Animation and Rendering of Complex Water Surfaces

Douglas Enright
Stanford University
Industrial Light & Magic
enright@stanford.edu

Stephen Marschner
Stanford University
srm@graphics.stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

Abstract

We present a new method for the animation and rendering of *photorealistic* water effects. Our method is designed to produce visually plausible three dimensional effects, for example the pouring of water into a glass (see figure 1) and the breaking of an ocean wave, in a manner which can be used in a computer animation environment. In order to better obtain photorealism in the behavior of the simulated water surface, we introduce a new "thickened" front tracking technique to accurately represent the water surface and a new velocity extrapolation method to move the surface in a smooth, water-like manner. The velocity extrapolation method allows us to provide a degree of control to the surface motion, e.g. to generate a wind-blown look or to force the water to settle quickly. To ensure that the photorealism of the simulation carries over to the final images, we have integrated our method with an advanced physically based rendering system.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing;

Keywords: computational fluid dynamics, implicit surfaces, natural phenomena, physically based animation, rendering, volume rendering

1 Introduction

Water surrounds us in our everyday lives. Given the ubiquity of water and our constant interaction with it, the animation and rendering of water poses one of the greatest challenges in computer graphics. The difficulty of this challenge was underscored recently through the use of water effects in a variety of motion pictures including the recent feature film "Shrek" where water, mud, beer and milk effects were seen. In response to a question concerning what was the single hardest shot in "Shrek", DreamWorks SKG principal and producer of "Shrek", Jeffrey Katzenberg, stated, *It's the pouring of milk into a glass.* [Hiltzik and Pham 2001].

The above quote illustrates the need for *photorealistic* simulation and rendering of water (and other liquids such as milk), especially in the case of complex, three dimensional behavior as seen when water is poured into a glass as in figure 1. A key to achieving this goal is the visually accurate treatment of the surface separating the water from the air. The behavior of this surface provides the visual impression that water is being seen. If the numerical simulation method used to model this surface is not robust enough to capture the essence of water, then the effect is ruined for the viewer. A variety of new techniques for treatment of the surface are proposed in this paper in order to provide visually pleasing motion and photorealistic results.

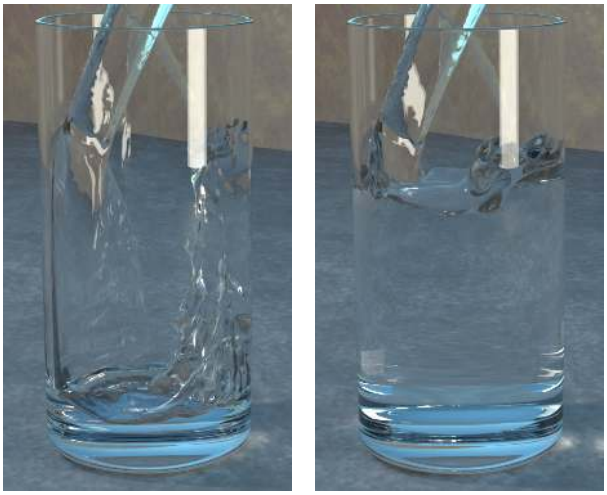


Figure 1: **Water being poured into a glass (55x120x55 grid cells).**

We propose a new "thickened" front tracking approach to modeling the surface, called the "particle level set method". It is a hybrid surface tracking method that uses massless marker particles combined with a dynamic implicit surface. This method was entirely inspired by the hybrid liquid volume model proposed in [Foster and Fedkiw 2001], but exhibits significantly more realistic surface behavior. This effect is achieved by focusing on modeling the surface as opposed to the liquid volume as was done by [Foster and Fedkiw 2001]. This shift in philosophy away from volume modeling and towards surface modeling, is the key idea behind our new techniques, resulting in better photorealistic behavior of the water surface.

We propose a new treatment of the velocity at the surface in order to obtain more visually realistic water surface behavior. The motion of both the massless marker particles and the implicit function representing the surface is dependent upon the velocities contained on the underlying computational mesh. By extrapolating velocities across the water surface and into the region occupied by the air, we obtain more accurate and better visual results. In the limit as the computational grid is refined, the resulting surface condition is identical to the traditional approach of making the velocity divergence free, but it gives more visually appealing and physically plausible results on coarse grids. Furthermore, this velocity extrapolation procedure allows us to add a degree of control to the behavior of the water surface. We can add dampening and/or churning effects forcing it to quiet down or splash up faster than would be

allowed by a straightforward physical simulation.

Our new advances can be easily incorporated into a pre-existing Navier-Stokes solver for water. In fact, we solve the Navier-Stokes equations for liquid water along the lines of [Foster and Fedkiw 2001], in particular using a semi-Lagrangian "stable fluid" approach introduced to the community by Stam [Stam 1999]. Our approach preserves as much of the realistic behavior of the water as possible, while at the same time providing a degree of control necessary for its use in a computer animation environment.

Photorealistic rendering is necessary in order to complete the computational illusion of real water. In some ways water is an easy material to render, because unlike many common materials its optical properties are well understood and are easy to describe. In all but the largest-scale scenes, surface tension prevents water surfaces from exhibiting the microscopic features that make reflection from many other materials so complicated. However, water invariably creates situations in which objects are illuminated through complex refracting surfaces, which means that the light transport problem that is so easy to state is difficult to solve. Most widely used rendering algorithms disregard this sort of illumination or handle it using simple approximations, but because water and its illumination effects are so familiar these approaches fail to achieve realism. There are several rendering algorithms that can properly account for all transport paths, including the illumination through the water surface; some examples are path tracing [Kajiya 1986], bidirectional path tracing [Heckbert 1990; Lafortune and Willems 1993; Veach and Guibas 1994], Metropolis light transport [Veach and Guibas 1997], and photon mapping [Jensen 1995]. In our renderings of clear water for this paper we have chosen photon mapping because it is simple and it makes it easy to avoid the distracting noise that often arises in pure path sampling algorithms from illumination through refracting surfaces.

