Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)
M. Otaduy and Z. Popovic (Editors)

Real-time Simulation of Large Bodies of Water with Small Scale Details

Nuttapong Chentanez and Matthias Müller

NVIDIA Corporation

Abstract

We present a hybrid water simulation method that combines grid based and particles based approaches. Our specialized shallow water solver can handle arbitrary underlying terrain slopes, arbitrary water depth and supports wet-dry regions tracking. To treat open water scenes we introduce a method for handling non-reflecting boundary conditions. Regions of liquid that cannot be represented by the height field including breaking waves, water falls and splashing due to rigid and soft bodies interaction are automatically turned into spray, splash and foam particles. The particles are treated as simple non-interacting point masses and they exchange mass and momentum with the height field fluid. We also present a method for procedurally adding small scale waves that are advected with the water flow. We demonstrate the effectiveness of our method in various test scene including a large flowing river along a valley with beaches, big rocks, steep cliffs and waterfalls. Both the grid and the particles simulations are implemented in CUDA. We achieve real-time performance on modern GPUs in all the examples.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling, Physically Based Modeling—Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism, Animation—Simulation and Modeling [I.6.8]: Type of Simulation, Animation—

Keywords: natural phenomena, physically based animation

1. Introduction

In recent years, physically based effects have become more and more popular in computer games. Features that used to be hand-animated are now driven by simulators. The most important examples are rigid bodies and particles. However, in most cases, large bodies of water such as rivers, lakes or oceans are still animated using procedural approaches. The reason for this is the large simulation domain in these scenarios. To get details on the surface of a lake using a full 3D fluid simulation would require millions of grid cells in the Eulerian case or millions of particles if an SPH approach is used. For real-time applications such as computer games, a viable simulation approach is to reduce the problem from 3D to 2D by treating the body of water as a 2D height field. This reduces simulation time substantially but comes at a price. Phenomena like waterfalls or overturning waves cannot be represented by a 2D height field. We solve this problem by turning height field water into particles automatically based on certain criteria that depend on the shape of the surface. These particles are turned back into height field water when they hit the water surface.

A simple and quite popular technique to simulate water surfaces is to solve the 2D wave equation. This approach works well for local phenomena like puddles or waves around a boat. The main drawback of this mathematical model, however, is that it only works with a vertical velocity field. Therefore, effects that are based on horizontal motion such as whirlpools or the flow of a river cannot be treated correctly. This is the reason we decided to use the shallow water equations for our simulations. The presence of a horizontal velocity field allows us to correctly advect floating objects and foam particles on the surface.

In the following sections we will show how various techniques can be put together to create a comprehensive system for simulating large bodies of water with small scale detail in real-time. This paper is not a system paper in the traditional sense though because we did not simply combine

DOI: 10.2312/SCA/SCA10/197-206

well known components but came up with various new approaches to solve the problems that arise when attempting to simulate a realistic water scenario with all its large and small scale features.

The main contributions of this paper are:

- Various improvements to a basic shallow water solver that enhance stability and allow the handling of arbitrary terrain slopes and arbitrary water depth.
- Criteria for automatically detecting surface discontinuities such as waterfalls and a method to turn the liquid volume in such locations into particles that carry mass and momentum of the height field across the discontinuity.
- A technique to automatically create and advect small scale FFT-based ripples on the water surface.
- A new and robust criterion to detect breaking waves and a method to turn height field fluid into particles creating spray, splash and foam.
- A way to absorb waves at the boundary of the simulation domain for handling open water scenes.

1.1. Related Work

Procedural methods for generating water animations have been used since the early days of computer graphics [FR86, Pea86]. There are several papers that describe how to use the spectral method for modeling ocean waves based on the Fast Fourier Transformation (FFT) in order to create animations of large bodies of water [MWM87, Tes99, TDG00, HNC02]. Although these methods are well suited to generate high resolution and large scale water animations, they cannot model the interaction with solid objects easily and are unable to generate vortices. With the exception of [Pea86] the separation of liquid from the main body of water has not been considered in these papers.

In computer graphics, [KM90] and [Tes99] were among the first to simulate water surfaces by solving the wave equation with internal boundaries on a 2D height field. Many researchers have used a pipe model where adjacent cells are connected by pipes that allow water to flow across cell boundaries. O'Brien and Hodgins [OH95] extended the pipe model by adding splash particles and modeling rigid body interaction with the water. Mould and Yang [MY97] later added bubbles and modeled droplets. Holmberg and Wünsche [HW04] used a weir model to allow the simulation of waterfalls. The method was later accelerated by a GPU implementation in [MFC06]. The pipe model was also used to simulate hydraulic erosion of terrains [vBBK08]. The authors produced impressive results by accelerating the simulation on GPUs. Recently, [YHK07] introduced a novel approach to approximate the solution of the wave equation by storing wave trains on 2D particles. Multiple shifting grids are used for simulating open water scenes by Cords and Staadt [CS09]. The main drawback of methods based on the wave equation or the pipe model is that vortices which are responsible for many interesting water phenomena such as swirling foam and whirlpools are not present in the model.

