the mechanisms described in section 9 take account of all the motion in the scene. This includes the character filling his mouth with mud. The mud is later ejected using a 3D space curve as a controller as outlined in section 10. The captions to each figure give the static grid size used during calculation along with computation times per frame (for motion, not rendering) on a PentiumII 500MHz.

12. Conclusion

We have presented a method for modeling and animating liquids that is a pragmatic compromise between the numerical care that needs to be taken when simulating non-linear physics and the interaction and control animators require. Where appropriate, we have drawn on techniques from computational fluid dynamics and combined them with recent computer graphics advances as well as new methods for free surface evolution and interaction between

moving objects and the liquid volume. The result is a technique that is very general, efficient, and offers flexible control mechanisms for specifying how the liquid should behave.

13. Acknowledgements

The work of the second author was supported in part by ONR N00014-97-1-0027.

A. Courant-Friedrichs-Levy (CFL) Condition

The CFL condition is a restriction on the size of the time step, Δt, that can be used together with a time-marching numerical simulation. It says that Δt must be less than the minimum time over which "something significant" can occur in the system for the simulation to remain numerically stable. The CFL condition depends both on the physical system being modeled as well as the specifics of the discretization method employed. In the context of the system described in this paper a good CFL condition is that a discrete element of liquid cannot "jump over" a cell in the computational grid, i.e. $\Delta t < \Delta\tau / |\mathbf{u}|$.

Note that the viscosity related terms also impose a CFL type restriction. This can be avoided by locally adjusting the magnitude of the viscosity in cells where the viscous terms would dictate the necessity for a smaller time step.

References

[1] Abbot, M. and Basco, D., "Computational Fluid Dynamics – An Introduction for Engineers", Longman, 1989.

[2] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. and van der Vorst, H., "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Society for Industrial and Applied Mathematics, 1993.

[3] Bloomenthal, J., Bajaj, C., Blinn, J., Cani-Gascual, M.-P., Rockwood, A., Wyvill, B. and Wyvill, G., "Introduction to Implicit Surfaces", Morgan Kaufmann Publishers Inc., San Francisco, 1997.

[4] Chen, J. and Lobo, N., "Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using the Navier-Stokes Equations", Graphical Models and Image Processing 57, 107-116 (1994).

[5] Chen, S., Johnson, D., Raad, P. and Fadda, D., "The Surface Marker and Micro Cell Method", Int. J. Numer. Methods in Fluids 25, 749-778 (1997).

[6] Courant, R., Issacson, E. and Rees, M., "On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences", Comm. Pure and Applied Math 5, 243-255 (1952).

[7] Desbrun, M. and Cani-Gascuel, M.P., "Active Implicit Surface for Animation", Graphics Interface 98, 143-150 (1998).

[8] Fedkiw, R., Aslam, T., Merriman, B. and Osher, S., "A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method)", J. Comput. Phys. 152, 457-492 (1999).

[9] Foster, N. and Metaxas, D., "Controlling Fluid Animation", Computer Graphics International 97, 178-188 (1997).

[10] Foster, N. and Metaxas, D., "Modeling the Motion of a Hot Turbulent Gas", ACM SIGGRAPH 97, 181-188 (1997).

[11] Foster, N. and Metaxas, D., "Realistic Animation of Liquids", Graphical Models and Image Processing 58, 471-483 (1996).

[12] Fournier, A. and Reeves, W.T., "A Simple Model of Ocean Waves", ACM SIGGRAPH 86, 75-84 (1986).

[13] Gates, W.F., "Interactive Flow Field Modeling for the Design and Control of Fluid Motion in Computer Animation", UBC CS Master's Thesis, 1994.

[14] Golub, G.H. and Van Loan, C.F., "Matrix Computations", The John Hopkins University Press, 1996.

[15] Harlow, F.H. and Welch, J.E., "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface", The Physics of Fluids 8, 2182-2189 (1965).

[16] Kang, M., Fedkiw, R. and Liu, X.-D., "A Boundary Condition Capturing Method For Multiphase Incompressible Flow", J. Sci. Comput. 15, 323-360 (2000).