2 Previous Work

Early (and continuing) work by the graphics community into the modeling of water phenomenon focused on reduced model representations of the water surface, ranging from Fourier synthesis methods [Masten et al. 1987] to parametric representations of the water surface [Schachter 1980; Fournier and Reeves 1986; Peachey 1986; Ts'o and Barsky 1987]. The last three references are notable in the way they attempt to model realistic wave behavior including the change in wave behavior as a function of the depth of the water. Fairly realistic two dimensional wave scenery can be developed using these methods including the *illusion* of breaking waves, but ultimately they are all constrained by the sinusoidal modeling assumption present in each of them. They are unable to easily deal with complex three dimensional behaviors such as flow around objects and dynamically changing boundaries. A summary of the above methods and their application to modeling and rendering of ocean wave environments can be found in [Tessendorf 2001].

In order to obtain water models which could potentially be used in a dynamic animation environment, researchers turned towards using two dimensional approximations to the full 3D Navier-Stokes equations. [Kass and Miller 1990] use a linearized form of the 2D shallow water equations to obtain a height field representation of the water surface. A pressure defined height field formulation was used by [Chen and Lobo 1994] in fluid simulations with moving obstacles. [O'Brien and Hodgins 1995] used a height model combined with a particle system in order to simulate splashing liquids. The use of a height field gives a three dimensional look to a two dimensional flow calculation, but it constrains the surface to be a function, i.e. the surface passes the vertical line test where for each (x,y) position there is at most one z value. The surface of a crashing wave or of water being poured into a glass does not satisfy the vertical line test. Use of particle systems permits the surface to become multivalued. A viscous spring particle representation of a liquid has been proposed by [Miller and Pearce 1989]. An alternative molecular dynamics approach to the simulation of particles in the liquid phase has been developed by [Terzopoulos et al. 1989]. Particle methods, while quite versatile, can pose difficulties when trying to reconstruct a smooth water surface from the locations of the particles alone.

The simulation of complex water effects using the full 3D Navier-Stokes equations has been based upon the large amount of research done by the computational fluid dynamics community over the past 50 years. [Foster and Metaxas 1996] utilized the work of [Harlow and Welch 1965] in developing a 3D Navier-Stokes methodology for the realistic animation of liquids. Further CFD enhancements to the traditional marker and cell method of Harlow and Welch which allow one to place particles only near the surface can be found in [Chen et al. 1997]. A semi-Lagrangian "stable fluids" treatment of the convection portion of the Navier-Stokes equations was introduced to the computer graphics community by [Stam 1999] in order to allow the use of significantly larger time steps without hindering stability. [Foster and Fedkiw 2001] made significant contributions to the simulation and control of three dimensional fluid simulations through the introduction of a hybrid liquid volume model combining implicit surfaces and massless marker particles; the formulation of plausible boundary conditions for moving objects in a liquid; the use of an efficient iterative method to solve for the pressure; and a time step subcycling scheme for the particle and implicit surface evolution equations in order to reduce the amount of visual error inherent to the large semi-Lagrangian "stable fluid" time step used for time evolving the fluid velocity and the pressure. The combination of all of the above advances in 3D fluid simulation technology along with ever increasing computational resources has set the stage for the inclusion of fully 3D fluid animation tools in a production environment.

Most work on simulating water at small scales has not specifically addressed rendering and has not used methods that correctly account for all significant light transport paths. Research on the rendering of illumination through water [Watt 1990; Nishita and Nakamae 1994] has used methods based on processing each polygon of a mesh that represents a fairly smooth water surface, so these methods cannot be used for the very complex implicit surfaces that result from our simulations. For the case of 2D wave fields in the open ocean, approaches motivated by physical correctness have produced excellent results [Premoze and Ashikhmin 2000; Tessendorf 2001].

3 Simulation Method

3.1 Motivation

[Foster and Fedkiw 2001] chose to define the liquid volume being simulated as one side of an isocontour of an implicit function, ϕ. The surface of the water was defined by the $\phi = 0$ isocontour with $\phi \leq 0$ representing the water and $\phi > 0$ representing the air. By using an implicit function representation of the liquid volume, they obtained a smooth, temporally coherent liquid surface. They rejected the use of particles alone to represent the liquid surface because it is difficult to a calculate a visually desirable smooth liquid surface from the discrete particles alone. The implicit surface was dynamically evolved in space and time according to the underlying liquid velocity $\vec{u}$. As shown in [Osher and Sethian 1988], the appropriate equation to do this is

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

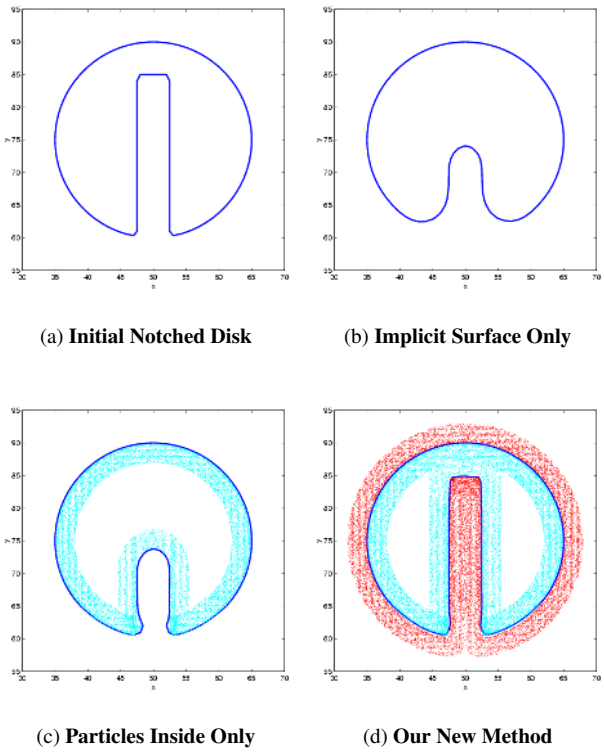
where ϕ_t is the partial derivative of ϕ with respect to time and $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ is the gradient operator.

An implicit function only approach to modeling the surface will not yield realistic surface behavior due to an excessive amount of volume loss on coarse grids. A seminal advance of [Foster and Fedkiw 2001] in creating realistic liquids for computer animation is the hybridization of the visually pleasing smooth implicit function modeling of the liquid volume with particles that can maintain the liquid volume on coarse grids. The inclusion of particles provides a way for capturing the liveliness of a real liquid with spray and splashing effects. Curvature was used as an indicator for allowing particles to influence the implicit surface representation of the water. This is a natural choice since small droplets of water have very high curvature and dynamic implicit surfaces have difficulty resolving features with sharp corners.

