Wave Particles

Cem Yuksel [*]
Computer Science
Texas A&M University

Donald H. House [†]
Visualization Laboratory
Texas A&M University

John Keyser [‡]
Computer Science
Texas A&M University

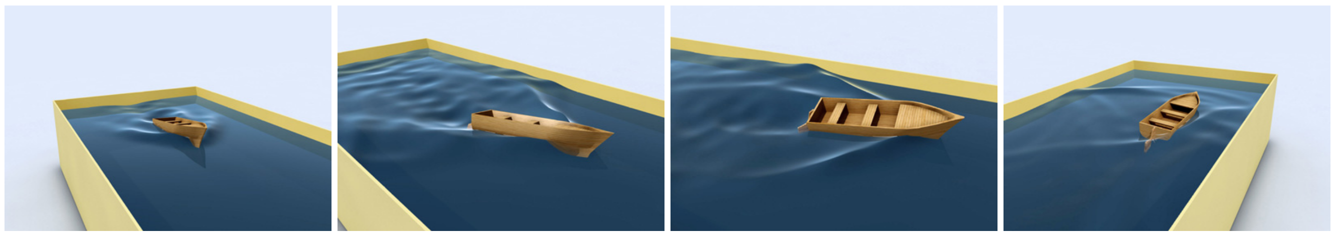

Figure 1: Sample frames captured from our real-time simulation system (approximately 100,000 wave particles)

Abstract

We present a new method for the real-time simulation of fluid surface waves and their interactions with floating objects. The method is based on the new concept of wave particles, which offers a simple, fast, and unconditionally stable approach to wave simulation. We show how graphics hardware can be used to convert wave particles to a height field surface, which is warped horizontally to account for local wave-induced flow. The method is appropriate for most fluid simulation situations that do not involve significant global flow. It is demonstrated to work well in constrained areas, including wave reflections off of boundaries, and in unconstrained areas, such as an ocean surface. Interactions with floating objects are easily integrated by including wave forces on the objects and wave generation due to object motion. Theoretical foundations and implementation details are provided, and experiments demonstrate that we achieve plausible realism. Timing studies show that the method is scalable to allow simulation of wave interaction with several hundreds of objects at real-time rates.

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

Keywords: wave particles, waves, real-time simulation, fluid-object interaction, GPU algorithms

1 Introduction

Today's water simulation techniques are highly realistic but require extensive offline computation, while achieving a similar realism in real time remains an open challenge. For real-time graphics we must resort to ad hoc methods with predictably low quality results. Furthermore, simulating interactions between water and floating objects is essential for achieving realism in a virtual environment, but physically accurate simulation of solid and fluid interaction is too expensive even for offline computation. Though a number of approximate methods are used in offline graphics, they are too complex for real-time graphics use.

[*]e-mail: cem@cemyuksel.com
[†]e-mail: house@viz.tamu.edu
[‡]e-mail: keyser@cs.tamu.edu

We will show that a broad class of visually-complex water phenomena can be modeled by an algorithmic formulation that is conceptually simple and computationally inexpensive, while being grounded in physical principle and experimental verification.

In our real-time method, we focus on the surface behavior of large bodies of incompressible, globally-flowless fluids in interaction with floating objects. We introduce the concept of wave particles to track wave motion, and to formulate deformations of the fluid surface. The fluid surface itself is represented by a height field, extended by horizontal warps induced by local flow. Using graphics hardware, we convert the wave particles to the extended height field, which is then used for rendering and computing forces applied on floating objects. To complete the representation of fluid-object interaction, the motion of floating objects is used to generate surface waves using wave particles. This method is unconditionally stable, and can easily support both bounded and unbounded configurations.

Figure 1 was created by capturing frames from a real-time simulation designed to demonstrate all aspects of our method. The boat is a dynamic element in the simulation, being driven under control of a joystick. Boat's forward motion and steering are due to interaction forces acting on its spinning propeller and its rudder. Realistic waves created by the boat's motion reflect off of the sides of the water tank, and in turn affect the motion of the boat.

In the next section we give a brief overview of previous work. Section 3 describes our wave simulation system, and interactions with floating objects are explained in Section 4. Implementation details are given in Section 5. Final sections present our results in Section 6, a discussion in Section 7, and a conclusion in Section 8.

2 Previous Work

In this section we briefly overview some of the previous work on water simulation and object interaction techniques in computer graphics. For an overview of rigid body simulations we recommend [Guendelman et al. 2003].

Early work on water simulation concentrated on modeling water as a surface using Fourier synthesis [Mastin et al. 1987] or parametric representations [Schachter 1980; Fournier and Reeves 1986; Peachey 1986; Ts'o and Barsky 1987]. In addition, [Schneider and Westermann 2001] used graphics hardware for real-time simulation of waves using a noise function [Perlin and Hoffert 1989]. For a summary on the generation of ocean waves we recommend [Tessendorf 2001]. [Tessendorf 2004] also shows how reflecting waves can be generated from objects in the water. While these methods achieve realistic results, they are not applicable for dynamic simulations of two-way object-fluid interactions.

Later work looked at shallow water equations using height fields. [Kass and Miller 1990] used finite differences to solve simplified 2D shallow water equations over a dynamic height field. [Chen and da Vitoria Lobo 1995] used a pressure-defined height field arising from a 2D solution of the Navier-Stokes equations. They proposed a way of handling two-way object to fluid coupling; however, their implementation is limited to one-way coupling only. [O'Brien and Hodgins 1995] added a particle system to simulate splashing liquids, but the height field simulation was still based on a 2D Navier-Stokes solution. [Baxter et al. 2004] used height field fluids for interactive watercolor painting. [Layton and van de Panne 2002] used implicit semi-Lagrangian integration for shallow water equations.

