Eurographics Workshop on Natural Phenomena (2009)
E. Galin and J. Schneider (Editors)

Real-Time Open Water Environments with Interacting Objects

H. Cords and O. Staadt

Visual Computing Group, University of Rostock, Germany

Abstract

Large bodies of water are an integral part of nature and, thus, are of high interest for interactive 3D applications, e.g., computer games and virtual environments. We present a new scheme for real-time wave simulation in large-scale water environments with physics-based object interaction. In addition to a fast and realistic liquid representation, our method focuses on the creation of plausible detailed waves caused by moving boats. We expand the well-known wave equation by applying it to moving grids to simulate an apparently limitless body of water. Additionally, we present a fast, particle-based boat simulation, which is coupled to water simulation. Importantly, most parts of our method can be implemented efficiently on GPUs. We demonstrate the visual realism and performance of our approach with several experiments using different boats and other floating objects, achieving high frame rates on a desktop PC.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Simulation of complex, physics-based liquid effects has become a staple in the field of computer animation and visual simulation. Today's offline-methods are realistically and detailed, but computational cost is very high. Due to the complexity of fluid physics in principle, achieving comparable realism in real-time is an ongoing challenge that is aided by the increasing performance of today's GPUs.

This paper describes a method to simulate interactive large-water environments that supports detailed surfaces with focus on object-wave interaction. We expand different techniques to achieve high-quality results in real-time. We reduce complexity by applying an adaptive height field-based method. It is based on the 2D wave equation and is solved using a finite difference method (FDM). Thus, a fine discretization can be used for simulation – even in interactive environments. Using a moving grid method, detailed simulation is carried out only in areas where it is needed. We extend the approach to the use of multiple grids to obtain an adaptive simulation. Additionally, we describe an efficient boat simulation and a two-way coupling with liquids simulation. Hence, realistic boat movements and boat waves can be created. The scheme has been evaluated in practice, achieving good performance and realistic results due to extensive use of the GPU. The approach is suited for real-time environments such as VR environments or video games.

2. Related Work

Interactive simulation of liquids can be classified into surface- and volume-based techniques. The latter apply the Navier-Stokes-Equations in 3D to model the liquid's physical flow properties. In interactive computer graphics, the equations are typically solved using an efficient finite difference method [Sta99] or smoothed-particle hydrodynamics (SPH) [MCG03]. Adaptive couplings of 3D free surface simulations with 2D simulations have been presented to increase performance [TRS06, Cor07]. An SPH simulation coupled with a rigid-body simulation was presented in [Ama06]. GPU-based implementations of 3D simulations pushed the limit of virtual liquids realism [Har04, HKK07, CIS07]. The use of adaptive domain translation within a 3D fluid simulation is presented in [SCP*04]. Our work uses a domain translation within a 2D FDM simulation as well to reach an adaptive representation at high performance. Surface extrac-

tion using, for example, a marching cubes algorithm [LC87] is another bottleneck of such 3D simulations. [MSD07] presented a fast screen-space approach for 3D surfaces to overcome this problem. Since complexity of simulation and surface extraction of 3D liquids is high, only small amounts of liquid can be simulated in real-time. It is important to note that many liquid effects (e.g., rain or puddles) can be approximated realistically without an underlying physical simulation [TI06].

The wave equation supports simulating the propagation of 2D waves efficiently. Usually, it is solved by applying a finite difference method [Gom00], resulting in a height field that can be rendered efficiently. [JH03] described the coupling of a tiled ambient wave simulation and a wave equation simulation — the presented work is following this paradigm, creating an infinite environment. [YHK07] presented a new method for solving an approximation of the wave equation. Particles are used to simulate propagating waves with impressive results at high frame rates. They also present a coupling of rigid-body simulation and wave particles, resulting in a real-time boat simulation. Their work is closest to our method, yet the simulation is different. Due to the importance of their work, a direct comparison is given in Section 6. The wave particles method has been coupled with a two dimensional flow simulation in [Cor08]. Other approaches of height field-based liquid simulation are based on solving the Navier-Stokes Equations in 2D and constructing a height field from pressure [CdVLHM97]. [KM90] used an approximated shallow water equation, which results from the Navier-Stokes Equations, disregarding the vertical velocity of the liquid. [OH95] presented the creation of splashes within height field-based liquids. [TSS*07] presented the creation of bubbles and foam within a shallow water simulation. Tessendorf presented a technique for the animation of realistic ocean waves [Tes04b] and expanded it for wave creation of objects in [Tes04a], see also [Bel03, JG01]. We use the technique to generate ambient waves. [JG01] described an entire ocean environment, including the shallow water equation. Rigid-body dynamics within liquids may be determined by particle systems discretizing the rigid-body. Especially within height field-based liquids, the acting forces can be determined efficiently in this way (e.g., [DSH*06]).