The Shallow Water Equations (SWE) can capture these phenomena because in addition to the height field, they describe the evolution of a 2D velocity field normal to the water columns. The SWE were introduced to computer graphics in [LvdP02]. The approach is based on the idea of assuming linear vertical pressure profiles in the 3D Navier-Stokes equations [CL95]. Hagen et al. [HHL*05] used a finite volume method for solving the SWE on the GPU to simulate water flowing on irregular terrains. As height field fluid approaches neither the wave equation nor the SWE can handle breaking waves. [TMFSG07] proposed a method to simulate breaking waves by automatically generating triangle mesh patches and evolving them. Their method cannot be parallelized easily though. [WMT07] added surface tension forces to the SWE to simulate water flow on arbitrary surfaces. Later, [ATBG08] used an implicit Newmark integration scheme to reduce numerical dissipation in the velocity advection step. To simplify the solver, [LO07] ignored the divergence term and used a collocated grid. Hess et al. [Hes07] integrated the SWE explicitly and described methods for two way water - rigid body coupling and wet-dry region tracking. The grid based portion of our proposed method also employs the SWE with several novel enhancements. Other approaches to model larger bodies of water such as rivers include the use of stream functions in [YNBH09] and the use of the 2D Navier-Strokes equations coupled loosely with the pipe model in [BAB09]. They advect particles that store texture coordinates for adding small scale detail to the water.

There are many works on the full 3D simulation of water which are beyond the scope of this paper. We wholeheartedly refer the reader to the book by Bridson [Bri08] for grid based simulation, and to a paper by Solenthaler and Pajarola [SP09] and references therein for particles based simulation.

2. Methods

We will state all the steps explicitly in full algorithmic detail. The goal is to provide enough information for the reproduction of our results. To get an overview of our method, the reader can skip the formulas and concentrate on the descriptions in the text.

Algorithm 1 Main loop (one time step)

1. Height field fluid simulation (Section 2.1)
2. Solids simulation (any standard simulator)
3. Two-way coupling of height field and solids(Section 2.3)
4. Particles generation and simulation (Section 2.4)
5. Rendering (Section 2.5)

Our main loop is summarized in Algorithm 1. We first simulate the height field fluid. Gravity, external forces and boundary conditions are taken into account as discussed in Section 2.1. Then, we simulate the solids such as rigid and soft bodies using a standard simulation package. We then couple the solids and the height field fluid as discussed in Section 2.3. Next, forces that the fluid exerts on the solids such as buoyancy, drag and lift are computed and applied. We then modify the fluid's height field and velocity field to take the movement of the solids into account. After that, we generate particles to replace regions of the liquid that the height field cannot resolve well. We then simulate the movement of the particles and eliminate them if they fall back into the height field (see Section 2.4). Finally, we render the height field with an additional displacement map that represents small waves with wave lengths below the resolution of the simulation grid. The splash particles are rendered with a screen-space technique as explained in Section 2.5

2.1. Height Field Fluid Simulation

We employ the Shallow Water Equations (SWE) which simplify the full Navier-Stokes equations to a 2D height field representation of the liquid surface. We assume gravity to act along the y-axis. Hence the plane for the 2D simulation is the x-z plane. In the following equations, h is the depth of the water, H is the y-coordinate of the terrain on the bottom, $\eta = H + h$ is the y-coordinate of the water surface, $\mathbf{v} = (u, w)$ is the horizontal velocity of the fluid, g is gravity and $\mathbf{a}_{\text{ext}}$ is an external acceleration. The shallow water equations can be written as

$$\frac{Dh}{Dt} = -h\nabla \cdot \mathbf{v}, \tag{1}$$

$$\frac{D\mathbf{v}}{Dt} = -g\nabla\eta + \mathbf{a}^{\text{ext}}, \tag{2}$$

where D is the material derivative operator. The equations describe conservation of mass and conservation of momentum. A detailed derivation of the SWE can be found in [Bri05]. Several authors [LvdP02], [WMT07], [ATBG08] solved the SWE using implicit methods. This guarantees unconditional stability. However, they either require an iterative method which incurs a significant run time cost or need to pre-factorize the matrix which means boundary conditions cannot be changed. We therefore use explicit integration and handle the terms that can lead to instability specially.

We discretize the simulation domain with a staggered grid where the heights $h_{i,j}$ and $H_{i,j}$ are stored at the cell centers and the velocities components $u_{i+\frac{1}{2},j}, w_{i,j+\frac{1}{2}}$ on faces following [Bri05]. We employ a time-splitting technique by first solving the self advection of the velocity field and then integrating the height field and velocity field forward in time. Throughout this paper we use meter (m) for distance and second (s) for time as our units. The grid spacing and time steps are denoted by Δx and Δt, respectively.

2.1.1. Velocity Advection

We solve the advection of $u_{i+\frac{1}{2},j}, v_{i,j+\frac{1}{2}}$ using an unconditionally stable modified MacCormack method as in [SFK*08] and fall back to the semi-Lagrangian method if the resulting value is not within the bounds of the velocity values used for bilinear interpolation of the first semi-Lagrangian sub-step as suggested in [SFK*08].

2.1.2. Height Integration

Equation 1 can be re-written as

$$\frac{\partial h}{\partial t} = -\nabla \cdot (h\mathbf{v}), \tag{3}$$

and discretized it to get

$$\frac{\partial h_{i,j}}{\partial t} = -\left(\frac{(\bar{h}u)_{i+\frac{1}{2},j} - (\bar{h}u)_{i-\frac{1}{2},j}}{\Delta x} + \frac{(\bar{h}w)_{i,j+\frac{1}{2}} - (\bar{h}w)_{i,j-\frac{1}{2}}}{\Delta x}\right), \tag{4}$$

where $\bar{h}$ is h evaluated in the upwind direction

$$\bar{h}_{i+\frac{1}{2},j} = \begin{cases} h_{i+1,j} & \text{if } u_{i+\frac{1}{2},j} \le 0 \\ h_{i,j} & \text{if } u_{i+\frac{1}{2},j} > 0, \end{cases} \tag{5}$$

and

$$\bar{h}_{i,j+\frac{1}{2}} = \begin{cases} h_{i,j+1} & \text{if } w_{i,j+\frac{1}{2}} \le 0 \\ h_{i,j} & \text{if } w_{i,j+\frac{1}{2}} > 0. \end{cases} \tag{6}$$

We integrate the height explicitly using

$$h_{i,j} \mathrel{+}= \frac{\partial h_{i,j}}{\partial t}\Delta t. \tag{7}$$

This guarantees mass conservation for the height integration step. We also found that it yields a more stable simulation than the choice of $\bar{h}_{i+\frac{1}{2},j} = \frac{h_{i,j}+h_{i+1,j}}{2}$ and $\bar{h}_{i,j+\frac{1}{2}} = \frac{h_{i,j}+h_{i,j+1}}{2}$ of [Hes07] in practice. The height integration step will be modified to take the discontinuity due to waterfalls into account in Section 2.4.3.