Particle systems [Reeves 1983] have been used in a number of ways in fluid simulation: a viscous-spring particle representation [Miller and Pearce 1989], molecular dynamics for melting solids [Terzopoulos et al. 1989], smoothed particle hydrodynamics [Müller et al. 2003], and the moving particle semi-implicit method [Premoze et al. 2003]. However, these Lagrangian methods are computationally expensive, requiring a large number of interacting particles for high quality. Interesting hybrid particle-grid approaches [Selle et al. 2005; Zhu and Bridson 2005] address this issue.

There is a large body of work done on Eulerian grid-based solutions to Navier-Stokes equations [Foster and Metaxas 1997b; Stam 1999; Foster and Fedkiw 2001]. Recent efforts to enhance the quality and reduce the computational time include: octree structures [Losasso et al. 2004], RLE implementations [Houston et al. 2006], and tetrahedral meshes [Feldman et al. 2005; Klingner et al. 2006]. One-way coupling between objects and fluid is used by many researchers in the form of boundary conditions [Foster and Metaxas 1996; Foster and Metaxas 1997a; Fedkiw et al. 2001; Enright et al. 2002; Fedkiw 2002], and a number of researchers proposed methods for two-way interactions [Takashi et al. 2002; Peskin 2002; Takashi et al. 2003; Genevaux et al. 2003; Carlson et al. 2004; Guendelman et al. 2005; Losasso et al. 2006; Klingner et al. 2006]. Recently [Irving et al. 2006] presented a technique for simulating large bodies of water with object to fluid coupling using the combination of a Navier-Stokes based fluid solver and a height field formulation, and [Losasso et al. 2006] developed a representation of multiple interacting liquids. Although their results are of high quality, all of these methods are suitable only for offline computation.

There are several interesting recent results in real-time fluid. [Angelidis and Neyret 2005] use a vorticity scheme to simulate and render turbulent smoke. [Kim et al. 2006] use the GPU to compute buoyant forces on arbitrary models, achieving one-way fluid on object interactions in real time (16 fps for about 50 objects). The model reduction approach of [Treuille et al. 2006] is especially successful, but it has large precomputation requirements and is only suitable for simulating scenarios previously "trained in". The closest work to that described in this paper is by [Jensen and Goliáš 2001]. They use grid based Eulerian methods to model deep ocean waves, including the effect of fluid on objects, but their computation of the effect of objects on the fluid involves numerical differencing and the addition of artificial damping, so we expect it to be highly sensitive to parameter tuning. Our approach can handle both deep ocean and constrained environments, and has a much simpler computational framework.

3 Wave Simulation

We begin the explanation of our method with a basic height field representation, introducing the extended height field and physically correct wave simulation at the end of this section.

A basic height field is defined as a continuous function of fluid level z over the horizontal position $\mathbf{x} = (x, y)$. External forces and interactions with objects generate deviations on the water surface, which turn into surface waves that propagate with speed υ, satisfying the second order wave equation

$$\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} = \frac{1}{\upsilon^2} \frac{\partial^2 z}{\partial t^2} \ . \tag{1}$$

We formulate an analytical solution to Equation 1 by first representing the height field as

$$z(\mathbf{x}, t) = z_0 + \eta_z(\mathbf{x}, t) \ , \tag{2}$$

the sum of base height z_0 and deviation field η_z; then formulating the deviation field as

$$\eta_z(\mathbf{x}, t) = \sum_i D_i(\mathbf{x}, t) \ , \tag{3}$$

the sum of a set of *local deviation functions*, each traveling with the wave's speed υ.

The heart of our method is to associate each local deviation function with what we call a *wave particle*, which moves with the wave's velocity. We can formulate the local deviation function corresponding to particle i as

$$D_i(x, t) = a_i \, W_i \left(\, x - x_i(t) \, \right) \ , \tag{4}$$

where a_i is the amplitude, W_i a constant *waveform function* and $x_i(t)$ the particle's position at time t. Each deviation function is stationary relative to its wave particle, and the position and propagation direction of the wave particle, along with a number of other wave particle properties, define the shape and behavior of its local deviation function. This approach reduces the dynamics of wave simulation to tracking a system of particles moving on a plane.

3.1 Construction of waves

The construction of waves from wave particles requires finding waveform functions that give a plausible wave shape and satisfy Equation 1. It also requires forming the waves into wavefronts that move in unison across the water surface.

In two dimensions, the natural choice for the waveform function is sinusoidal, giving a shape similar to the vertical deviation of most

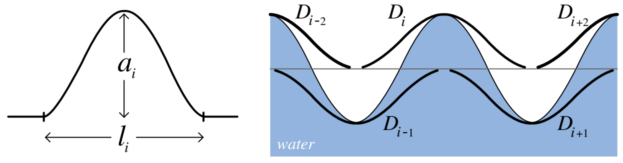

Figure 2: (a) Shape of waveform function, (b) Continuous waves constructed from local deviation functions

water surface waves. We use

$$W_i(u) = \frac{1}{2}\left(\cos\left(\frac{2\pi u}{l_i}\right) + 1\right) \Pi\left(\frac{u}{l_i}\right) , \quad (5)$$

where l_i is the wavelength and Π is a rectangle function[1]. Besides shape and the fact that it satisfies Equation 1, this particular waveform function is a good choice because:

- It is non-zero over a finite range, with zero first derivative at the endpoints (Figure 2a).
- It is easy to create continuous waves with fixed wavelength by the alternate spacing of local deviation functions with positive and negative amplitudes (Figure 2b).