Reflection and refraction effects in real-time environments are usually approximated by mapping techniques [SW01, Joh04, Sou05]. GPU-based raycasting of an environment that features two height field-based water and ground surfaces can be applied using [BD06].

3. Overview

Our goal is to create a realistic, yet interactive ocean-scale water environment with static objects and moving water crafts. Focus lies on the representation of surface waves. Therefore, we spend computational effort on the most visible property leading to more detailed results. We apply a FDM-based approach of the wave equation and create a dynamic simulation grid, using high grid resolutions. Although the simulation grid has discrete resolution, we focus on creating smooth animations. We devise an adaptive solution where simulations occur only in such areas where needed (see Figure 1). Another important benefit of a dynamic grid is that the grid can follow moving objects like boats. By placing a specific object at the grid's center, the wave creation in the object's surrounding area is always interactive. For example, when a camera follows a moving boat, detailed waves are created in it's neighborhood, while the infinite ocean surface with ambient waves outside of the simulation grid is not affected.

4. Simulation

Simulating large water environments in real-time with highly detailed waves everywhere is not feasible. Hence, we only *simulate* the liquid surface where fine detail is needed (e.g., in the wake of moving water crafts) and *animate* it in regions with no object–water interaction. Most ocean waves are created by wind and move independently. We classify waves into *interactive waves* when interaction with objects occurs and *ambient waves* otherwise

Ambient Waves For ambient wave animation, we use the effective method of Tessendorf [Tes04b]. This results in realistic wave propagation for open water. The model is based on statistical measurements of real ocean environments. The frequency spectrum is generated by the Phillips spectrum, supporting handling of wind direction and wind strength. Since this method generates tiled height fields, several fields with different aperiodic scales can be used to generate an infinite ocean surface, where the tiling is not visible.

Interactive Waves The wave equation describes the propagation of waves at time t and position $\mathbf{x} \in \mathbb{R}^2$. A 2D linear partial differential equation (PDE) can be used for simulating radial wave propagation of liquid surface waves:

$$\triangle f(\mathbf{x},t) - \frac{1}{c^2}\frac{\partial^2 f(\mathbf{x},t)}{\partial t^2} = 0. \qquad (1)$$

Here, $\triangle = \nabla^2 = \sum_1^2 \frac{\partial^2}{\partial x_i^2}$ is the Laplacian in 2D and c is the velocity at which waves propagate across the surface. This PDE can be solved in a fast and straightforward manner using an Eulerian grid-based approach. Boundary conditions have to be included for collision objects as well as for the boundary of the simulation grid. Such bounding conditions can be modeled within the wave equation directly. Adapting $f(\mathbf{x},t_i)$ to constant values at the boundary of collision objects within the liquid models objects and, hence, the corresponding reflection of waves. Radially propagating waves can be created at any position $\mathbf{x_0}$ and time t_i by displacing $f(\mathbf{x_0},t_i)$. Displacing several $f(\mathbf{x}_i,t_i)$ can create, for example, large waves or wave trains.

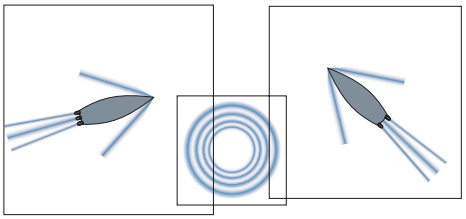


Figure 1: *Principle: Several simultaneous wave simulations are used within the infinite ocean surface.*

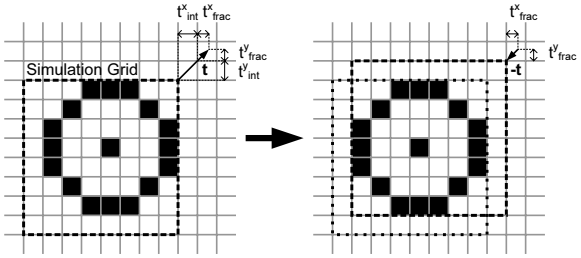


Figure 2: *The simulation grid is moved relative to the infinite ocean surface. Therefore, the translation vector* $\mathbf{t}$ *is separated into an integer and a fractional part.*

Additional maps that influence the simulation of the wave equation can be used to model specific behavior. In shallow water, for example, velocity depends on water depth. Hence, c can be varied according to depth and described by a velocity map. The same applies to damping and a damping map may describe the damping according to the type of ground.