2.1.3. Velocity Integration

The face velocities are updated by taking the gradient of the water height into account as follows:

$$u_{i+\frac{1}{2},j} \mathrel{+}= \left(\frac{-g}{\Delta x}(\eta_{i+1,j} - \eta_{i,j}) + a_x^{\text{ext}}\right)\Delta t, \tag{8}$$

$$w_{i,j+\frac{1}{2}} \mathrel{+}= \left(\frac{-g}{\Delta x}(\eta_{i,j+1} - \eta_{i,j}) + a_z^{\text{ext}}\right)\Delta t. \tag{9}$$

This step will also be modified in Section 2.4.3.

2.1.4. Boundary Conditions

For a face that is marked by the user as reflective such as walls or permanent static obstacles, the velocity value is always set to 0 at the end of every time step and is not updated in the advection step. Moreover, we treat a face $(i + \frac{1}{2}, j)$ as reflective if either of the following is true:

- $h_{i,j} \leq \varepsilon$ and $H_{i,j} > \eta_{i+1,j}$ or
- $h_{i+1,j} \leq \varepsilon$ and $H_{i+1,j} > \eta_{i,j}$,

where $\varepsilon > 0$ is a small constant. We use $10^{-4}\Delta x$ in all examples. A similar condition applies to a face $(i, j+\frac{1}{2})$. Conceptually, the water level in a wet cell needs to be higher than the terrain level in the neighboring dry cell before the flow starts. Otherwise, the face behaves as a wall.

To simulate open water scenes, borders that absorb all incoming waves are needed. We adapt the Perfectly Matched Layers (PMLs) method [SM09], [Joh08] with a few modifications to take the rest water level into account and to improve stability. The idea is to define a damping region near the border of the domain and make waves within that region decay fast enough such that their amplitudes are negligible when they hit the border. To damp out the waves that move along the x-axis we evolve a field ϕ, defined only in the damping region. The field is initialized to 0 and then used to update h and u as

$$h_{i,j} \mathrel{+}= (-\sigma_{i,j}(h_{i,j} - h_{\text{rest}}) + \phi_{i,j})\Delta t, \quad (10)$$

$$u_{i+\frac{1}{2},j} \mathrel{+}= -\frac{1}{2}(\sigma_{i+1,j} + \sigma_{i,j})u_{i+\frac{1}{2},j}\Delta t, \quad (11)$$

$$\phi_{i,j} \mathrel{+}= -\lambda_{\text{update}}\sigma_{i,j}\frac{(w_{i,j+\frac{1}{2}} - w_{i,j-\frac{1}{2}})}{\Delta x}\Delta t, \quad (12)$$

$$\phi_{i,j} \mathrel{\times}= \lambda_{\text{decay}}, \quad (13)$$

where the constant $\sigma_{i,j}$ controls how fast the wave's amplitudes decay, h_{rest} is the depth of the water at rest and $0 < \lambda_{\text{decay}} < 1$ and $0 < \lambda_{\text{update}} < 1$ are constants to help improve the stability of the explicit Euler time integration. As suggested in [Joh08], $\sigma_{i,j}$ should increase quadratically or cubically from the inner edge of the damping region to the domain border. Large $\sigma_{i,j}$ will result in a faster decay and allow narrower damping regions but may introduce numerical reflection when a wave is about to enter the region. We use $\lambda_{\text{decay}} = 0.9$ and $\lambda_{\text{update}} = 0.1$ for all the examples. Waves moving in another directions are damped by the above method with an attenuation factor proportional to the cosine of the angle of the direction with the x-axis [Joh08]. To sufficiently damp the waves moving in all directions, a similar formulation is used for the z-axis, see Appendix 3.1. In all of our examples that require it, the width of the damping region is 10 cells. An example scene showing the PML is shown in Figure 1. The top and the bottom borders are absorbing z-moving waves while the left and the right borders are reflective. Notice how waves reflect only from the left and right borders. The technique is used in the boat riding scene in Figure 8 and in the open ocean scene shown in Figure 2.

2.1.5. Stability Enhancements

We propose several measures to improve the stability of the simulation without losing much visual quality. Due to numerical error, $h_{i,j}$ can become smaller than zero which is not physical and can cause stability issues. Therefore, we clamp $h_{i,j}$ to always be ≥ 0.