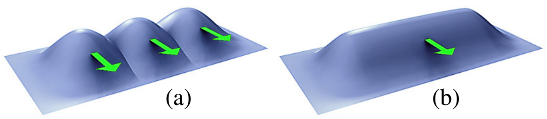
Figure 3: (a) Individual wave particles (b) Wavefront formed by these wave particles

In three dimensions, water surface waves take the form of continuous wavefronts. Instead of formulating our local deviation functions to represent a whole wavefront, we model wavefronts by placing local deviation functions side by side as shown in Figure 3, and blending their shapes so that they are non-zero only over a finite area. Thus, the local deviation function for particle i is

$$D_i(\mathbf{x},t) = a_i\, W_i(u)\, B_i(v) , \quad (6)$$

where $u = \hat{\mathbf{u}}_i \cdot (\mathbf{x} - \mathbf{x}_i)$ and $v = \hat{\mathbf{u}}_i^{\perp} \cdot (\mathbf{x} - \mathbf{x}_i)$ are in the local coordinates of the particle such that $\hat{\mathbf{u}}_i$ is the propagation direction and $\hat{\mathbf{u}}_i^{\perp}$ is a horizontal direction perpendicular to propagation, and B_i is a *blending function*. Note that the choice of blending function is somewhat arbitrary: any function that has finite support and whose translates sum to one is acceptable, yielding local deviation functions that are non-zero only over a finite quadrilateral area.

3.2 Wave Particles

Wave particles can be easily implemented using the generalized formulation explained above. However, here we introduce an approximation to Equation 6, using a deviation function with radial support. This radial formulation allows us to implement wave particles efficiently on currently available graphics hardware. First, notice that Equation 5 can be used as the blending function B_i in Equation 6. Rather than the quadrilateral definition of D_i in Equation 6, we use a radial definition formulating our local deviation functions as

$$D_i(\mathbf{x},t) = \frac{a_i}{2}\left(\cos\left(\frac{\pi|\mathbf{x} - \mathbf{x}_i(t)|}{r_i}\right) + 1\right) \Pi\left(\frac{|\mathbf{x} - \mathbf{x}_i(t)|}{2\, r_i}\right) , \quad (7)$$

where r_i is the radius of the wave particle. This radial definition introduces some error in the final wavefront shape, the amount of which is determined by the distance between two neighboring particles. We empirically found that by keeping this distance smaller than one half of the wave particle radius, the maximum deviation of the waveform shape from ideal is less than 3% of the peak amplitude, while the maximum deviation along the wave crest is less than 0.1%.

Wave particles can do more than track wave position over time. They can also carry additional properties describing the shape and behavior of their local deviation functions, such as amplitude and radius. For example, wave amplitude may be decreased with time to account for energy loss due to viscosity or other damping.

[1]Rectangle function $\Pi(x)$ is 1 for $|x| < \frac{1}{2}$, $\frac{1}{2}$ for $|x| = \frac{1}{2}$, and 0 otherwise.

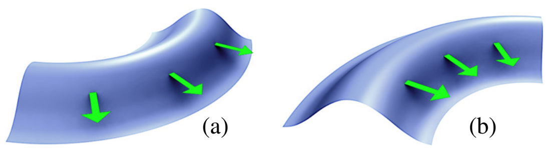
Figure 4: (a) Expanding wavefront (b) Contracting wavefront

In reality, wavefronts can be *expanding* or *contracting* as shown in Figure 4. Therefore, the distance between neighboring particles in a wavefront changes over time. To account for this behavior, wave particles also carry a *dispersion angle*. Since our wave speed is constant, we can use this property to track the distance between neighboring particles, avoiding explicit inter-particle distance computation. To keep the surface representation error within the limits described above, we convert a wave particle into three new wave particles when the distance is above half of the wave particle radius. We call this procedure wave particle *subdivision*. When a wave particle goes through subdivision, amplitudes and dispersion angles of the three child particles become one third of the parent particle, while the radii of the new particles stay the same as the parent.

Just like water surface waves, wave particles do not interact with each other, and their speed is determined by the water medium. The superposition of all wave particles (i.e. local deviation functions) gives the total deviation of the water surface.

3.3 Boundaries

Boundaries are the edges of the container that holds the simulated water. Wave particles bounce back from the boundaries to simulate reflecting waves. When the boundary is curved, this reflection changes the dispersion angle of the wave particle according to the curvature of the boundary. Note that when the water volume is contained within closed boundaries, once a wave is generated it always stays within these boundaries. When simulating an ocean surface we simply remove all the boundaries, and waves become free to move away and out of the current view. Thus, with our method it is actually easier to simulate an ocean than a water pool.

3.4 Water Waves and Extended Height Field

The basic height field formulation we have presented up to now accounts only for transverse waves, but water surface waves are actually a combination of both transverse and longitudinal components (Figure 5). This is why water moves in circles when a wave travels on the surface, as shown in Figure 6. This longitudinal component plays an important role in many parts of the simulation, including simulating realistic interactions with floating objects, conserving energy during wave propagation and superposition, and giving a realistic shape to the simulated wavefront. To account for this, we extend the basic height field definition so that it includes the horizontal surface deviations due to the longitudinal waves.