[17] Kass, M. and Miller, G., "Rapid, Stable Fluid Dynamics for Computer Graphics", ACM SIGGRAPH 90, 49-57 (1990).

[18] Lorenson, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics 21, 163-169 (1987).

[19] Miller, G. and Pearce, A., "Globular Dynamics: A Connected Particle System for Animating Viscous Fluids", Computers and Graphics 13, 305-309 (1989).

[20] O'Brien, J. and Hodgins, J., "Dynamic Simulation of Splashing Fluids", Computer Animation 95, 198-205 (1995).

[21] Osher, S. and Sethian, J.A., "Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations", J. Comput. Phys. 79, 12-49 (1988).

[22] Peachy, D., "Modeling Waves and Surf", ACM SIGGRAPH 86, 65-74 (1986).

[23] Schachter, B., "Long Crested Wave Models", Computer Graphics and Image Processing 12, 187-201 (1980).

[24] Sethian, J.A. "Level Set Methods and Fast Marching Methods", Cambridge University Press, Cambridge 1999.

[25] Stam, J., "Stable Fluids", ACM SIGGRAPH 99, 121-128 (1999).

[26] Staniforth, A. and Cote, J., "Semi-Lagrangian Integration Schemes for Atmospheric Models – A Review", Monthly Weather Review 119, 2206-2223 (1991).

[27] Terzopoulos, D., Platt, J. and Fleischer, K., "Heating and Melting Deformable Models (From Goop to Glop)", Graphics Interface 89, 219-226 (1995).

[28] Yngve, G., O'Brien, J. and Hodgins, J., "Animating Explosions", ACM SIGGRAPH 00, 29-36 (2000).

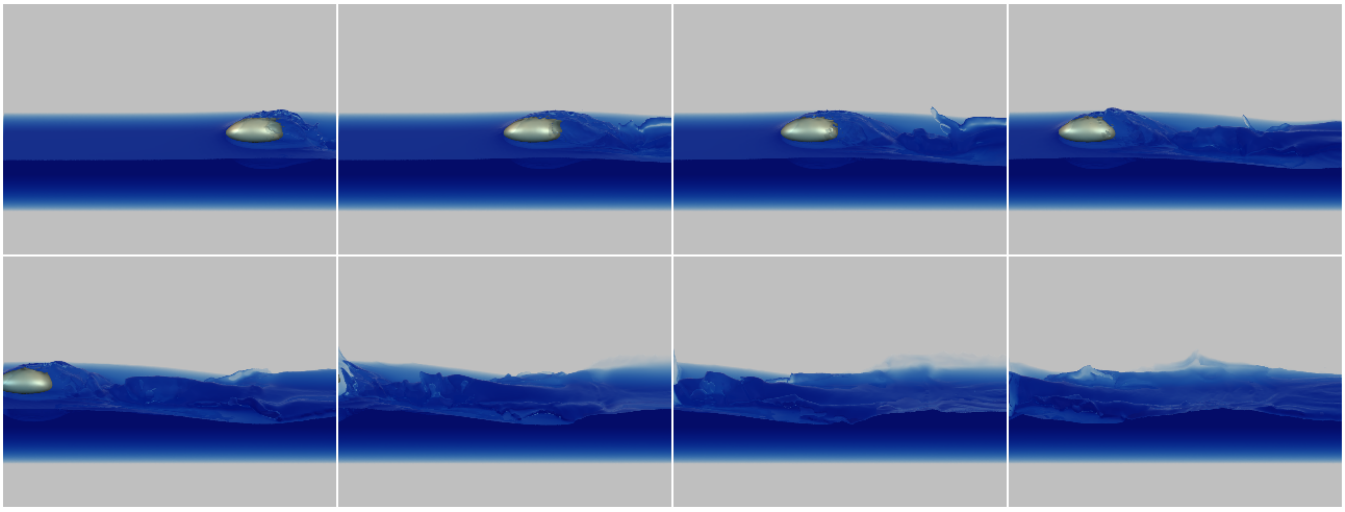

Figure 5: *An ellipsoid slips along through shallow water. The combination of particle and level set tracking allows water to flow over the object without any visual loss of volume. The environment for this example was 250x75x90 cells. It took approximately seven minutes to calculate the liquid motion (including surface evolution) per frame.*