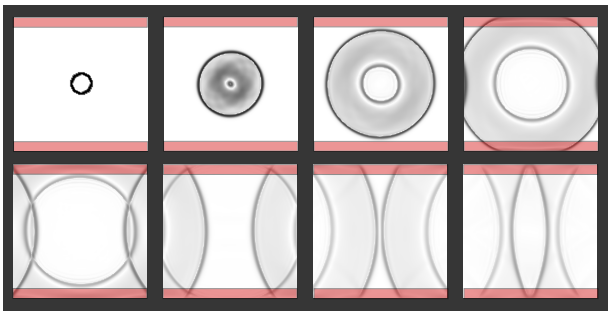


Figure 1: *A domain with a reflecting boundary condition on left and right, and absorbing region on the top and bottom. The absorbing regions are highlighted in pink.*

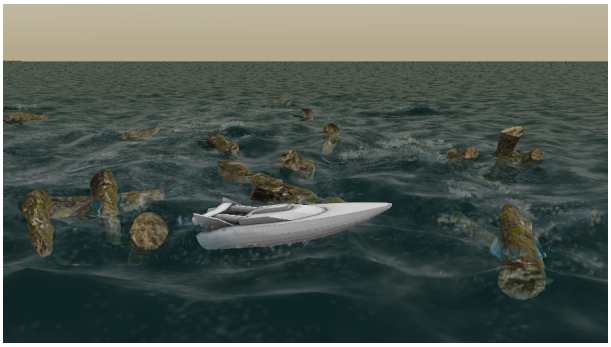


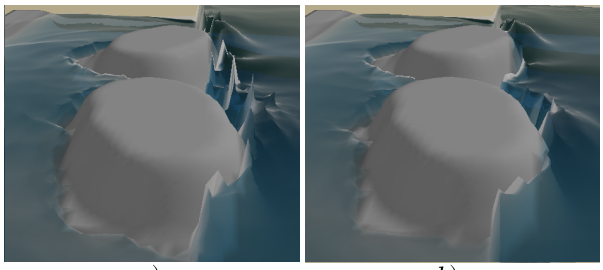
Figure 2: *An open ocean scene using an absorbing layer around the SWE domain which is surrounded by an FFT based ocean simulation.*

We also clamp the magnitudes of $u_{i+\frac{1}{2},j}$ and $v_{i,j+\frac{1}{2}}$ to be less than $\alpha\frac{\Delta x}{\Delta t}$, where we use $\alpha = 0.5$ in all examples. This places a limit on the wave speed within the simulation, but greatly improves the stability for scenes with violent water.

Additionally, we limit the water depth used for the height integration by replacing $\bar{h}$ in Equation 4 by $\bar{h} - h_{\text{adj}}$, where $h_{\text{adj}} = \max(0, \frac{\bar{h}_{i+1,j}+\bar{h}_{i-1,j}+\bar{h}_{i,j+1}+\bar{h}_{i,j-1}}{4} - h_{\text{avgmax}})$ and $h_{\text{avgmax}} = \beta\frac{\Delta x}{g\Delta t}$. We use $\beta = 2.0$ in all examples. This places a limit on the wave lengths and amplitudes in deep water regions, but improves the stability of the simulation while still produce plausible results.

2.2. Overshooting Reduction

When a wave from deeper waters enters a shallower region, its amplitude increases and the wavelength decreases. Eventually, the wave lengths become smaller than Δx and cannot be resolved by the grid anymore but become numerical ripples instead. The use of explicit integration makes the situation worse as it tends to amplify the edge of the first wave



a) b)

Figure 3: *a) Overshooting artifacts on top of breaking waves. b) Artifacts reduced by our technique.*

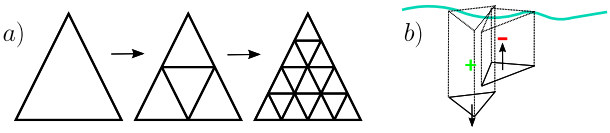


Figure 4: *a) Division of triangles into similar triangles b) Estimating the submerged volume by adding/subtracting the prism between downward/upward facing triangles and the water surface.*

front that enters the shallower water region. This manifests itself as overshooting artifacts in the simulation result, as shown in Figure 3a. To reduce the problem, we detect edges of these waves and reduce their magnitude by checking

- if $\eta_{i,j} - \eta_{i-1,j} > \lambda_{\text{edge}}$ and $\eta_{i,j} > \eta_{i+1,j}$, then $h_{i,j} \mathrel{+}= \alpha_{\text{edge}}(\max(0, \frac{1}{2}(h_{i,j} + h_{i+1,j})) - h_{i,j})$.
- if $\eta_{i,j} - \eta_{i+1,j} > \lambda_{\text{edge}}$ and $\eta_{i,j} > \eta_{i-1,j}$, then $h_{i,j} \mathrel{+}= \alpha_{\text{edge}}(\max(0, \frac{1}{2}(h_{i,j} + h_{i-1,j})) - h_{i,j})$.

We use $\lambda_{\text{edge}} = 2\Delta x$ and $0.1 < \alpha_{\text{edge}} < 0.5$ in our examples. A similar fix is used to waves moving along the z-axis. This reduces the overshooting for waves traveling in other directions as well, albeit less effective. Notice how the overshooting artifact is reduced in Figure 3b.

2.3. Two-Way Coupling of the Height Field with Solids

In this section we describe how we handle the coupling of rigid and soft bodies with the height field fluid. For all the triangles of the rigid bodies and the soft bodies we recursively divide each triangle into smaller similar sub-triangles until the area of these sub-triangles falls below $\kappa\Delta x^2$ (see Figure 4a). With decreasing κ the force computation becomes more accurate but also more expensive. We use $\kappa = 1$ in all our examples.

Let $\mathbf{p} = [p_x, p_y, p_z]^T$ and $\mathbf{v} = [v_x, v_y, v_z]^T$ be the position and velocity of the centroid of a sub triangle with area A. $\mathbf{p}$ and $\mathbf{v}$ are obtained by barycentric interpolation from the corresponding original triangle. Let $\mathbf{n} = [n_x, n_y, n_z]^T$ be the normal vector of the triangle. Throughout the section, $\hat{\mathbf{y}}$ denotes the unit vector $[0,1,0]^T$.