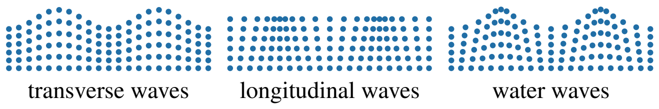

Figure 5: Transverse and longitudinal components of water waves

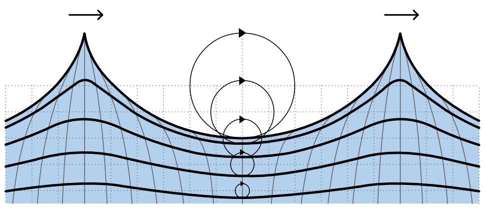
Figure 6: Isopressure lines showing the circular motion of water due to surface waves [Gerstner 1802]

The extended height field is in fact a surface deformation field $\eta : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, such that the final position $\mathbf{x}'$ of a point $\mathbf{x}$ on the water surface is

$$\mathbf{x}'(\mathbf{x},t) = \mathbf{x} + \eta(\mathbf{x},t) \ . \tag{8}$$

The vertical component of η is η_z from Equation 3. Similarly, the horizontal component is

$$\eta_{xy}(\mathbf{x},t) = \sum_i \mathbf{D}_i^L(\mathbf{x},t) \ . \tag{9}$$

$\mathbf{D}_i^L$ is a *longitudinal local deviation function* (Figure 7), which can be formulated as

$$\mathbf{D}_i^L(\mathbf{x},t) = \mathbf{L}_i(\hat{\mathbf{u}}_i \cdot (\mathbf{x} - \mathbf{x}_i))\, D_i(\mathbf{x},t) \tag{10}$$

where $\mathbf{L}_i$ is a vector function describing the longitudinal waveform. We derive the longitudinal waveform that corresponds to our transverse waveform function (Equation 5) from the circular motion of continuous waves (Figure 6). A longitudinal waveform that gives a circular motion when combined with the transverse component is

$$\mathbf{L}_i(u) = -\sin\left(\frac{\pi u}{r_i}\right) \Pi\left(\frac{u}{2r_i}\right) \hat{\mathbf{u}}_i \ . \tag{11}$$

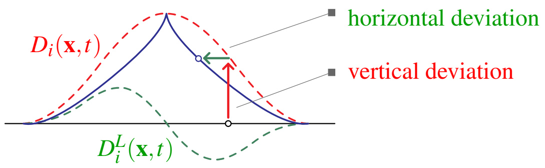

Figure 7: Components of the local deviation function in 2D

In our extended height field, along with the surface deviations, we also keep surface gradient (to compute surface normals) and 3D surface velocity derived from space and time derivatives of the above functions. To find the velocity at any point inside the fluid volume, we exponentially scale down the surface velocity with depth as in Figure 6 [Gerstner 1802].

4 Fluid Object Interaction

We treat interactions between fluids and objects by separating the interaction into two separate one-way couplings. Most interactions use the velocity of the individual object face relative to the velocity of the fluid at the face.

4.1 Object to Fluid Coupling

When an object is fully or partially submerged, it affects existing waves on the fluid surface and also introduces new waves due to its relative motion. The height field deviation then becomes the summation of the affected existing waves and the new waves created by the submerged object. Instead of actually modifying a wave particle interacting with an object, a new particle is generated that accounts for the modification. For example, a negative wave created on top of an existing wave would effectively delete it. Therefore, existing waves propagate without any modification, and we can simulate the object to fluid coupling simply by generating new waves.

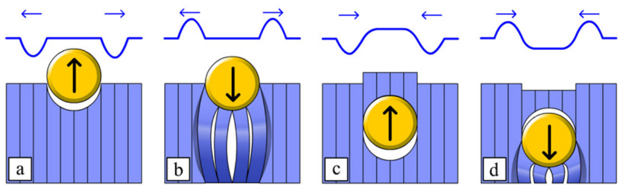

Figure 8: Cases of wave generation, (a-b) object is on the surface, (c-d) object is inside the fluid volume.

Instead of using wave particles directly for object to fluid coupling, we use the extended height field formulation generated from the wave particles. Thus, we eliminate collision detection between wave particles and objects, yet still achieve plausible results. This is done by generating new wave particles (including those to reflect and/or cancel waves) directly from the motion of the object relative to the fluid motion in the extended height field.

Our wave generation approach is based on volume conservation. Note that here we do not enforce energy conservation, since during wave generation a portion of the energy is lost to heat and turbulence. Thus, the energy of the waves does not correspond to the whole energy transfer, whereas volume is always conserved.

For wave generation, we consider the water body around the interacting object as being organized into vertical cells similar to a height field fluid approach (Figure 8). Here, the amplitude of a wave corresponds to the volume change in the cells, and the wave direction is set according to the expected pressure difference between neighboring cells caused by the object motion. Basically, we generate a positive amplitude wave where the object is *pushing* the fluid, and a negative amplitude wave where the object is *pulling* the fluid around it. The generated waves move away from the object unless the object is fully submerged. Since the object shape and motion can be much more complicated than the ones shown in Figure 8, we handle each face of the object separately. Even though this is a highly simplified wave generation approach, it achieves plausible realism in our test scenes.

We start by computing the volume of fluid displaced by the relative motion of each face. This is given by

$$V = A_{face}(\mathbf{U} \cdot \mathbf{N})\Delta t, \tag{12}$$

where A_{face} is the face area, $\mathbf{U}$ is the relative velocity at the face center, $\mathbf{N}$ is the face normal, and Δt is the time step. Note that V is positive if the fluid is pushed by the face, and negative if the fluid is pulled. Since we do not have a 3D fluid simulation, we also exponentially scale down V as the depth into the fluid increases, such that the object motions deep inside the fluid have less effect on the surface.