Discrete wave equation The discretization of the wave equation (Equation 1) using a finite difference method can be carried out using a 2D map $z_{ij} = f(i,j)$, with $i,j \in [0,size]$ and step size $h = 1/size$. Using central differences, the discretization leads to [Gom00]:

$$z_{i,j}^{t+1} = a \cdot \sum_{\substack{k=i\pm1,l=j;\\ k=i,l=j\pm1}} z_{k,l}^{t} + (2-4a)\cdot z_{i,j}^{t} - z_{i,j}^{t-1}, \qquad (2)$$

with $a = \frac{c^2\Delta t^2}{h^2}$ and time step Δt. Hence, the wave equation is solved for a rectangular area.

For smooth movement of the simulation grid in surface space, we define a translation $\mathbf{t} \in \mathbb{R}^2$. Note that a translation with non-integer numbers would result in large numerical diffusion due to the necessary dissipation step. The translated grid elements has to be refiltered onto the discrete grid values within each time step, resulting in diverging wave trains. To overcome this problem, we split the translation $\mathbf{t}$ into the integral part $\mathbf{t}_{int} = (t_{int}^x, t_{int}^y)^T \in \mathbb{N}^2$ and the fractional part $\mathbf{t}_{frac} \in \mathbb{R}^2$ with $0 \leq \mathbf{t}_{frac}^x, \mathbf{t}_{frac}^y \leq 1$ (illustrated in Figure 2). Since the numeric solution of Equation 2 depends on the current and the previous time step, the integral translation $\mathbf{t}_{int}^{old}$ of the previous time step is stored for reuse. Consequently, the discrete wave equation is then changed to

$$z_{i,j}^{t+1} = a \cdot \sum_{\substack{k=m\pm1,l=n;\\ k=m,l=n\pm1}} z_{k,l}^{t} + (2-4a)\cdot z_{m,n}^{t} - z_{o,p}^{t-1}, \qquad (3)$$

with $m = i - t_{int}^x$, $n = j - t_{int}^y$, $o = m - t_{int}^{x\,old}$ and $p = n - t_{int}^{y\,old}$. The simulation grid is accessed in surface coordinates, adding the fractional translation $\mathbf{t}_{frac}$. Hence, numerical dissipation is suppressed, due to the use of a discrete grid. Transformation within the grid is carried out on integer basis. However, the grid can still be moved smoothly, because access of coordinates works on a fractional basis. To scale the surface in x or y direction by s_x and s_y, the translation vector is scaled by $(1./s_x, 1./s_y)^T$.

The described simulation method can be executed efficiently on a GPU. Using a 2D grid, the method can be implemented within a fragment shader using textures. A general purpose GPU framework, such as CUDA, can be used too. Yet the texture-based point of view lets us set the boundary conditions efficiently by rendering directly into the simulation grid.

Stability We use an explicit approach for solving the wave equation, we got the following stability constraints (e.g., [Gom00]) according to explicit finite difference simulation methods. The simulation becomes unstable, if condition $(c^2\Delta t^2)/(h^2) \leq 0.5$ is not adhered and hence, the height field grows exponentially (h: Gridstep). Yet, the maximum wave propagation speed can be determined by the given condition.

Multiple Grids The above method allows for placing the simulation grid at an arbitrary position on the infinite water surface. The grid can follow moving objects and the simulation grid can follow the view frustum. We describe the use of several simulation grids for adaptive simulation below.

Different Locations In some scenarios it may be of importance to simulate the surrounding of more than one object or the area within the view frustum only. Our method can be expanded to support multiple moving grids. Even more, each grid can be moved independently as depicted in Figure 1. Final height-field values can be determined by superposition of all overlapping grids. Simulation grid sizes are adapted according to the importance of the actual simulation. Hence, waves are simulated where necessary. As a result, we achieve a fast, environment specific simulation. For example, different moving boats can have their own simulation grid and, hence, full interaction with surrounding liquid. Simulation grid sizes may vary according to the importance of details within that grid.

Interference of waves is a fundamental physical principle and result in superposition of amplitudes (not intensities). Since waves are interfering without any interaction, different grids can be simulated independently — wave interaction

between different grids do not occur. Hence, objects in different grids have to be coupled to their belonging simulation grid only and the final height field can then be determined by superposition. Take note that the independence of simulation grids is important for efficient execution, since no data has to be transferred between simulation grids. Height field values are scaled down smoothly to zero at the boundaries of moving grids to guarantee a soft transition to the height values of ambient waves or interfering simulation grids. A static grid can be used, if the wave propagation within a static region is of importance.

Different Scales Our approach can not only be used for adaptive simulation at different locations, but also at different scales. Since multiple grids can be used, one grid can be used to represent high details if the camera comes close to the liquid's surface. For example, in a multiple grid simulation, one tiled grid simulates the detailed wave propagation of a rain drop hitting the water surface and another grid simulates the wave propagation of a moving boat (see Figure 1). If different scales are used, the simulation grid's translations have to be normalized: They are divided by their resolution — otherwise different grids are moving with different velocities.