Figure 2(a) shows a notched disk that we rotate for one rigid body rotation about the point (50,50). The inside of the disk can be thought of as a volume of liquid. Figure 2(b) shows the result of using an implicit surface only approach (after one rotation) where both the higher and lower corners of the disk are erroneously shaved off causing both loss of visual features and an artificially viscous look for the liquid. This numerical result was obtained using a highly accurate fifth order WENO discretization of equation 1 (see e.g. [Jiang and Peng 2000; Osher and Fedkiw 2002]). For the sake of comparison, we note that [Sethian 1999] only proposes second order accurate methods for discretizing this equation. Figure 2(c) shows the result obtained with our implementation of the method from [Foster and Fedkiw 2001]. The particles inside the disk do not allow the implicit surface to cross over them and help to preserve the two corners near the bottom. However, there is little they can do to stop the implicit surface from drifting away from them near the top corners. This represents loss of air or bubbles as the method erroneously gains liquid volume. This is not desirable since many complex water motions such as wave phenomenon are due in part to differing heights of water columns adjacent to each other. Loss of air in a water column reduces the pressure forces from neighboring columns destroying many of the dynamic splashing effects as well as the overall visually stimulating liveliness of the liquid.

While the hybrid liquid volume model of Foster and Fedkiw attempts to maintain the volume of the liquid accurately, it fails to model the air or more generally the opposite side of the liquid surface. We shift the focus away from maintaining a liquid volume towards maintaining the liquid surface itself. An immediate advantage of this approach is that it leads to symmetry in particle placement. We place particles on both sides of the surface and use them to maintain an accurate representation of the surface itself regardless of what may be on one side or the other. The particles are not meant to track volume, they are intended to correct errors in the surface representation by the implicit function. In [Enright et al. 2002] we showed that this surface only approach leads to the most accurate 3D results for surface tracking ever achieved (in both CFD and CG). This was done for analytical "test" velocity fields. Figure 2(d) shows that this new method correctly computes the rigid body rotation for the notched disk preserving both the water and the air volumes so that more realistic water motion can be obtained.

In this current paper, we couple this new method to real velocity fields and fluid dynamics calculations for the first time. Representative results of this new method can be seen in figure 3. A ball is thrown into a tank of water with the same tank geometry, grid spacing and ball speed as seen in figure 4 (courtesy of [Foster and Fedkiw 2001]). The resulting splash after the ball impacts the surface of the water is dramatically different between the two figures. Our new method produces the well formed, thin sheet one would visually expect. Note that the distorted look of the ball in our figure is due to the correct calculation of the refraction of light when it passes through the surface of the water. To give an indication of the additional computational cost incurred using our new simulation method, figure 3 took about 11 minutes per frame, figure 4 took about 7 minutes per frame and a level set only solution takes about 3 minutes per frame. The rendering time for figure 3 was approximately 15 minutes per frame.



(a) **Initial Notched Disk** (b) **Implicit Surface Only**

(c) **Particles Inside Only** (d) **Our New Method**

Figure 2: **Rigid Body Rotation Test**

3.2 Particle Level Set Method

3.2.1 Initialization of Particles

Two sets of particles are randomly placed in a "thickened" surface region (we use three grid cells on each side of the surface) with *positive* particles in the $\phi > 0$ region and *negative* particles in the $\phi \leq 0$ region. There is no need to place particles far away from the surface since the sign of the level set function readily identifies these regions gaining large computational savings. The number of particles placed in each cell is an adjustable parameter that can be used to control the amount of resolution, e.g. we use 64 particles per cell for most of our examples. Each particle possesses a radius, r_p, which is constrained to take a minimum and maximum value based upon the size of the underlying computational cells used in the simulation. A minimum radius of $.1\min(\Delta x, \Delta y, \Delta z)$ and maximum radius of $.5\min(\Delta x, \Delta y, \Delta z)$ appear to work well. The radius of a particle changes dynamically throughout the simulation, since a particle's location relative to the surface changes. The radius is set according to:

$$r_p = \begin{cases} r_{max} & \text{if } s_p\phi(\vec{x}_p) > r_{max} \\ s_p\phi(\vec{x}_p) & \text{if } r_{min} \leq s_p\phi(\vec{x}_p) \leq r_{max} \\ r_{min} & \text{if } s_p\phi(\vec{x}_p) < r_{min} \end{cases}, \qquad (2)$$

where s_p is the sign of the particle (+1 for positive particles and -1 for negative particles). This radius adjustment keeps the boundary of the spherical particle tangent to the surface whenever possible. This fact combined with the overlapping nature of the particle

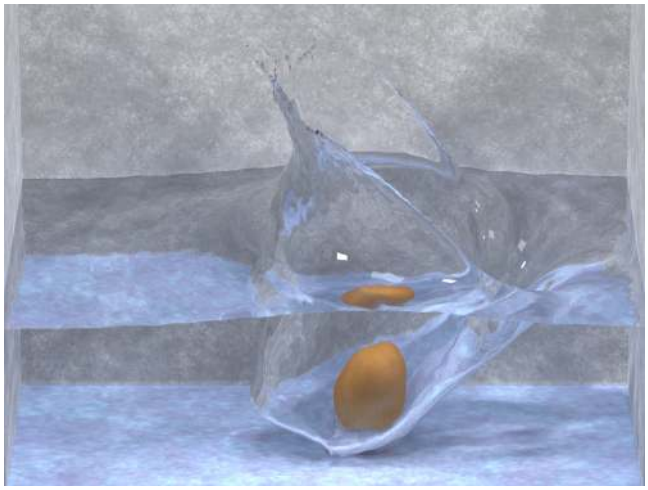


Figure 3: **Our New Method (140x110x90 grid cells).**

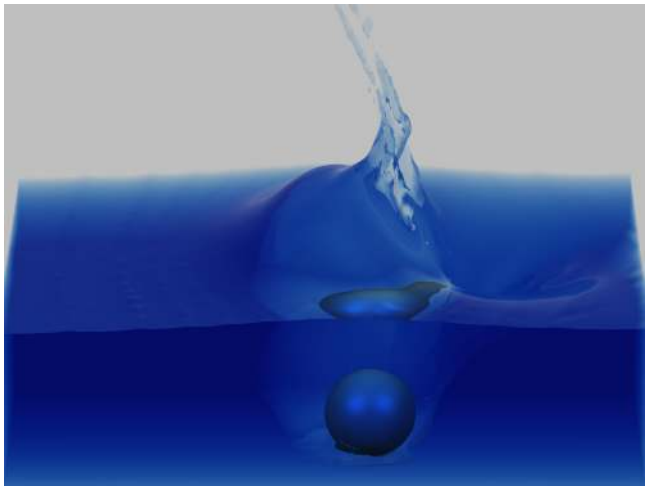


Figure 4: **Foster and Fedkiw 2001 (140x110x90 grid cells).**

spheres allows for an enhanced reconstruction capability of the liquid surface.