2.3.1. Fluid to Solids

We model the forces that the fluid exerts on a solid by taking three major components into account namely buoyancy, drag and lift forces. Buoyancy is an upward pointing force proportional to the weight of the displaced fluid. It can be computed as $-g\rho V$, where V is the volume of the object below the water surface. V is estimated simply by adding and subtracting the volumes of the prisms formed by projecting the triangle along the y-axis to the water surface. Volume is added for the downward facing triangles and subtracted for the upward facing triangles, as shown in Figure 4b.

The contribution of a sub triangle to the buoyancy force is hence

$$\mathbf{f}^{\text{bouyancy}} = \begin{cases} 0 & \text{if } \eta(\mathbf{p}) < p_y \\ g\rho A(\eta(\mathbf{p}) - p_y)n_y\hat{\mathbf{y}} & \text{otherwise,} \end{cases} \tag{14}$$

where $\eta(\mathbf{p})$ is the water level at $\mathbf{p}$, evaluated by a bi-linear interpolation.

The drag and lift forces are computed as in [YHK07],

$$\mathbf{f}^{\text{drag}} = -\frac{1}{2}C_D A^{\text{eff}}|\mathbf{v}_{\text{rel}}|\mathbf{v}_{\text{rel}}, \tag{15}$$

$$\mathbf{f}^{\text{lift}} = -\frac{1}{2}C_L A^{\text{eff}}|\mathbf{v}_{\text{rel}}|(\mathbf{v}_{\text{rel}} \times \frac{\mathbf{n} \times \mathbf{v}_{\text{rel}}}{|\mathbf{n} \times \mathbf{v}_{\text{rel}}|}), \tag{16}$$

where C_D and C_L are the drag and lift coefficients, $\mathbf{v}_{\text{rel}} = \mathbf{v} - \mathbf{v}_{\text{fluid}}$ is the relative velocity of the sub-triangle with respect to the fluid. $\mathbf{v}_{\text{fluid}} = [u, u\frac{\partial \eta}{x} + w\frac{\partial \eta}{z}, w]^T$. A^{eff} is the effective area of the sub triangle, which depends on the overall structure of the solid. It is computed as

$$A^{\text{eff}} = \begin{cases} 0 & \text{if } \eta(\mathbf{p}) < p_y \text{ or } \mathbf{n} \cdot v_{\text{rel}} < 0 \\ (\frac{\mathbf{n} \cdot \mathbf{v}_{\text{rel}}}{|\mathbf{v}_{\text{rel}}|}\omega + (1-\omega))A & \text{otherwise,} \end{cases} \tag{17}$$

where $0 \leq \omega \leq 1$ is a user defined parameter for adjusting the effective area.

We then add the force $\mathbf{f}^{\text{bouyancy}} + \mathbf{f}^{\text{drag}} + \mathbf{f}^{\text{lift}}$ at position $\mathbf{p}$ to the corresponding solid the sub-triangle belongs to. In the case the solid is a soft body, we distribute the force to the three vertices of the original triangle weighted by the barycentric coordinates.

2.3.2. Solids to Fluid

We modify the height and the velocity of the fluid due to solids using Algorithm 2.

During a time step, a fast moving object may pass through many grid cells. In that case the changes to the grid cells and faces are applied to all the cells along the path (lines 4 to 15). We also reduce the effect of solids on the fluid exponentially

Algorithm 2 Solids to Fluid

1: $num\_substeps = \max(1, \lfloor |\mathbf{v} - v_y \hat{\mathbf{y}}| \frac{\Delta t}{\Delta x} + 0.5 \rfloor)$
2: $V_{\text{disp}} = \mathbf{n} \cdot \mathbf{v}_{\text{rel}} A \Delta t$
3: $sign = (n_y > 0)?1 : -1$
4: **for** $q = 1$ to $num\_substeps$ **do**
5: $\quad \mathbf{p}_s = \mathbf{p} + \mathbf{v} \frac{q\Delta t}{num\_substeps}$
6: $\quad (i, j) = closest\_gridpoint(\mathbf{p})$
7: $\quad depth = \eta_{i,j} - p_y$
8: $\quad$ **if** $depth > 0$ **then**
9: $\quad\quad decay = e^{\lambda(-depth)}$
10: $\quad\quad h_{i,j} \mathrel{+}= decay \frac{V_{\text{disp}}}{num\_substeps(\Delta x)^2} C_{\text{dis}}$
11: $\quad\quad coeff = \min(1.0, decay C_{\text{adapt}} \frac{depth}{\eta_{i,j}} sign \frac{\Delta t}{(\Delta x)^2} A)$
12: $\quad\quad u_{i+\frac{1}{2},j} \mathrel{+}= coeff(v_x - u_{i+\frac{1}{2},j})$
13: $\quad\quad v_{i,j+\frac{1}{2}} \mathrel{+}= coeff(v_z - v_{i,j+\frac{1}{2}})$
14: $\quad$ **end if**
15: **end for**

with increasing distance to the surface (line 9). The decay rate $\lambda > 0$ is set to 1 in all our examples. $C_{\text{dis}} > 0$ and $C_{\text{adapt}} > 0$ controls how much the solid effects the fluid height and velocity respectively. We use $C_{\text{dis}} = 1.0$ and $C_{\text{adapt}} = 0.2$ in all examples. The height change depends on the amount of volume that the triangle sweeps through the fluid during that sub time-step, $\frac{V_{\text{disp}}}{num\_substeps}$, divided by the area of the cell, $(\Delta x)^2$, see line 10.