Another possibility is the use as an LoD-Simulation. Depending on the distance to the camera, simulation grid size can be decreased, resulting in less details and faster simulations for objects far away.

Foam Foam is a highly complex and chaotic phenomena, which is important to increase visual realism of simulated liquid. We model two types of foam. A foam mapping represents the wave peak of a rendered liquid. Depending on the height of a grid element (i.e., water wave), the foam map's intensity is faded in [JG01] to increase realism. In addition to this method, we developed a boat wake state machine. It is described in Section 4.2.

4.1. Rigid-Body Dynamics

Visual realism does not depend on the liquid simulation alone, but also on the believability of object–liquid interaction. We separate interaction into (i) liquid→object and (ii) object→liquid interaction. The former describes the dynamics of floating objects and their reaction to liquid motion. The latter describes the liquid's reaction to moving boats or object impacts. We simulate both types independently.

Realistic motion of objects in water depends on standard rigid-body dynamics coupled with buoyancy forces and damping forces. In the ideal case, the damping forces are split into drag and lift forces. More more details we refer the interested reader to a fluid dynamics textbook. To achieve a real-time simulation, we apply an explicit integration scheme.

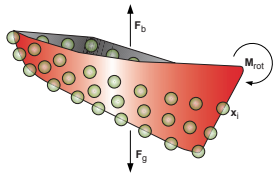


Figure 3: *Rigid-bodies objects such as boats are discretized using particles.*

4.1.1. Representation

We use particle systems for rigid-body physics. Each rigid-body object is discretized by a set of n particles with static positions $\mathbf{x}_i^0$ $(i = 1 \ldots n)$ relative to the center of mass $\mathbf{x}_c$. Dynamic motion is determined according to the translation of and orientation to $\mathbf{x}_c$. Each particle i of a rigid-body has the following position in world space at time t (see Figure 3): $\mathbf{x}_i^t = \mathbf{x}_c^t + \mathbf{M}_{rot}^t \mathbf{x}_i^0$., where $\mathbf{M}_{rot}^t$ describes the rotation of the rigid-body object. Each particle represents the surrounding mass of the rigid-body object and, thus, the buoyancy force is determined only for particles lying below the water surface: $(\mathbf{x}_i^t)^z < h((\mathbf{x}_i^t)^x, (\mathbf{x}_i^t)^y)$, where $h(x,y) : \mathbb{R}^2 \rightarrow \mathbb{R}$ describes liquid's height at position (x,y) and $\mathbf{x} = (\mathbf{x}^x, \mathbf{x}^y, \mathbf{x}^z)^T$. $\mathbf{M}_{rot}^t$ is updated within each time step Δt according to angular acceleration as follows: $\mathbf{M}_{rot}^t = \mathbf{M}_{rot}^{t-1} + \Delta\mathbf{M}_{rot}$, where $\Delta\mathbf{M}_{rot}$ is the rotation matrix of $\Delta t \cdot \|\mathbf{M}\|$ about the vector $\mathbf{M}/\|\mathbf{M}\|$ ($\mathbf{M}$: angular momentum). The momentum of a collection of particles is equal to the sum of angular momenta of each particle. Hence, $\Delta\mathbf{M}_{rot}$ can be determined by the sum of the angular accelerations of each particle. Hence, arbitrary object shapes with different physical parameters can be simulated.

Stability Since we use a relatively sparse discretization of rigid-body objects to achieve high performance, the discontinuity of buoyancy at the free surface can lead to instantaneous force changes if a particle is passing the free surface. To avoid such side effects, the buoyancy force is scaled linearly according to a threshold distance d_t under water, such that the buoyancy force reaches zero at the surface. Consequently, rigid-body movement is smooth – even for sparsely-sampled objects.

4.1.2. Collisions

Fast collision handling for rigid-bodies has been well studied in the field of computer graphics. Our approach is influenced by the observation that a complex and exact rigid-body collision handling is not required for the kind of floating objects in our system. In many cases (except for dropping elements), the problem is equivalent to the 2D case. We apply the following approximation for dynamic object collisions: according to the distance between different dynamic objects, a force is introduced that acts between them in opposite direction .