3.2.2 Time Integration

The marker particles and the implicit function are separately integrated forward in time using a forward Euler time integration scheme. The implicit function is integrated forward using equation 1, while the particles are passively advected with the flow using $d\vec{x}_p/dt = \vec{u}_p$, where $\vec{u}_p$ is the fluid velocity interpolated to the particle position $\vec{x}_p$.

3.2.3 Error Correction of the Implicit Surface

Identification of Error: The main role of the particles is to detect when the implicit surface has suffered inaccuracies due to the coarseness of the computational grid in regions with sharp features. Particles that are on the wrong side of the interface by more than their radius, as determined by a locally interpolated value of ϕ at the particle position $\vec{x}_p$, are considered to have *escaped* their side of the interface. This indicates errors in the implicit surface representation. In smooth, well resolved regions of the interface, our dynamic implicit surface is highly accurate and particles do not drift a non-trivial distance across the interface.

Quantification of Error: We associate a spherical implicit function , designated ϕ_p, with each particle p whose size is determined by the particle radius, i.e.

$$\phi_p(\vec{x}) = s_p(r_p - |\vec{x} - \vec{x}_p|). \tag{3}$$

Any difference in ϕ from ϕ_p indicates errors in the implicit function representation of the surface. That is, the implicit version of the surface and the particle version of the surface disagree.

Error Correction: We use escaped positive particles to rebuild the $\phi > 0$ region and escaped negative particles to rebuild the $\phi \leq 0$ region as defined by the implicit function. The reconstruction of the implicit surface occurs locally within the cell that each escaped particle currently occupies. Using equation 3, the ϕ_p values of escaped particles are calculated for the eight grid points on the boundary of the cell containing the particle. This value is compared to the current value of ϕ for each grid point and we take the smaller value (in magnitude) which is the value closest to the $\phi = 0$ isocontour defining the surface. We do this for all escaped positive and escaped negative particles. The result is an improved representation of the surface of the liquid.

3.2.4 When To Apply Error Correction

We apply the error correction method discussed above after any computational step in which ϕ has been modified in some way. This occurs when ϕ is integrated forward in time and when the implicit function is smoothed to obtain a visually pleasing surface. We smooth the implicit surface with an equation of the form

$$\phi_\tau = -S(\phi_{\tau=0})(|\nabla\phi| - 1), \tag{4}$$

where τ is a fictitious time and $S(\phi)$ is a smoothed signed distance function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + (\Delta x)^2}}. \tag{5}$$

More details on this are given in [Foster and Fedkiw 2001].

3.2.5 Particle Reseeding

In complex flows, a liquid interface can be stretched and torn in a dynamic fashion. The use of only an initial seeding of particles will not capture these effects well, as regions will form that lack a sufficient number of particles to adequately perform the error correction step. Periodically, e.g. every 20 frames, we randomly reseed particles about the "thickened" interface to avoid this dilemma. This is done by randomly placing particles near the interface, and then using geometric information contained within the implicit function (e.g. the direction of the shortest possible path to the surface is given by $\vec{N} = \nabla\phi/|\nabla\phi|$) to move the particles to their respective domains, $\phi > 0$ or $\phi \leq 0$. The goal of this reseeding step is to preserve the initial particle resolution of the interface, e.g. 64 particles per cell. Thus, if a given cell has too few or too many particles, some can be added or deleted respectively.

3.2.6 A Note on Alternative Methods

If we felt that preserving the volume of the fluid was absolutely necessary in order to obtain visually pleasing fluid behavior, we would have chosen to use a volume of fluid (VOF) [Hirt and Nichols 1981] representation of the fluid. Although VOF methods explicitly conserve volume, they produce visually disturbing artifacts allowing thin liquid sheets to artificially break up and form "blobbies" and

"flotsam" of liquid. Also, a visually desirable smooth fluid interface is difficult to construct when using these methods.

Another alternative is to explicitly discretize the free surface with particles and maintain a connectivity list between particles, see e.g. [Unverdi and Tryggvason 1992]. This connectivity list is difficult to maintain when parts of the free surface break apart or merge together as is often seen in complex flows of water and other liquids. Our approach avoids the especially difficult issues associated with maintaining particle connectivity information.

3.3 Velocities at the Surface

Although the Navier-Stokes equations can be used to find the velocity within the liquid volume, boundary conditions are needed for the velocity on the air side near the free surface. These boundary condition velocities are used in updating the Navier-Stokes equations, moving the surface, and moving the particles placed near the surface. The velocity at the free surface of the water can be determined through the usual enforcement of the conservation of mass (volume) constraint, $\nabla \cdot \vec{u} = 0$, where $\vec{u} = (u, v, w)$ is the velocity of the liquid. This equation allows us to determine the velocities on all the faces of computational cells that contain the $\phi = 0$ isocontour. Unfortunately, the procedure for doing this is not unique when more than one face of a cell needs a boundary condition velocity. A variety of methods have been proposed, e.g. see [Chen et al. 1995] and [Foster and Metaxas 1996].

We propose a different approach altogether, the extrapolation of the velocity across the surface into the surrounding air. As the computational grid is refined, this method is equivalent to the usual method, but it gives a smoother and more visually pleasing motion of the surface on coarser (practical sized) grids. We extrapolate the velocity out a few grid cells into the air, obtaining boundary condition velocities in a band of cells on the air side of the surface. This allows us to use higher order accurate methods and obtain better results when moving the implicit surface using equation 1 and also provides velocities for updating the position of particles on the air side of the surface. Velocity extrapolation also assists in the implementation of the semi-Lagrangian "stable fluid" method, since there are times when characteristics generated by this approach look back across the interface (a number of cells) into the air region for valid velocities.