The next step is to use this displaced volume for generating waves on the surface. If the face is on the top side of the object as in Figure 9a, we generate a ripple on the surface (a wave particle with 2π dispersion angle), which corresponds to the volume displaced by the face. Otherwise, as in Figure 9b, we distribute its displaced volume to the nearest points that have a direct connection to the fluid surface, and then generate waves from these points.

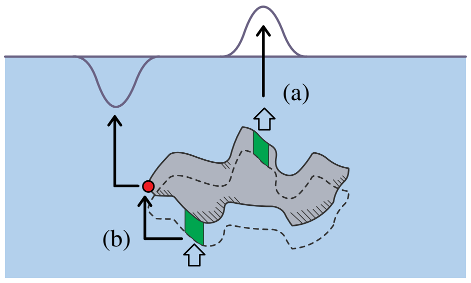

Figure 9: (a) faces that have direct access to the surface, (b) faces that do not have direct access to the surface

4.2 Fluid to Object Coupling

To allow for realistic motion of objects in the water, we incorporate fluid to object coupling by applying static and dynamic forces on the floating objects. Here we present the formulations we use to compute these forces. We recommend that the interested reader refer to [Munson et al. 2006] for detailed explanations.

The static component of the force is buoyancy

$$\mathbf{F}_{buoyancy} = -\mathbf{g}\,\rho\,V_{displaced}\ , \tag{13}$$

where $\mathbf{g}$ is the gravitational acceleration, ρ is the density of the fluid, and $V_{displaced}$ is the volume of the object inside the fluid. After computing the buoyant force, it is applied to the centroid of the part of the object inside the fluid.

The dynamic forces are computed over the interface of fluid and object and they are based on the relative motion of the object surface to the fluid. The component of the dynamic force in the opposite direction of relative motion is *drag force* and the component perpendicular to the relative motion is *lift force* (Figure 10). While computing these dynamic forces, we assume that the total force acting on the object is the sum of the forces acting on each face, applied at the face centroids. Even though this ignores the interactions between faces, generally it is a reasonable approximation. The drag and lift forces acting on each face are

$$\begin{aligned} \mathbf{F}_{drag} &= -\frac{1}{2}\,\rho\,C_D\,|A|\,|\mathbf{U}|\,\mathbf{U}\ , && (14) \\ \mathbf{F}_{lift} &= -\frac{1}{2}\,\rho\,C_L\,A\,|\mathbf{U}| \left(\mathbf{U}\times\frac{(\mathbf{N}\times\mathbf{U})}{|(\mathbf{N}\times\mathbf{U})|}\right)\ , && (15) \end{aligned}$$

where C_D and C_L are the drag and lift coefficients, $\mathbf{U}$ is the relative velocity, and A is the effective area of the face. The values of C_D and C_L as well as the formulation of A depend on the shape of the object and properties of the fluid medium, and they are typically obtained from experiments or advanced analysis. Here we leave C_D and C_L as user defined parameters so that objects with different aerodynamic properties can be simulated. The effective area of a surface is complicated to define, and depends on the overall structure of the object, however it always ranges between the projected area in the direction of motion, and the total area of the surface. Thus we formulate it as

$$A = \left(\frac{\mathbf{N}\cdot\mathbf{U}}{|\mathbf{U}|}\,\alpha + (1-\alpha)\right) A_{face}\ , \tag{16}$$

where A_{face} is the area of the face inside the fluid and $0 \le \alpha \le 1$ is a user defined parameter used to adjust for effective area. In our experiments we set alpha to one. Smaller values can be used to dampen object motion.

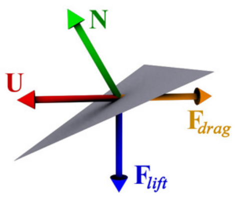

Figure 10: Directions of drag and lift forces on an object face moving with velocity $\mathbf{U}$ relative to the fluid's local velocity.

5 Implementation Details

Here we give an overview of our wave simulation and object interaction implementation, providing details about nontrivial steps and some optimization techniques. Most of our operations use graphics hardware only for rendering simple primitives and filtering, therefore they can be implemented on any GPU with floating point support. The main structure of a time step looks like

```
TimeStep ( )
    IterateWaveParticles ( )
    ComputeObjectForces ( )
    IterateObjects ( )
    GenerateWaveParticles ( )
    RenderHeightFields ( )
```

`IterateWaveParticles` moves the wave particles to their new positions and handles subdivision and reflection events that occur within the time step. The wave particles can be easily iterated on the GPU, but even this iteration can be eliminated and the position of a wave particle can be found using its velocity and its known position at any time. However, wave particles subdivide and reflect from stationary boundaries, so `IterateWaveParticles` should still handle these events (note that we do not compute wave particle collisions with dynamic objects as explained in Section 4.1). Fortunately, this can be done using a time table of events, since when a wave particle is generated, its exact subdivision or reflection times can be found. Using this approach, `IterateWaveParticles` visits only those particles that should subdivide or reflect within the current time step. Therefore, iteration time is related to the number of events, but not directly related to the total number of wave particles in the system. This allows the iteration of millions of wave particles with minimal cost. Before subdividing a wave particle, we check its amplitude, and if the amplitude is below a user defined threshold, kill the wave particle instead of subdividing.