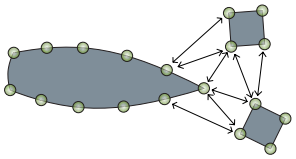


Figure 4: *Collision particles are introduced for an effective collision handling. Repulsive forces act between them.*

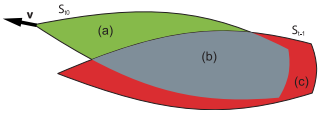


Figure 5: *Waves are generated by intersection of S_{t0} and S_{t-1} at the actual and the previous time step. Hence, bow waves are generated in (a) and stern waves in (c).*

We introduce collision particles $\mathbf{x}_{\text{col}}$ to each rigid-body object as illustrated in Figure 4. These particles are used to determine the distance d i.e., the collision forces $\pm\mathbf{F}_{\text{col}}$. The collision force is represented by a linear distance kernel. To avoid exhaustive tests between all collision particles, a standard grid-based orthogonal range search data structure can be used, that decreases complexity significantly. This method achieves fast and plausible results without any overlapping for moderate velocities, even though correct results are not guaranteed.

4.2. Boat Simulation

Since our method uses particles to model rigid-body objects, we can handle objects of arbitrary shape in our simulation. Figure 3 depicts a boat represented by particles. Obviously, our performance decreases with an increasing number of particles. To reduce the number of particles we only model the boat's hull. Since physics calculations are on a per-particle basis, this procedure results in realistic motion based on existing waves.

4.2.1. Wave Creation

Since we do not simulate liquid flow, wave creation from rigid-body objects can only occur when dropping or moving objects. In case of a dropping object, a wave with the shape of the intersection of object and surface is created while the object is intersecting the surface. To determine the starting time of a wave, each object knows its state s: (0) above water, (1) under water. The state is refreshed at each time step and a wave is created when the object begins to intersect the surface:

$$create\_wave = \Big[(s^{t-1} = 0) \wedge ((\mathbf{x}_c^t)^z < 0)\Big] \vee \Big[(s^{t-1} = 1) \wedge ((\mathbf{x}_c^t)^z > 0)\Big].$$

If *create_wave* is equal to one for a given rigid-body object, the object has intersected the water surface and a wave is created. Hence, the object's intersection with the surface plane is projected onto the simulation grid, with an intensity according to the intensity of the impact.

For creating waves caused by moving objects, we take advantage of the following physical properties. Pressure on an moving boat's hull is higher at its bow than at its stern. Thus, *positive* waves are created in front of the boat and *negative* ones at the back. The intensity of created waves depends on the speed of the boat. To determine the area that a boat is passing through within a time step, we combine the position and orientation of the current and the previous time steps. For each time step we generate a map containing the intersection of the boat's hull and the water surface. Since calculating the exact intersection area is too expensive, we make two assumptions: (i) we approximate the surface by a plane. Since ambient waves are usually small compared to the extent of the surface, this is a reasonable assumption. However, in scenarios with large ambient waves, this assumption becomes a limitation and may result in artifacts. In the final animation, the difference is hardly visible, since the boat is in water and the waves are created along the hull in addition to the ambient ones. (ii) we assume that every boat slice within $z = const., z < 0$ is a subset of the slice at $z = 0$. A slice is defined by the intersection of the polygonal boat and a plane with $z = const$. This assumption is valid for most water crafts, even for a catamaran. For more complex floating objects being pulled through the water, a more sophisticated approach has to be used. For example, the intersection contour could be extracted by culling according to the two planes $z = \pm\varepsilon$ Hence, even strongly dynamic objects with complex shape would create waves at their boundary layer intersection. After the texture of intersection S^t is created, it is combined with the corresponding texture from the previous time step S^{t-1}. Considering the textures as a set, three different wave creation states can be constructed (see Figure 5, $\setminus$: Difference.):

1. Bow wave: $S^t \setminus S^{t-1}$,
2. Body: $B^t = S^t \bigcap S^{t-1}$,
3. Stern wave: $S^{t-1} \setminus S^t$.

To create waves, wave intensities may be added to or subtracted from actual simulation grid values. Hence, the created waves interfere with existing ones, which is necessary if the simulated boat should interact with existing waves. If this effect is not required, the values within the simulation grid can simply be replaced. If they are replaced, they should be scaled slightly and translated accordingly to the front. Thus, artifacts at the boats contour are reduced and a bow wave is created. If the boats velocity is faster than wave propagation speed, the bow wave stays constant in front. According to reality, this area could be improved with splashing effects known from today's computer games.

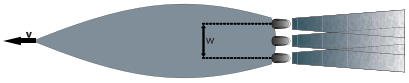


Figure 6: *Placement of engine waves and the foam generator (three engines are used in this example).*

Figure 7: *Interactive speedboat.*

4.2.2. Wave Reflection

If the boat's hull should reflect interactive waves, the generated body set B^t can be used to detect the boats position. If at rest, $S^t = B^t$, otherwise it results in the area where no waves are created. Hence, the simulation grid is filled with values of no displacement within the area of B^t. Thus, the boundary conditions are set according to the outer contour and waves are reflected correctly at the boat's hull. Take note, that the wave reflection cannot be used while wave creation is enabled, since both methods replace the grid values. This constraint may be acceptable though, because the wave reflection of a moving boat may hardly be visible.