3.3.1 Extrapolation Method

The equation modeling this extrapolation for the x component of the velocity, u, is given by

$$\frac{\partial u}{\partial \tau} = -\vec{N} \cdot \nabla u, \tag{6}$$

where $\vec{N} = (n_x, n_y, n_z)$ is a unit vector perpendicular to the implicit surface and τ is fictitious time. A similar equation holds for the v and w components of velocity field. Since we use an implicit surface to describe the fluid, $\vec{N} = \nabla\phi / |\nabla\phi|$. Fast methods exist for solving this equation in $O(n \log n)$ time, where n is the number of grid points that one needs to extrapolate over, in our case a five grid cell thick band on the air side of the interface. The fast method is based upon the observation that information in equation 6 propagates in only one direction away from the surface. This implies that we do not have to revisit previously computed values of $\vec{u}_{ext}$ (the extrapolated velocity) if we perform the calculation in the correct order. The order is determined by the value of ϕ allowing us to do an $O(n \log n)$ sorting of the points before beginning the calculation. The value of u itself is determined by enforcing the condition at steady state, namely $\nabla\phi \cdot \nabla u = 0$ where the derivatives are determined using previously calculated values of ϕ and u. From this scalar equation, a new value of u can be determined, and then we proceed to the next point corresponding to the next smallest value of ϕ, etc. Further details of this method are discussed in [Adalsteinsson and Sethian 1999].

3.3.2 Velocity Advection

The momentum portion of the Navier-Stokes equations is:

$$\vec{u}_t = -\vec{u} \cdot \nabla\vec{u} + \nu \nabla \cdot (\nabla\vec{u}) - \frac{1}{\rho}\nabla p + \vec{g}, \tag{7}$$

where ν is the kinematic viscosity of the fluid, ρ is the density of the fluid, p is the pressure, and $\vec{g}$ is an externally applied gravity field. We use the semi-Lagrangian "stable fluids" method [Stam 1999] to update the convective portion of this equation, i.e. the $\vec{u} \cdot \nabla\vec{u}$ term. This method calculates the first term on the left hand side of equation 7 by following the fluid characteristics backwards in time to determine from which computational cell the volume of fluid came, and then taking an average of the appropriate velocities there. This allows one to stably take much larger time steps than would be allowed using other time advancement schemes for velocity advection. A consequence of the now allowed large semi-Lagrangian time step is that near the surface, we might look across the interface as many as a few grid cells into the air region to find velocities. In a standard approach, valid velocities are not defined in this region. However, our velocity extrapolation technique easily handles this case ensuring that physically plausible velocities exist a few grid cells into the air region. In fact, we extrapolate the velocity the maximum distance from the surface that would be allowed during a semi-Lagrangian "stable fluids" time step guaranteeing that a smooth, physically plausible and visually appealing velocity can be found there.

3.3.3 Control

Our velocity extrapolation method enables us to apply a newly devised method for controlling the nature of the surface motion. This is done simply by modifying the extrapolated velocities on the air side of the surface. For example, to model wind-blown water as a result of air drag, we take a convex combination of the extrapolated velocities with a pre-determined wind velocity field

$$\vec{u} = (1 - \alpha)\vec{u}_{ext} + \alpha\vec{u}_{wind}, \tag{8}$$

where $\vec{u}_{ext}$ is the extrapolated velocity, $\vec{u}_{wind}$ a desired air-like velocity, and $0 \leq \alpha \leq 1$ the mixing constant. This can be applied throughout the surface or only locally in select portions of the computational domain as desired. Note that setting $\vec{u}_{wind} = 0$ forces churning water to settle down faster with the fastest settling resulting from $\alpha = 1$. All of the figures shown in the paper used $\alpha = 0$, but we demonstrate how α can be used to force a poured glass of water to settle more quickly in the accompanying video.

3.4 Summary

We divide up the computational domain into voxels with the components of $\vec{u}$ stored on the appropriate faces and p, ρ and ϕ stored at the center of each cell. This arrangement of computational variables is the classic staggered MAC-style arrangement [Harlow and Welch 1965]. The density of a given cell is given by the value of ϕ at the center of the cell. The evolution of $\vec{u}$, p, ρ and ϕ in a given time step is performed in a series of three steps as outlined below:

1. The current surface velocity is smoothly extrapolated across the surface into the air region as discussed in section 3.3.1. Appropriate control behavior modifications to the velocity field are made.

2. The water surface and marker particles are integrated forward in time via an explicit time step subcycling method with the appropriate corrections to ϕ as described in section 3.2.3.

3. The velocity field is updated with equation 7 using the updated values for ρ. This is done by first using the semi-Lagrangian "stable fluid" method to find an estimate for the velocity. This estimate is further augmented by the viscous and forcing terms. The spatial derivatives used in calculating these terms are calculated using a standard centered second order accurate finite difference scheme. Then a system of linear equations is assembled and solved for the pressure in order to make this intermediate velocity field divergence free. The newly calculated pressure is applied as a correction to the intermediate velocity in order to fully update the water's velocity field. Interaction of the liquid with objects, walls, etc. is treated here as well. For this step, we follow exactly the method of [Foster and Fedkiw 2001] and refer the reader there for more details.

This sequence of steps is repeated until a user defined stopping point is reached. The time step for each iteration of the above steps is determined using the water's velocity to calculate an appropriate CFL condition which is approximately five times larger than the traditional CFL condition used in fluid simulations (the semi-Lagrangian "stable fluids" method allows this). Also, any viscosity related CFL restrictions are locally dealt with by reducing the viscosity in the offending cells in order to allow for our larger time step.

3.5 Rendering

Our results are rendered using a physically based Monte Carlo ray tracer capable of handling all types of illumination using photon maps and irradiance caching [Jensen 2001]. To integrate our simulation system with the renderer we implemented a geometry primitive that intersects rays with the implicit surface directly by solving for the root of the signed distance along the ray. Depending on the accuracy required by a particular scene we use either a trilinear or a tricubic filter [Marschner and Lobb 1994] to reconstruct ϕ. The normal is computed using trilinearly interpolated central differences for the trilinear surface, or simply using the derivative of the reconstructed tricubic surface.