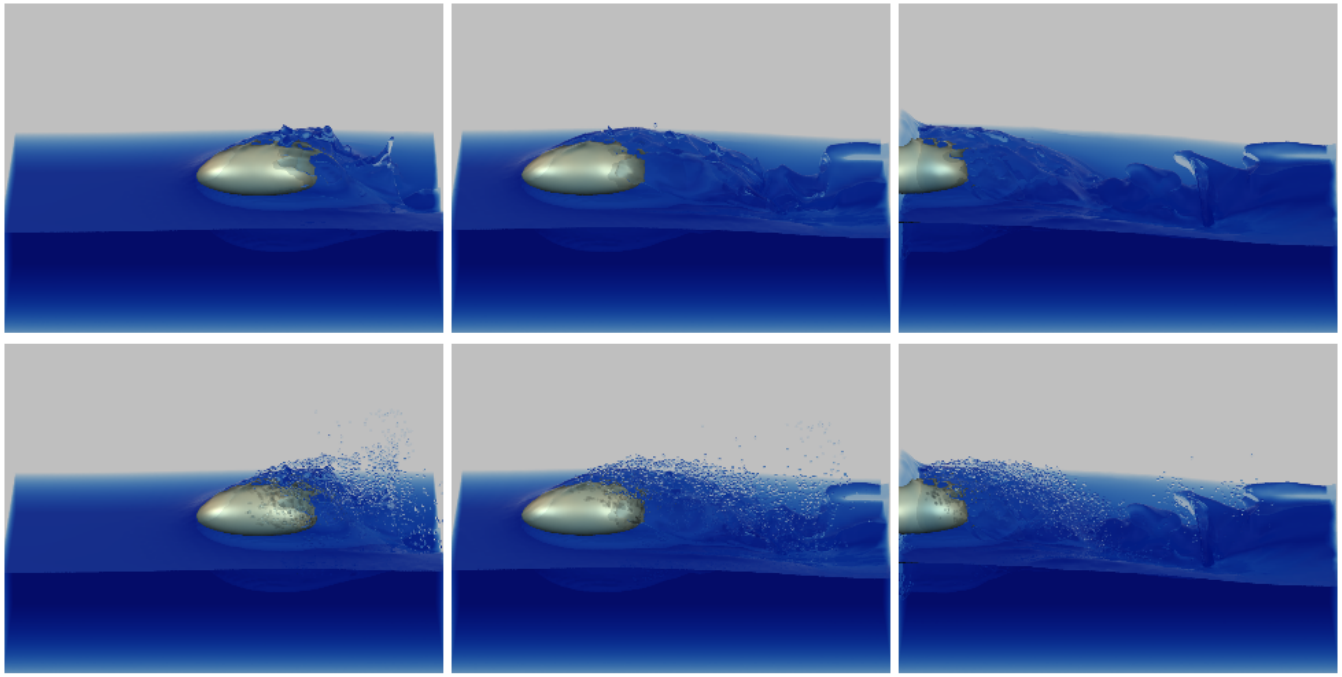

Figure 6: *A close up of the ellipsoid from figure 5 showing the implicit surface derived from combining the particle basis functions and level set (top), and with the addition of the freely splashing particles raytraced as small spheres (bottom). The environment for this example was 150x75x90 cells. Calculation times were approximately four minutes per frame.*

Figure 7: *A fully articulated animated character interacts with viscous mud. The environment surrounding the character is 150x200x150 cells. That resolution is sufficient to accurately model the character filling his mouth with mud. A 3D control curve is used to eject (spit) the mouthful of mud later in the sequence. This example runs at three minutes per frame.*