4.2.3. Motor Boats: Engines and Wakes

So far, we have only considered boats without engine. Typically, a powerboat causes a distinct wake, which is more pronounced than the wake of a sailboat. Since we do not simulate flow, we reproduce the engine's wake generation by creating a wave at the given engine position. As many water crafts have more than one engine, we use symmetry and place the i-th engine wave generator at the subsequent y-position e_y according to the number of engines m and the boat's width w (see Figure 6): $e_y = -(w/2) + i \cdot (w/m) + (w/2m)$. The wave generator can generate different shapes. Still we found that a circular shape results in the most realistic waves. Size and intensity of the wave generator may depend linearly on the boat's velocity. Our approach for wake animation is based on the observation that boat wakes have, beside the chaotic and complex foam movement, a very typical yet simple shape. The amount of foam depends on the engine's speed. It decreases over time, and stays within the wake. We use a state machine to represent these characteristics. Within the engine's area new foam f is created $f \in [0,1]$ in a *foam map*. The foam map is updated at each time step. We use the following fading and smoothing formula, which is simple a filter function:

$$f_{ij} \rightarrow d \cdot f_{ij} + \frac{1-d}{4} \cdot (f_{i+1j} + f_{ij+1} + f_{i-1j} + f_{ij-1}),$$

where $d \in [0,1]$ describes diffusion and fading velocity. Figure 6 depicts the associated propagation of foam. The size of generated foam may vary depending on the boat's velocity as well. The state machine for representing foam can be used to create fading foam interactively at any position and with arbitrary shape within a simulation grid (e.g., if an object drops into water).

5. Surface Extraction & Rendering

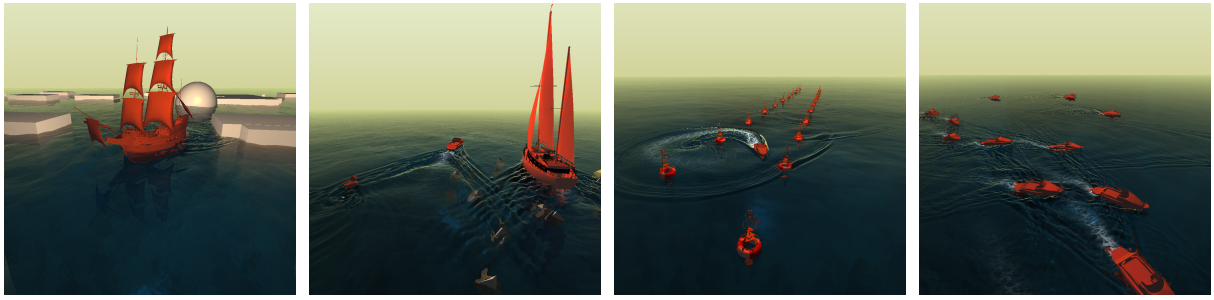
To combine multiple dynamic height fields we apply the projected grid method [HNC02, Joh04]. Our projected grid implementation is entirely GPU-based. Nevertheless, the surface extraction step is the most time consuming within our framework (see also Section 6). We determine normal vectors by combining overlapping simulation grids and calculate partial derivations by forward differencing. To reduce aliasing effects at the horizon resulting from the projected grid method, we adjust normals $\mathbf{n}$ according to normalized camera distance in z-direction c_d: $\mathbf{n} \rightarrow \|(n_x, n_y \cdot (1 + (c_d \cdot i)), n_z)^T\|$, where $i \geq 0$ is the smoothing intensity. Sometimes it is convenient to additionally smooth the height field or the associated normals.

After calculating normals, lighting and optical effects can be applied. In our implementation we use a combination of standard Phong lighting, standard cube mapping and reflection-/refraction-mapping techniques [Sou05, Joh04] to approximate reflections and refractions of objects within the liquid. Absorption of the liquid is simulated by the use of a view-dependent method: By subtracting the collision-object depth map from the depth map of the liquid's surface (both seen from camera position), the resulting map is a distance map of object–surface distances. The distance can be used with any absorption function fading out the object's color. Hence, the liquid exhibits more visual depth and is not unrealistically transparent.

6. Results & Discussion

The experiments presented in this paper ran on a quad-core desktop PC with a 2.4 GHz Intel Q6600 CPU, 4GB RAM, and nVidia GeForce GTX 280 GPU. Using only a *single* CPU core, the measurement times include simulations as well as surface generation and rendering. Results were generated at a screen resolution of 1024×1024 pixels. All examples include two wave-equation simulations with a resolution of 2048×2048 and 1024×1024 that were solved on the GPU with floating-point precision. The foam state machine has a grid size of 1024×1024. The accompanying video illustrates the visual quality and performance of our method.