2.4. Particle Generation and Simulation

We employ three types of particle in our simulator: spray, splash and foam. Spray and splash particles represent parts of liquid that break free from the height field. Spray particles are small, fast moving droplets while splash particles represent the remainder. Foam particles represent foam that floats on the surface of the water. Spray, splash and foam particles are generated within our simulation framework from breaking waves, waterfalls and interaction of solids with the fluid. In addition, they can be generated explicitly by user defined sources such as faucets, pouring waters or rain drops. We do not model bubbles raising from below the water in this work. Such effects could be added if desired with the method presented in [TSS*07].

2.4.1. Spray, Splash and Foam Simulation

We model spray and splash using particle systems without particle-particle interaction. The spray and splash particles need to be generated separately and simulated using different drag coefficients. When a particle is generated from a grid cell, it takes away some mass and momentum from the height field. When it falls back, it deposits its mass and momentum back at another location. If a particle is of radius r, the height that will be added/subtract is $\frac{V_{\text{eff}}}{(\Delta x)^2}$, where $V_{\text{eff}} = C_{\text{deposit}} \frac{4}{3} \pi r^3$ is the amplified volume of the particles. $C_{\text{deposit}} \geq 1$ makes the volume of the particles more than it physically is. C_{deposit} can be used to control the number of particles active in the simulation. We use the values of C_{deposit} between 1 and 10 in our examples. Let $u_{i+\frac{1}{2},j}$ be the nearest velocity along x-axis sample that a particle falls into, $\mathbf{v}^{\text{par}} = [v_x^{\text{par}}, v_y^{\text{par}}, v_z^{\text{par}}]^T$ be its velocity. We can update $u_{i+\frac{1}{2},j}$ by using

$$u_{i+\frac{1}{2},j} = \frac{u_{i+\frac{1}{2},j} h_{i,j} (\Delta x)^2 + v_x^{\text{par}} V_{\text{eff}}}{h_{i,j} (\Delta x)^2 + V_{\text{eff}}}. \tag{18}$$

A similar equation is used for $w_{i,j+\frac{1}{2}}$. When a splash particle hits the surface we create a foam particle with some probability, depending on the impact speed of the droplet.

We jitter the initial positions of particles by moving them by a random fraction of a time step. This makes them look as if they were generated somewhere in the middle of the time interval. We also add some noise to the initial velocities of the particles.

Foam is advected by the velocity field of the fluid simulation and projected onto the fluid surface. Its lifetime is a user-defined parameter modulated with some noise. It nicely conveys the horizontal swirling water motion in Figure 8.

2.4.2. Breaking Waves

Breaking waves cannot be resolved by a height field model. In a 2D framework, waves that would break in three dimensions produce disturbing ripples due to numerical instabilities. To solve this problem, we damp such waves and create particles to make sure we do not lose the effect as shown in Figure 5a. A cell (i, j) is considered to contain a breaking wave if it satisfies all of these conditions:

1. $|\nabla \eta_{i,j}| > \alpha_{\text{minSplash}} \frac{g\Delta t}{\Delta x}$ // Steep enough to break
2. $\frac{h_{i,j} - h_{i,j}^{\text{prev}}}{\Delta t} > v_{\text{minSplash}}$ // Raising fast and is front of a wave
3. $\nabla^2 \eta_{i,j} < l_{\text{minSplash}}$, // Is top of a wave

where $h_{i,j}^{\text{prev}}$ is the height in the previous time step. Condition 1 makes sure that the wave is steep enough. It is similar to a condition presented by Thuerey et al. [TMFSG07]. Condition 2 requires that the cell is part of the front of a fast raising wave. In our examples, we found it to be more robust than the condition $\nabla \eta_{i,j} \cdot [u_{i,j}, v_{i,j}]^T < 0$ of [TMFSG07]. Condition 3 ensures that we generate particles only near the top of a wave if its wave length is many grid cells wide. We use $v_{\text{minSplash}} = 4$, $\alpha_{\text{minSplash}} = 0.45$, $l_{\text{minSplash}} = -4$ in our examples. We compute $\nabla \eta_{i,j}$ using the maximum among the one-sided derivatives, to allow detection of a fast height change, as

$$\nabla \eta_{i,j} = \begin{bmatrix} \frac{\max_{\text{abs}}(\eta_{i+1,j} - \eta_{i,j}, \eta_{i,j} - \eta_{i-1,j})}{\Delta x} \\ \frac{\max_{\text{abs}}(\eta_{i,j+1} - \eta_{i,j}, \eta_{i,j} - \eta_{i,j-1})}{\Delta x} \end{bmatrix}, \tag{19}$$

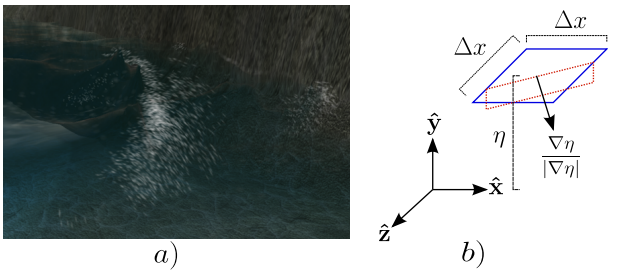


Figure 5: *a) Breaking wave particles. b) Particles are placed within the red dotted rectangle.*