The properties of ϕ have two advantages in the rendering context. First, intersecting a ray with the surface is much more efficient than it would be for an isosurface of a general function. The signed distance function provides a lower bound on the distance to the intersection along the ray, allowing us to take large steps when the current point is far from the surface. Second, it is easy to provide for motion blur in the standard distribution ray tracing framework. To compute the surface at an intermediate time between two frames we simply interpolate between the two signed distance volumes and use the same intersection algorithm unchanged. For the small motions that occur between frames the special properties of ϕ are not significantly compromised.

4 Results

4.1 Pouring Water Into A Glass

Figures 1 and 7 show the high degree of complexity in the water surface. Note the liveliness of the surface of the water when the water is initially being poured. The ability to maintain the visually pleasing thin sheets of water during the turbulent mixing phase is a consequence of our new method. We do not loose any of the fine detail with regards to the air pockets formed, since we model both sides of the water surface. Even though the calculation was performed on a Cartesian computational grid, the glass was shaped as a cylinder on the grid, with the grid points outside the cylinder treated as an object which does not permit the fluid to interpenetrate it. The glass was modeled as smooth and clear in order to highlight the action of the water being poured into the glass. The computational cost was approximately 8 minutes per frame.

To our knowledge, the only other complex, three dimensional simulation of a liquid being poured into a glass for computer animation is from the Gingerbread Man torture scene in the feature film "Shrek". Milk lacks the transparency of water making it difficult to clearly view the dynamic behavior of the milk surface. Also, the modeling of a thick polygonal glass with a rough surface does not provide a clear view of the milk making a direct comparison difficult.

The scene used to render our result contains just a simple cylindrical glass, the simulated water surface, and a few texture mapped polygons for the background. Illumination comes from a physically based sky model and two area sources. Because water only reflects and refracts other objects, the scene including the sky is very important to the appearance. The glass and the water together create three dielectric interfaces: glass-air, water-air, and air-water, so the inside surface of the glass has two sets of material properties depending on whether it is inside or outside the implicitly defined surface (which is easy to determine by checking the sign of ϕ). The illumination in the shadow of the glass is provided by a photon map, and illumination from the sky on diffuse surfaces in the scene is computed using Monte Carlo integration. We also note that milk would not present as difficult a global illumination problem due to its lack of transparency. Motion blur was included for these renderings. The rendering time for the glass was approximately 15 minutes per frame.

4.2 Breaking Wave

As a second example, we have performed a breaking wave calculation in a numerical wave tank as shown in figure 8. To begin to model this phenomenon, we needed to chose an initial condition for the wave. We chose to use the theoretical solution of a solitary wave of finite amplitude propagating without shape change [Radovitzky and Ortiz 1998]. The initial velocities u and v in the x and y directions respectively and surface height η are given by:

$$u = \sqrt{gd}\frac{H}{d}\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right] \quad (9)$$

$$v = \sqrt{3gd}\left(\frac{H}{d}\right)^{3/2}\frac{y}{d}\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right]\tanh\left[\sqrt{\frac{3H}{4d^3}}x\right] \quad (10)$$

$$\eta = d + H\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right], \quad (11)$$

where g is the gravitational constant 9.8 m/s. For our simulations, we set $H = 7$ and $d = 10$. We used the same initial conditions in three spatial dimensions, replicating the two dimensional initial conditions along the z-axis.

Next, we needed to introduce a model of a sloping underwater shelf in order to cause the propagating pulse to actually pile up on itself and form a breaking wave. We performed a variety of two dimensional tests to determine the best underwater shelf geometry to generate a visually pleasing breaking wave. Figure 5 is a sequence of frames from one of the two dimensional tests we ran with the same x-y cross section as can be found in our 3D example. We found this prototyping technique to be a fast and easy way to explore possible breaking wave behaviors in a fraction of the time it would take to run a fully three dimensional test case.

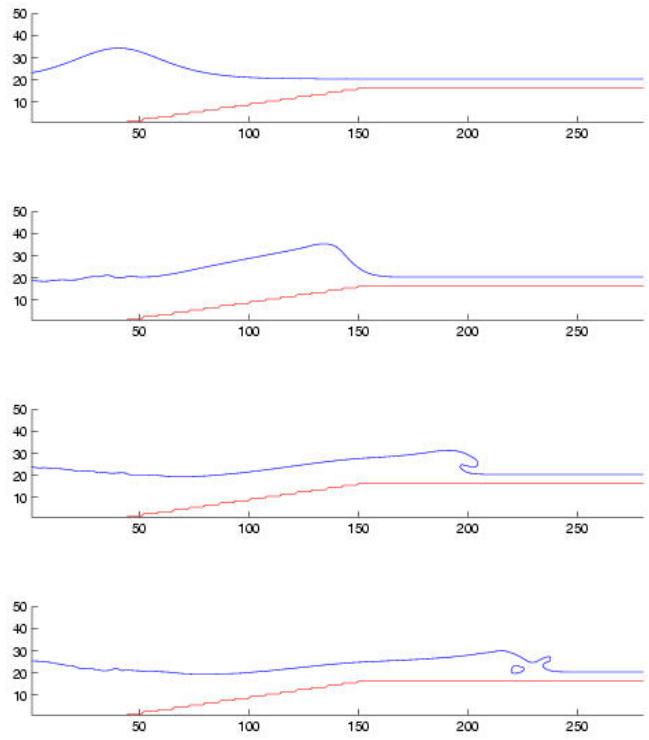


Figure 5: **Two Dimensional Breaking Wave**

After determining the best submerged shelf geometry to generate a breaking wave, we generated a slight tilt in the geometry in order to induce a curl in the break of the wave and enhance the three dimensional look. A sketch of the shelf geometry used in our three dimensional calculations is shown in figure 6. An incline of slope 1:7 rises up from the sea bottom to a depth of 2 m below the surface of the water. Instead of allowing the incline to go all the way to the surface we chose to have it flatten out at this depth in order to illustrate the splash up effects after the initial breaking of the wave.

Next we ran a fully three dimensional simulation, the results of which can be seen in figure 8. The wave has the intended curling effect. The formation of a tube of air is clearly seen after the wave splashes down, with the "air" particles maintaining the tube even after the wave begins a secondary splash up. We observe some solely three dimensional effects including the fingering of the breaking wave. The computational cost was approximately 13 minutes per frame.