Performance Frame rates for different projected grid step sizes of our unoptimized prototype are given in Table 1.



(a) (b) (c) (d)

Figure 8: *Examples: Corsair (a), 3 Boats (b), Buoys (c) and Armada (d).*

The distribution of calculation time according to the different steps are shown for projected grid steps 1×1 in Table 2. Measured frame rates allow full interactivity (see accompanying video), including the wave creation of cruising boats. Note that most of the time is spent for surface extraction and rendering, not for surface simulation, since it can be implemented very effective on the GPU. The cost of the projected grid method increases quadratically with viewport size. Hence, we have to trade off screen-space resolution and, thus, aliasing artifacts with performance. Another bottleneck of our method is the memory read back from GPU. Since rigid-body simulation is executed on the CPU, the local area surrounding each object has to be read back to be included within the rigid-body simulation. Wave generation could be accelerated by using low-resolution models of the relevant boat – this is a bottleneck of our specific implementation, reducing performance for many boats (see example *Armada* depicted in Figure 8d).

Quality The presented real-time method aims at large, but realistic liquid environments including liquid–object interaction. To achieve fast simulation times, we concentrate on the main visual property of liquids in those scenarios, which we tried to represent realistically: surface waves. Hence, our method is valid only in environments where liquid flow and 3D liquid effects can be disregarded. The method is view dependent and additional simulations can be carried out in important areas (e.g., the surroundings of moving ships) and different scales of surface simulations can be combined. But it should be noted that the 2D wave equation is physically only applicable to shallow water or constant-wavelength waves. Within deep ocean water wave propagation is more complex, due to dispersion. However, aiming at interactive environments, we believe this limitation is acceptable. Our rigid-body object and boat simulation method results in effective and realistic movements and detailed wave creation, even for multiple moving boats. Rivers or lakes can be simulated straight-forwardly by our method. Obviously, our model is not intended to replace existing high-quality offline techniques, but may be well-suited for real-time scenarios, such as computer games and virtual environments.

Comparison with Wave Particles A fundamental difference between our work and Wave Particles [YHK07] lies in the simulation method: instead of using wave particles to approximate the wave equation, we solve the wave equation using an FDM, which allows us to simulate Huygen's principle. In other words, using wave particles, an object hitting the water surface would cause a radially propagating wave front with a flat surface inside. Similarly, diffraction cannot be represented using particles as well. As demonstrated in the results, we can simulate the more detailed waves between wave fronts and diffraction, which cannot be reached with the wave particles method, principally. We believe that both effects improve the richness of the animation. Wave parti-

	1×1	2×2	5×5
Speedboat (3)	40.3	56.3	69.4
Corsair (3)	35.2	46.7	58.8
3 Boats (43)	32.2	43.0	56.8
Buoys (41)	26.6	34.9	40.9
Armada (40)	22.7	27.6	30.3

Table 1: *Time results in FPS for different projected grid spacings. The number of used rigid-bodies within each simulation is indicated within brackets.*

	RB	LF	R	Total
Speedboat	1.8	35.1	63.1	100.0
Corsair	2.0	29.3	68.7	100.0
3 Boats	2.7	23.7	73.6	100.0
Buoys	12.3	23.6	64.1	100.0
Armada	1.3	36.0	62.7	100.0

Table 2: *Distribution of calculation time (in percentages). Results for projected grid spacing* 1×1. *Rigid-body simulation (RB), liquid and foam simulation (LF) and surface extraction and rendering (R).*

cles are very fast, but computational cost increases with the number of waves, as more particles are needed to represent additional wavefronts. The grid size of our FDM approach does not depend on the number of waves, since the whole grid is refreshed each time step. Performance of our method can be improved by decreasing simulation sized (which are large in our examples) and projected grid size.

7. Conclusion

We presented a scheme for simulating interactive large water environments including moving boats. The liquid simulation environment is designed to be executed on modern GPUs. We demonstrate that a simplified approach can achieve fast and realistic wave creation. We introduced an extension of the wave equation to simulate large, apparently infinite water environments. Therefore, we introduced independently moving simulation grids. The presented method achieves real-time performance, is highly scalable and allows for full interaction between objects and liquid surfaces. The presented model of moving boats produces realistic and detailed waves, as we have demonstrated with several test cases.

Future work will include the coupling with a 3D Navier-Stokes-based flow simulation in areas where necessary. Hence, additional realistic 3D liquid effects could be created — e.g., splashing and flow caused by a boat's engine. Additionally, we would like to improve performance of rigid-body simulation by using the GPU. (e.g., [Har07]).

References

[Ama06] AMADA T.: Real-time particle based fluid simulation with rigid body interaction. In *Game Programming Gems 6* (2006), Charles River Media, pp. 189–205.