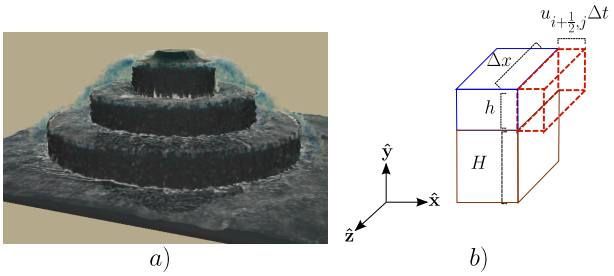


Figure 6: *a) A scene with multiple layers of waterfalls. b) The waterfall particles are seeded within the dotted red box.*

where $\max_{\text{abs}}(s,t) = (|s| > |t|)?s : t$. We compute $\nabla^2\eta_{i,j}$ using a central finite differencing as

$$\nabla^2\eta_{i,j} = \frac{\eta_{i+1,j}+\eta_{i-1,j}+\eta_{i,j+1}+\eta_{i,j-1}-4\eta_{i,j}}{(\Delta x)^2}. \quad (20)$$

We then generate particles within a rectangle whose edges are the y-axis and the line perpendicular to the direction of the gradient passing through the cell center clipped against cell edges, as shown in Figure 5b. The total volume of the added particles is proportional to $|\nabla\eta_{i,j}|$. We jitter these particle positions by uniform noise sampled form $[-\frac{\Delta x}{2}, \frac{\Delta x}{2}]$. The x and z components of the particle velocities are given by the wave velocity [TMFSG07] and are clamped to the maximum water depth, $\frac{-\nabla\eta_{i,j}\sqrt{g\min(h_{i,j},h_{\text{avgmax}})}}{|\nabla\eta_{i,j}|}$. The y components is $\lambda_y\frac{h_{i,j}-h^{\text{prev}}_{i,j}}{\Delta t}$, where we use $\lambda_y = 0.1$ in all examples. We also label a fraction of these particles as spray.

2.4.3. Waterfalls

We treat a face $(i+\frac{1}{2}, j)$ as a waterfall face if

- case 1: $\frac{H_{i,j}-H_{i+1,j}}{\Delta x} > \Delta H_{\text{cap}}$ and $H_{i,j} > \eta_{i+1,j}$ or
- case 2: $\frac{H_{i+1,j}-H_{i,j}}{\Delta x} > \Delta H_{\text{cap}}$ and $H_{i+1,j} > \eta_{i,j}$.

In each case, the left condition states that the terrain slope is larger than ΔH_{cap}, where we use $\frac{3}{2}$ in all examples. This indicates an approximate discontinuity in the terrain height. The right condition makes sure that the lower end of the waterfall is not already filled to the same level as the higher end. An example with multiple layers of waterfalls is shown in Figure 6a.

We generate splash particles by sampling uniformly over an axis aligned bounding box defined by $\mathbf{p}^{\text{wf}}_{\text{min}}$ and $\mathbf{p}^{\text{wf}}_{\text{max}}$ as shown in Figure 6b. Their velocities are set to $\mathbf{v}^{\text{wf}}$. This bounding box represents the volume of the fluid that would flow across the face $(i+\frac{1}{2}, j)$ during the current time step. These variables are computed using the following algorithm:

if $\eta_{i+1,j} < H_{i,j}$ **then**
  $\mathbf{v}^{\text{wf}} = [u_{i+\frac{1}{2},j}, 0, \frac{w_{i,j+\frac{1}{2}}+w_{i,j-\frac{1}{2}}}{2}]^T$
  $\mathbf{p}^{\text{wf}}_{\text{min}} = [i\Delta x, H_{i,j}, j\Delta x]^T$
  $\mathbf{p}^{\text{wf}}_{\text{max}} = [i\Delta x + u_{i+\frac{1}{2},j}\Delta t, \eta_{i,j}, (j+1)\Delta x]^T$
else
  $\mathbf{v}^{\text{wf}} = [u_{i+\frac{1}{2},j}, 0, \frac{w_{i+1,j+\frac{1}{2}}+w_{i+1,j-\frac{1}{2}}}{2}]^T$
  $\mathbf{p}^{\text{wf}}_{\text{min}} = [(i+1)\Delta x + u_{i+\frac{1}{2},j}\Delta t, H_{i+1,j}, j\Delta x]^T$
  $\mathbf{p}^{\text{wf}}_{\text{max}} = [(i+1)\Delta x, \eta_{i+1,j}, (j+1)\Delta x]^T$
end if

We do not subtract the volume of the splash particles from the height field because the modified height integration step explained next already take this into account. The height and velocity integration is modified as follows:

- case 1: Replace $u_{i+\frac{1}{2},j}$ with 0 when updating $h_{i+1,j}$.
  Replace Eq. 8 with $u_{i+\frac{1}{2},j} -= g\Delta t(-\frac{h_{i,j}}{\Delta x} - \Delta H_{\text{cap}})$.
- case 2: Replace $u_{i+\frac{1}{2},j}$ with 0 when updating $h_{i,j}$.
  Replace Eq. 8 with $u_{i+\frac{1}{2},j} -= g\Delta t(\frac{h_{i+1,j}}{\Delta x} + \Delta H_{\text{cap}})$.

These modifications have the effect that the cell at the bottom of the waterfall treats the waterfall face as a reflecting boundary and that the waterfall face's velocity is updated specially. The volume that would flows across this boundary is carried by the particles instead. Our waterfall particle generation algorithm not only enhances the visual richness of the scene, but also makes the simulation more stable because steep terrain slopes cause numerical instabilities. The advection of quantities along the velocity field would have to be modified as well to make sure that they are not traced up the waterfall. However, we found that this can safely be ignored in practice without introducing visual artifacts. Waterfall faces $(i, j+\frac{1}{2})$ are treated in a similar manner.