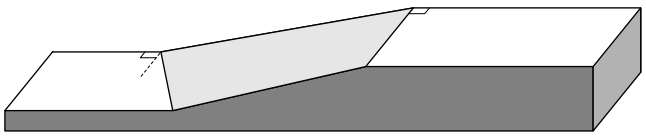


Figure 6: **Submerged Shelf**

The simulation shown captures the basic phenomena of the breaking wave on a very coarse grid, but in real waves there are small-scale features that, while not important to the *behavior* of the large wave, are very important for its *appearance*. Coupling a 2D simulation for the small scale features to the wave simulation is a promising avenue for future work. Once those waves are incorporated it will become important to treat diffusion of light through the water more correctly. Also, texture mapping of the wave, e.g. use of a Philips spectrum [Tessendorf 2001] or a bump mapping technique [Fournier and Reeves 1986], needs to be explored further. Another challenge will be to augment our method to naturally handle spray and foam.

5 Conclusion and Future Work

We have presented some novel computational methods for enhanced surface tracking and modeling of the surface motion. Combining these leads to the possibility of creating photorealistic behavior in 3D water simulations for the purpose of computer graphics animation. We discussed some advances in rendering the simulated photorealistic behavior in order to complete the illusion that real water is being seen. The computational methods presented can easily be included into an existing three dimensional fluid simulation animation tool. Also, we believe that our "thickened" front tracking approach should enable texture mapping of an implicitly defined fluid surface, and we will pursue this as future work.

6 Acknowledgment

Research supported in part by an ONR YIP and PECASE award N00014-01-1-0620, NSF DMS-0106694, NSF ACI-0121288, NSF IIS-0085864, and the DOE ASCI Academic Strategic Alliances Program (LLNL contract B341491).

The authors acknowledge Henrik Wann Jensen for the use of his *dali* rendering system and for his help with the extensions that were added to it for this paper.

The authors would also like to thank Willi Geiger, Andy Hendrickson, Neil Herzinger, Sebastian Marino, Nick Rasmussen, Philippe Rebours, and Industrial Light + Magic for their support and assistance in rendering the breaking wave.

The third author would like to acknowledge the fruitful collaboration with Nick Foster regarding the hybridization of particle and level set methods that was published in [Foster and Fedkiw 2001]. Without it, neither [Enright et al. 2002] nor this current work would have been possible.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1999. The fast construction of extension velocities in level set methods. *J. Comp. Phys. 148*, 2–22.

CHEN, J., AND LOBO, N. 1994. Toward interactive-rate simulation of fluids with moving obstacles using the navier-stokes equations. *Computer Graphics and Image Processing 57*, 107–116.

CHEN, S., JOHNSON, D., AND RAAD, P. 1995. Velocity boundary conditions for the simulation of free surface fluid flow. *J. Comp. Phys. 116*, 262–276.

CHEN, S., JOHNSON, D., RAAD, P., AND FADDA, D. 1997. The surface marker and micro cell method. *Int. J. for Num. Meth. in Fluids 25*, 749–778.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*. To appear.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM

SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graphical Models and Image Processing 58*, 471–483.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *Computer Graphics (Proceedings of SIGGRAPH 86), 20(4)*, ACM, 75–84.

HARLOW, F., AND WELCH, J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids 8*, 2182–2189.

HECKBERT, P. S. 1990. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 145–154.

HILTZIK, M. A., AND PHAM, A. 2001. Synthetic actors guild. *Los Angeles Times*. May 8, 2001, natl. ed. : A1+.

HIRT, C., AND NICHOLS, B. 1981. Volume of fluid (VOF) method for the dynamics of free boundaries. *J. Comp. Phys. 39*, 201–225.

JENSEN, H. W. 1995. Importance driven path tracing using the photon map. In *Eurographics Rendering Workshop 1995*, 326–335.

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Maps*. A K Peters.

JIANG, G.-S., AND PENG, D. 2000. Weighted eno schemes for hamilton-jacobi equations. *SIAM J. Sci. Comput. 21*, 2126–2143.

KAJIYA, J. T. 1986. The rendering equation. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol. 20, 143–150.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 49–57.

LAFORTUNE, E. P., AND WILLEMS, Y. D. 1993. Bi-directional path tracing. In *Proceedings of Compugraphics '93*, 145–153.

MARSCHNER, S., AND LOBB, R. 1994. An evaluation of reconstruction filters for volume rendering. In *Proceedings of Visualization 94*, IEEE Comput. Soc. Press, 100–107.

MASTEN, G., WATTERBERG, P., AND MAREDA, I. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Application 7*, 16–23.

MILLER, G., AND PEARCE, A. 1989. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics 13*, 3, 305–309.

NISHITA, T., AND NAKAMAE, E. 1994. Method of displaying optical effects within water using accumulation buffer. In *Proceedings of SIGGRAPH 94*, ACM SIGGRAPH / ACM Press, Computer Graphics Proceedings, Annual Conference Series, ACM, 373–381.

O'BRIEN, J., AND HODGINS, J. 1995. Dynamic simulation of splashing fluids. In *Proceedings of Computer Animation 95*, 198–205.

OSHER, S., AND FEDKIW, R. 2002. *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature dependent speed: Algorithms based on hamiliton-jacobi formulations. *J. Comp. Phys. 79*, 12–49.

PEACHEY, D. R. 1986. Modeling waves and surf. In *Computer Graphics (Proceedings of SIGGRAPH 86), 20(4)*, ACM, 65–74.

PREMOZE, S., AND ASHIKHMIN, M. 2000. Rendering natural waters. In *The proceedings of Pacific Graphics 2000*, 23–30.

RADOVITZKY, R., AND ORTIZ, M. 1998. Lagrangian finite element analysis of newtonian fluid flows. *Int. J. Numer. Meth. Engng. 43*, 607–619.

SCHACHTER, B. 1980. Long crested wave models. *Computer Graphics and Image Processing 12*, 187–201.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*. Cambridge University Press.

STAM, J. 1999. Stable fluids. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, ACM, 121–128.

TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. 1989. Heating and melting deformable models (from goop to glop). In *Graphics Interface '89*, 219–226.

TESSENDORF, J. 2001. Simulating ocean water. In *Simulating Nature: Realistic and Interactive Techniques*. SIGGRAPH 2001 Course Notes 47.