[BD06] BABOUD L., DECORET X.: Realistic water volumes in real-time. In *EG Workshop on Natural Phenomena* (2006).

[Bel03] BELYAEV V.: Real-time simulation of water surface. In *GraphiCon-2003* (2003), pp. 131–138.

[CdVLHM97] CHEN J. X., D. V. LOBO N., HUGHES C. E., MOSHELL J. M.: Real-time fluid simulation in a dynamic virtual environment. *IEEE Comput. Graph. Appl. 17*, 3 (1997), 52–61.

[CIS07] CRANE K., I.LLAMAS, S.TARIQ: Real-time simulation and rendering of 3D fluids. In *GPU Gems 3* (2007), Addison-Wesley, pp. 633–673.

[Cor07] CORDS H.: Mode-splitting for highly detailed, interactive liquid simulation. *Proc. of GRAPHITE* (2007), 265–272.

[Cor08] CORDS H.: Moving with the flow: Wave particles in flowing liquids. *Journals of WSCG* (2008).

[DSH*06] DOBASHI Y., SATO M., HASEGAWA S., YAMAMOTO T., KATO M., NISHITA T.: A fluid resistance map method for real-time haptic interaction with fluids. In *VRST'06* (2006).

[Gom00] GOMEZ M.: Interactive simulation of water surfaces. In *Game Programming Gems* (2000), Charles River Media, pp. 187–195.

[Har04] HARRIS M.: Fast fluid dynamics simulation on the GPU. In *GPU Gems* (2004), Charles River Media Graphics.

[Har07] HARADA T.: Real-time rigid body simulation on GPUs. In *GPU Gems 3* (2007), Addison-Wesley, pp. 611–632.

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Smoothed particle hydrodynamics on GPUs. In *Computer Graphics International* (2007), pp. 63–70.

[HNC02] HINSINGER D., NEYRET F., CANI M.-P.: Interactive animation of ocean waves. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)* (july 2002).

[JG01] JENSEN L., GOLIAS R.: Deep-water animation and rendering. In *Game Developer's Conference (Gamasutra)* (2001).

[JH03] JAMES G., HARRIS M.: Simulation and animation using hardware accelerated procedural textures. In *Game Developers Conference Tutorial* (2003).

[Joh04] JOHANSON C.: Real-time water rendering - introducing the projected grid concept. In *Master of Science Thesis (Lund University)* (2004).

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *Proc. of the 17th annual conf. on Computer graphics and interactive techniques* (1990), pp. 49–57.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87* (1987), vol. 21, ACM.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of Symposium on Computer animation* (2003).

[MSD07] MÜLLER M., SCHIRM S., DUTHALER S.: Screen space meshes. In *Proceedings of Symposium on Computer animation* (2007), Eurographics Association, pp. 9–15.

[OH95] O'BRIEN J. F., HODGINS J. K.: Dynamic simulation of splashing fluids. In *Computer Animation* (1995).

[SCP*04] SHAH M., COHEN J. M., PATEL S., LEE P., PIGHIN F.: Extended galilean invariance for adaptive fluid simulation. In *Proc. of SCA'04* (2004).

[Sou05] SOUSA T.: Generic refraction simulation. In *GPU Gems 2* (2005), Addison-Wesley, pp. 295–305.

[Sta99] STAM J.: Stable fluids. In *Proc. of SIGGRAPH* (1999).

[SW01] SCHNEIDER J., WESTERMANN R.: Towards real-time visual simulation of water surfaces. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001* (2001), Aka GmbH, pp. 211–218.

[Tes04a] TESSENDORF J.: Interactive water surfaces. In *Game Programming Gems 4* (2004), Charles River Media.

[Tes04b] TESSENDORF J.: Simulating ocean water. In *Course Notes Siggraph: The Elements of Nature: Interactive and Realistic Techniques (Course 31)* (2004).

[TI06] TATARCHUK N., ISIDORO J.: Artist-directable real-time rain rendering in city environments. In *Eurographics Workshop on Natural Phenomena* (2006), ACM Press.

[TRS06] THUEREY N., RÜDE U., STAMMINGER M.: Animation of open water phenomena with coupled shallow water and free surface simulation. *Proc. of the 2006 EG/ACM SIGGRAPH Symposium on Computer Animation* (2006), 157–166.

[TSS*07] THUEREY N., SADLO F., SCHIRM S., MUELLER M., GROSS M.: Real-time simulations of bubbles and foam within a shallow-water framework. In *Proc. Symposium on Computer Animation* (2007).

[YHK07] YUKSEL C., HOUSE D. H., KEYSER J.: Wave particles. *ACM Transactions on Graphics 26*, 3 (2007), 99–107.