`ComputeObjectForces` computes static and dynamic forces on the GPU and applies them to objects. To find $V_{displaced}$ in Equation 13, we render the object in top view with additive blending. In the fragment shader, for backfacing fragments we assign the depth of the fragment as the color value, and for frontfacing fragments we assign negative depth value, while discarding the fragments that are outside the fluid volume. Multiplying the final pixel values by the pixel area yields $V_{displaced}$. We compute the dynamic forces using equations 14, 15 and 16 onto a floating point texture, where each pixel corresponds to an object face. Then we copy these textures to main memory and use them to determine applied forces on the objects.

`IterateObjects` handles the rigid body simulation. Deformable objects can also be simulated here. Although our implementation of `IterateObjects` runs on the CPU, a GPU implementation would better suit our system, since it would eliminate texture reads to main memory.

`GenerateWaveParticles` creates new wave particles due to the computed object motion. For efficiency reasons, while computing the dynamic forces we also calculate the volume of fluid pushed or pulled by each face using Equation 12. Here we render a low resolution silhouette of the part of the object inside the fluid from a top view and identify the boundary pixels of the silhouette. If the face is on top of the object, we generate a wave particle at the position of the face, otherwise we distribute the displaced volume by the face to the nearest boundary pixels. After all the displaced volumes are distributed, we generate waves from the boundary pixels. The directions of the generated waves are outwards from the boundary pixels unless the part of the object on the boundary is fully inside the fluid volume (Figure 8), and the dispersion angle is determined by examining the directions of neighboring pixels.

`RenderHeightFields` converts the wave particle representation to an extended height field. Our implementation of the wave particle system uses the same radius value for all particles. Because of this, `RenderHeightFields` can be implemented efficiently by drawing wave particles as point primitives on the height field texture, which significantly reduces the number of fragments to be blended. Then we filter this image using Equation 7 as a kernel. This filtering can be further optimized by approximating this kernel by a horizontal followed by a vertical filtering with a 1D version of the kernel given by Equation 5.

For most of the fluid object interactions, we need to find the fluid height and velocity at the surface right above a point inside the fluid volume. However, the values that we read from the extended height field on a surface point also have horizontal displacement, so this surface point is often not right above the point of interest. Finding the correct location on the extended height field requires the inverse transformation of the final horizontal displacement field. To eliminate this complicated procedure, we convert the extended height field to a basic height field by rendering the fluid surface using the extended height field onto the basic height field texture. This overhead can be minimized by using a low resolution version of the surface for this conversion.

6 Results

To provide enhanced visualizations of our approach, we use the GPU based techniques described in [Johanson 2004] for water rendering, with real-time caustics as in [Shah et al. 2007] (Figure 13). For scenes involving ocean surfaces, we couple the water surface grid to the camera as in [Johanson 2004], and we generate ambient waves using the method of [Tessendorf 2001].

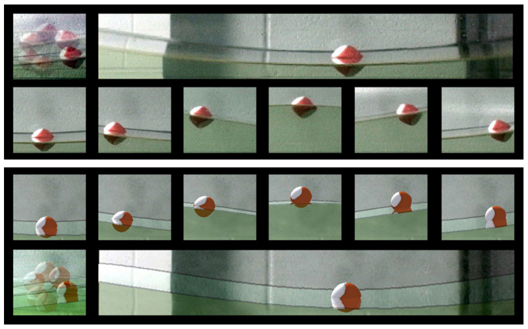

Figure 11: Circular motion due to waves moving from left to right. (top) real video capture, (bottom) simulated using our method.

In order to evaluate our method, we video taped a number of experiments in a wave tank and compared them with our wave simulations. In the simulations, wave particles are generated at one side of the tank to emulate the tank's wave generator. Figure 11 shows one such comparison. The larger images show that the overall wave shape is effectively mimicked. The sequences of smaller frames demonstrate that the circular motion induced by the waves is reproduced in our simulation, successfully capturing both transverse and longitudinal wave action.

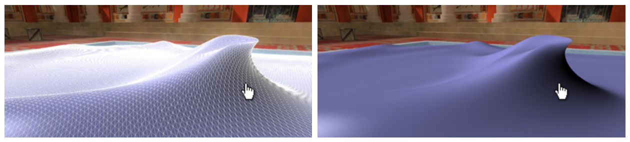

Figure 12: Exaggerated effect of extended height field.

Although we are using a height field approach, the height field is extended to allow for a significant longitudinal component affecting the wave shape. The effect of this component is demonstrated in exaggerated form by the wave shown in Figure 12.

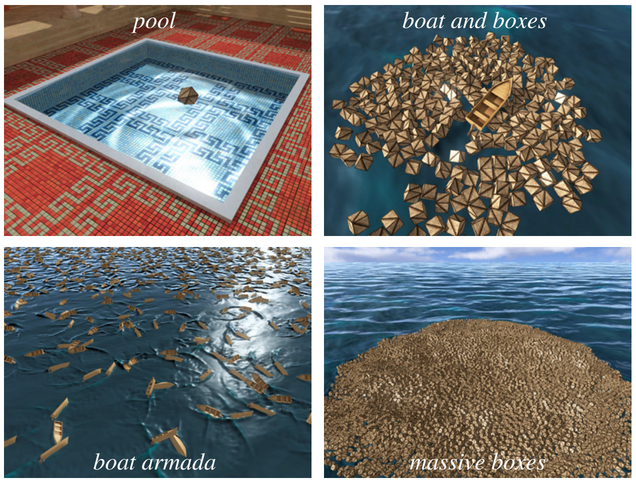

Figure 13: Pool and ocean test scenes with boats and boxes.