TS'O, P. Y., AND BARSKY, B. A. 1987. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics 6*, 3, 191–214.

UNVERDI, S.-O., AND TRYGGVASON, G. 1992. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comp. Phys. 100*, 25–37.

VEACH, E., AND GUIBAS, L. J. 1994. Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings*, 147–162.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Computer Graphics Proceedings, Annual Conference Series, ACM, 65–76.

WATT, M. 1990. Light-water interaction using backward beam tracing. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 377–385.

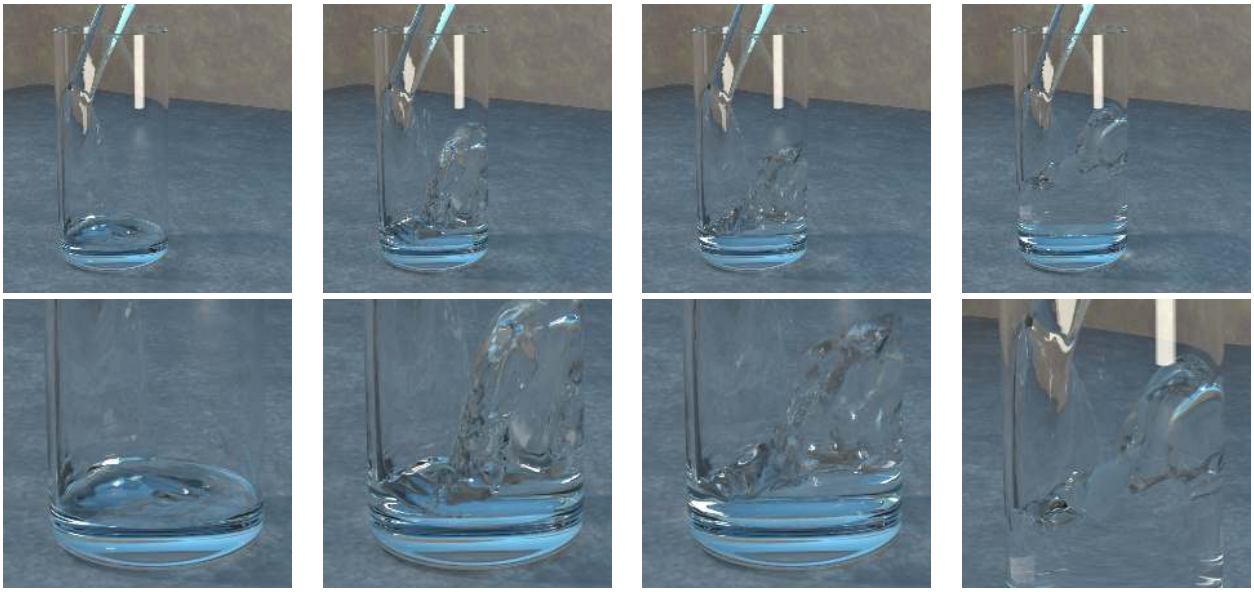


Figure 7: **Water being poured into a clear, cylindrical glass (55x55x120 grid cells). Our method makes possible the fine detail seen in the turbulent mixing of the water and air.**

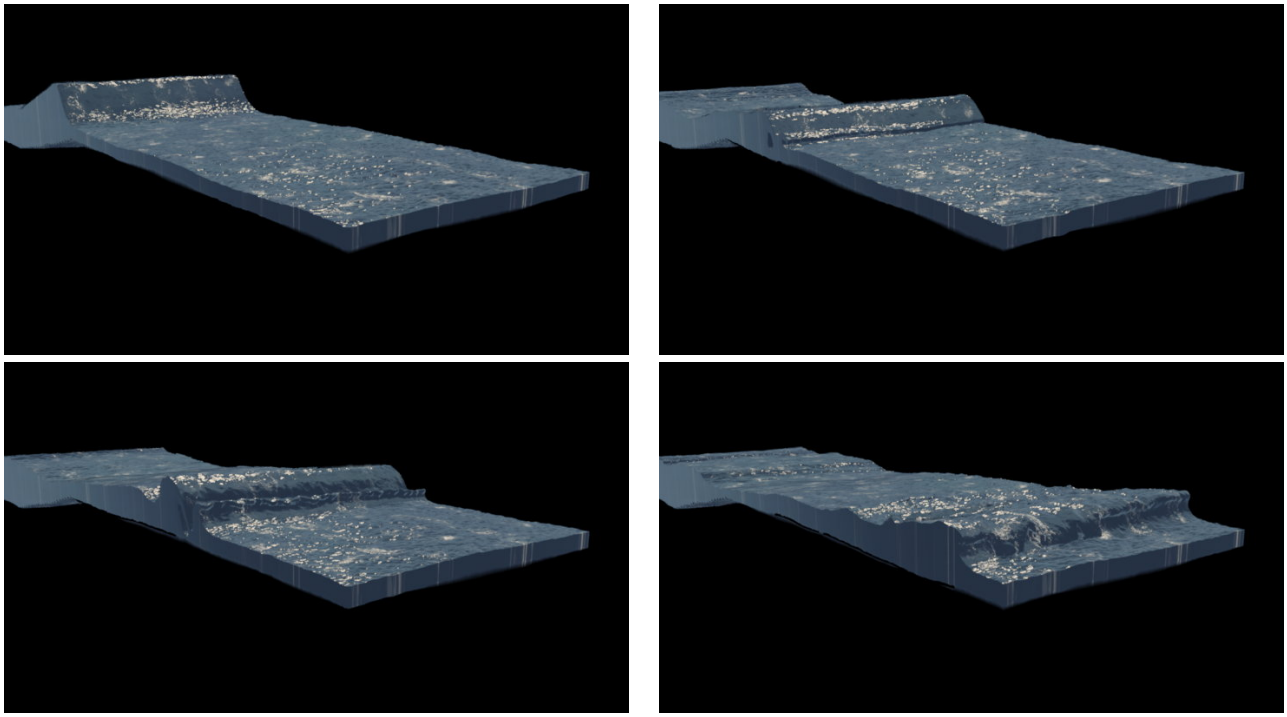


Figure 8: **View of wave breaking on a submerged shelf (540x75x120 grid cells). Note the ability to properly model the initial breaking (top two frames) and secondary splash up (bottom two frames) phases. Rendered by proprietary software at ILM.**