To demonstrate the speed and scalability of our approach, we simulated a number of scenes on a standard PC with a 2.13GHz Core2 Duo processor and GeForce 7900 graphics card, recording performance data for each run. This data is shown in Table 1. Our *boat in tank* scene from Figure 1 has a height field resolution of 512×128, and includes a single propeller and rudder as user controllable elements. We allowed the simulation to use up to 100,000 active wave particles by keeping the amplitude threshold for killing wave particles very low. Nearly identical results can be achieved using far fewer (less than 10.000) wave particles. Our *boat & boxes* scene, shown in Figure 13, used 600,000 active wave particles, a height field resolution of 256×512, and included an active boat driving through water populated with 125 boxes with a total of 6176 faces. We achieved real-time performance in both of these scenes. To

Table 1: Time results in milliseconds

	boat in tank	boat & boxes	boat armada	massive boxes
`IterateWaveParticles`	0.28	1.28	57.54	85.95
`ComputeObjectForces`	0.29	1.74	36.95	129.23
`IterateObjects`	0.04	0.49	75.51	1061.60
`GenerateWaveParticles`	1.15	2.59	142.50	115.15
`RenderHeightFields`	4.97	20.40	132.93	159.12
Total Simulation Time	5.87	24.49	206.78	1073.38
Simulation Frames Per Second	170 fps	40.8 fps	4.84 fps	0.93 fps

stretch our method, we constructed two massive scenes shown in Figure 13 using the same boat and box models and height field resolution, but extending to 8,000,000 active wave particles and many more rigid body elements. The first, *boat armada*, had 1681 active boats with 295,856 faces, and runs at several frames per second. The second, *massive boxes*, had 9261 falling and floating boxes with 444,528 faces, and runs at nearly one frame per second. Here the rigid body simulation becomes the bottle-neck, due to interactions between boxes.

The computation times for each step of the simulation have different scene complexity dependencies. The cost of `IterateWaveParticles` is linearly dependent on the number of wave events within a time step, which reflects the number of object faces in the scene interacting with the fluid and the shape of generated waves. On the other hand, the times of `ComputeObjectForces` and `GenerateWaveParticles` depend only on the number of object faces, while `IterateObjects` is related to the number of objects and collisions between objects. While `RenderHeightFields` has a constant cost due to the filtering operation that is related to the height field resolution and the wave particle radius, its computation time also depends on the number of active wave particles. However, this dependence is quite weak, as long as the number of active wave particles is not too high (less than 100,000 on our test hardware).

7 Discussion

The method presented here concentrates on efficiently simulating a subset of fluid behavior with object interactions, rather than formulating an ultimate technique to handle all fluid phenomena. We restrict ourselves to surface waves, therefore our approach is only applicable when the 3D flow is unimportant.

Within this solution domain, our method has some important advantages over a full fluid simulation. Perhaps the most important are its speed and scalability. Wave particles offer a fast and unconditionally stable wave simulation, and their generalized form is sufficient to represent all possible surface deformations with correct wave behavior. Damping in our system is not a result of numerical simplification [Stam 1999], and can be added or eliminated without affecting the stability.

In our formulation, when the amplitude of a wave is greater than half of its length, the final wave shape intersects with itself as observed by [Gerstner 1802]. [Bascom 1980] reports that waves become unstable and break when the amplitude is greater than one-seventh of the wave length. Therefore, when a wave particle with high amplitude is introduced, it could be used as an indication for secondary effects like splash and foam.

Using graphics hardware, we can efficiently convert wave particles to an extended height field, which can be directly used for rendering without an additional cost of surface generation. The decoupling of height field representation and wave simulation allows biased sampling of the final surface to concentrate the detail in important areas (near the camera) for simulating a large surface like the ocean.

Our fluid-to-object coupling technique is based on the assumptions and simplifications typically used in fluid mechanics. Therefore, we can generate realistic object motion from interactions with the fluid. The scalability of our method allows the simulation of several thousand interacting objects in a fraction of a second.

The true physical process for water wave generation can be very complex, involving turbulent flow that we do not attempt to model. We have introduced a simplified wave generation approach based on volume conservation. The direction of initial wave movement is based on a simple model of pressure induced by object movement. More experimental work is needed here, to perfect this approach.

8 Conclusion

We have introduced the concept of wave particles for the fast and stable simulation of surface deformations, and shown how to use these wave particles for simulating surface waves with floating object interaction. Our method is fast, is highly scalable, and can produce plausible realism for fluid and floating object interactions in real time. Therefore, it is ready to be used in interactive applications. Although our method is tailored to the requirements of real-time graphics, it can easily be used in offline simulations.

Future extensions of our approach include improved wave generation with splash, simulating breaking waves with wave particles, diffraction of waves, and coupling wave particle simulation with a 3D fluid solver to extend the solution domain of our method, while minimizing the computational overhead and improving the scalability of the fluid solver. Since animators highly value the ability to observe motion in real time, our method could be used to form the heart of a wave choreography system.

Acknowledgements

We would like to thank Can Yuksel, Zeki Melek, and Levent Yilmaz for their help. We would also like to thank Dr. Kuang-An Chang and H. A. K. S. Ariyarathne for our wave-tank experiments. This work is supported in part by NSF grant CCR-0220047 and ITR-0326194.

References

ANGELIDIS, A., AND NEYRET, F. 2005. Simulation of smoke based on vortex filament primitives. In *ACM-SIGGRAPH/EG Symposium on Computer Animation*.

BASCOM, W. 1980. *Waves and Beaches*. Anchor Books, Garden City, NY.

BAXTER, W., WENDT, J., AND LIN, M. C. 2004. Impasto: a realistic, interactive model for paint. In *NPAR '04*, 45–148.

CARLSON, M., MUCHA, P. J., AND TURK, G. 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. 23*, 3, 377–384.

CHEN, J. X., AND DA VITORIA LOBO, N. 1995. Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graph. Models Image Process. 57*, 2, 107–116.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. In *Proc. of SIGGRAPH '02*, 736–744.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proc. of SIGGRAPH '01*, 15–22.

FEDKIW, R. P. 2002. Coupling an eulerian fluid calculation to a lagrangian solid calculation with the ghost fluid method. *J. Comput. Phys. 175*, 1, 200–224.

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. In *Proc. of SIGGRAPH '05*, 904–909.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. of SIGGRAPH '01*, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models Image Process. 58*, 5, 471–483.

FOSTER, N., AND METAXAS, D. 1997. Controlling fluid animation. In *CGI '97: Proc. of the 1997 Conference on Computer Graphics International*, 178.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Proc. of SIGGRAPH '97*, 181–188.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *Proc. of SIGGRAPH '86*, 75–84.

GENEVAUX, O., HABIBI, A., AND DISCHLER, J. M. 2003. Simulating fluid-solid interaction. In *Graphics Interface*, 31–38.

GERSTNER, F. V. 1802. Theory of waves. *Abhandlungen der Koenigl, boehmischen Gesellschaft der Wissenschaften zu Prag*.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. 22*, 3, 871–878.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. In *Proc. of SIGGRAPH '05*, 973–981.

HOUSTON, B., NIELSEN, M. B., BATTY, C., NILSSON, O., AND MUSETH, K. 2006. Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Trans. Graph. 25*, 1, 151–175.

IRVING, G., GUENDELMAN, E., LOSASSO, F., AND FEDKIW, R. 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *Proc. of SIGGRAPH '06*, 805–811.

JENSEN, L. S., AND GOLIÁŠ, R. 2001. Deep-water animation and rendering. In *Proc. Game Developer's Conference*.

JOHANSON, C. 2004. *Real-time Water Rendering*. Ms thesis, Lund University.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Proc. of SIGGRAPH '90*, 49–57.

KIM, J., KIM, S., KO, H., AND TERZOPOULOS, D. 2006. Fast gpu computation of the mass properties of a general shape and its application to buoyancy simulation. *The Visual Computer 22*, 856–864.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. In *Proc. of SIGGRAPH '06*, 820–825.

LAYTON, A. T., AND VAN DE PANNE, M. 2002. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer 18*, 1, 41–53.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3, 457–462.

LOSASSO, F., IRVING, G., AND GUENDELMAN, E. 2006. Melting and burning solids into liquids and gases. *IEEE Trans. on Vis. and Comp. Graph. 12*, 3, 343–352.

MASTIN, G. A., WATTERBERG, P. A., AND MAREDA, J. F. 1987. Fourier synthesis of ocean scenes. *IEEE Comput. Graph. Appl. 7*, 3, 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.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 154–159.

MUNSON, B. R., YOUNG, D. F., AND OKIISHI, T. H. 2006. *Fundamentals of fluid mechanics*. Wiley, New York, NY.

O'BRIEN, J. F., AND HODGINS, J. K. 1995. Dynamic simulation of splashing fluids. In *CA '95: Proc. of the Computer Animation*, 198.

PEACHEY, D. R. 1986. Modeling waves and surf. In *Proc. of SIGGRAPH '86*, 65–74.

PERLIN, K., AND HOFFERT, E. M. 1989. Hypertexture. In *Proc. of SIGGRAPH '89*, 253–262.

PESKIN, C. S. 2002. The immersed boundary method. *Acta Numerica 11*, 479–517.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle-based simulation of fluids. In *Comp. Graph. Forum (Eurographics Proc.)*, vol. 22, 401–410.

REEVES, W. T. 1983. Particle systems a technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2, 91–108.

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

SCHNEIDER, J., AND WESTERMANN, R. 2001. Towards real-time visual simulation of water surfaces. In *VMV '01: Proc. of the Vision Modeling and Visualization Conference 2001*, 211–218.

SELLE, A., RASMUSSEN, N., AND FEDKIW, R. 2005. A vortex particle method for smoke, water and explosions. In *Proc. of SIGGRAPH '05*, 910–914.

SHAH, M. A., KONTTINEN, J., AND PATTANAIK, S. 2007. Caustics mapping: An image-space technique for real-time caustics. *IEEE TVCG 13*, 2, 272–280.

STAM, J. 1999. Stable fluids. In *Proc. of SIGGRAPH '99*, 121–128.

TAKASHI, T., HEIHACHI, U., AND KUNIMATSU, A. 2002. The simulation of fluid-rigid body interaction. In *SIGGRAPH '02: Sketches & Applications*, 226.

TAKASHI, T., FUJII, H., KUNIMATSU, A., HIWADA, K., SAITO, T., TANAKA, K., AND UEKI, H. 2003. Realistic animation of fluid with splash and foam. *Computer Graphics Forum 22 3*, 391–401.

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

TESSENDORF, J. 2001. Simulating ocean water. In *Simulating Nature: Realistic and Interactive Techniques*, ACM SIGGRAPH '01 Course #47 Notes.

TESSENDORF, J. 2004. Interactive water surfaces. *Game Programming Gems 4*.

TREUILLE, A., LEWIS, A., AND POPOVIC, Z. 2006. Model reduction for real-time fluids. *ACM Trans. Graph. 25*, 3, 826–834.

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 Trans. Graph. 6*, 3, 191–214.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 3, 965–972.