2.4.4. Splashes from Interaction with Bodies

We also generate splashes and spray when rigid and soft bodies interact with the fluid. For this we go through each surface triangle of the solid and check if it sweeps through

the height field during the current time step and its speed is above a threshold, v_{thres}. If so, we sample the triangle uniformly by recursive subdivision as in Section 2.3. We stop when the area is roughly the same as the size of splash particles. For each sample, we check whether it sweeps through the height field. If so, we generate splash particles. Let $\mathbf{p}^s$, $\mathbf{v}^s$, $\mathbf{n}^s$ be the sample's position, velocity and normal found by barycentric interpolation from the original triangle and let $\mathbf{v}^s_{\text{fluid}}$ be the fluid velocity at $\mathbf{p}^s$. The velocity of the particles generated, $\mathbf{v}_{\text{par}}$, is computed using the following algorithm:

$$\mathbf{v}_{\text{rel}} = \mathbf{v}^s - \mathbf{v}^s_{\text{fluid}},$$
$$\mathbf{v}_{\text{rn}} = (\mathbf{v}_{\text{rel}} \cdot \mathbf{n}^s)\mathbf{n}^s,$$
$$\mathbf{v}_{\text{rp}} = \mathbf{v}_{\text{rel}} - \mathbf{v}_{\text{rn}},$$
$$\mathbf{v}_{\text{par}} = \mathbf{v}^s_{\text{fluid}} + \alpha_{\text{rn}}\mathbf{v}_{\text{rn}} + \alpha_{\text{p}}\mathbf{v}_{\text{rp}} + \alpha_{\text{n}}\mathbf{n}^s|\mathbf{v}_{\text{rel}} \cdot \hat{\mathbf{y}}| + \alpha_{\text{r}}(\mathbf{v}_{\text{rel}} - 2(\hat{\mathbf{y}} \cdot \mathbf{v}_{\text{rel}})\hat{\mathbf{y}}).$$

The α's parameters control the look of the splashes. In our examples, we use $\alpha_{\text{rn}} = 0.86, 0.10 \leq \alpha_{\text{p}} \leq 0.23, 0.00 \leq \alpha_{\text{n}} \leq 0.01$, and $0.17 \leq \alpha_{\text{r}} \leq 0.61$. We then generate particles at $\mathbf{p}^s + 2rq\mathbf{n}^s$, where $0 \leq q < num\_par$, with initial velocity $\mathbf{v}_{\text{par}}$. We make num_par proportional to $|\mathbf{v}^s| - v_{\text{thres}}$. The position of these particles are jittered randomly within the radius r.

2.5. Rendering

After a time step, we have a newly updated state of the height field fluid and the particles. In this section we describe how to render them. Our goal is to convey the complex fluid flow features and to enhance the surface details of the fluid.

2.5.1. Wave Crest Alignment and Choppy Waves

The height field fluid can be thought of as a rectangular grid of quads over the x-z plane. $\eta_{i,j}$ determines the y-coordinate of the vertex (i,j) of the grid. Each quad can be split into two triangles in two ways. Silhouette artifacts are reduced by picking the diagonal that better aligns with wave crests. We pick diagonal $(i,j)-(i+1,j+1)$ if $\eta_{i,j}+\eta_{i+1,j+1} > \eta_{i+1,j}+\eta_{i,j+1}$, otherwise, we pick diagonal $(i+1,j)-(i,j+1)$. This test can be implemented efficiently on the GPU using a geometric shader. We employ this technique in the beach scene (Figure 3) and the open ocean scene (Figure 2). To improve the appearance of the water surface further, we make the waves look choppy as in [LO07], by replacing vertex coordinates (x,z) with $(x + c\frac{\partial \eta}{\partial x}, z + c\frac{\partial \eta}{\partial z})$, where $c > 0$ control the choppiness. We also clamp the displacement in each direction to not exceed $\frac{\Delta x}{2}$ in order to avoid self intersections.

2.5.2. FFT Waves Advection

The height field simulation described so far cannot resolve waves with wave lengths smaller than the grid resolution Δx. With the constraint of being real-time, decreasing Δx further is not an option because this would increase the number of cells and require a smaller time step Δt for a stable simulation. Instead, we want to add smaller waves on top of the height field with the following properties:

1. They should be advected with the velocity field.
2. They must not be distorted excessively over time.
3. They disappear if being stretched too much.
4. The method must be relatively cheap.

The key idea is to use an FFT based wave simulation [MWM87, Tes99] with periodic boundary conditions to generate a high resolution wave texture $F(s,t)$ and lookup this texture for the additional sub-grid displacements of the height field. The FFT wave texture only contains high frequency waves. The texture coordinates and the weights used for modulating the additional displacement are computed on the simulation grid points. They are then bi-linearly interpolated for per-pixel bump mapping or for displacing the rendering grid. We use the texture for bump mapping in all examples. We also use it for displacing a high resolution rendering grid with a grid spacing of $\frac{1}{4}\Delta x$ in the boat riding example. Figure 7 shows the comparisons between rendering without FFT waves, with constant weight FFT waves and with varying weight FFT waves. Notice how the constant weight case contains unnaturally distorted small waves.

We employ the method present by Max and Becker [MB96] for the texture coordinates re-generation and texture blending. For each grid cell (i,j) we additionally store 3 sets of texture coordinates $(s^k_{i,j}, t^k_{i,j}), k = 1..3$ which are initialized with $(i\Delta x, j\Delta x)$. They are transported with the velocity field using semi-Lagrangian advection. We reset $(s^k_{i,j}, t^k_{i,j})$ to $(i\Delta x, j\Delta x)$ every τ seconds with a phase shift of $\tau/3$ between k and $(k+1) \mod 3$. The weight used for multiplying the displacement $\omega_{i,j}$ should depend on how much the velocity field would *stretch* the wave. We use $\omega_{i,j} = e^{-\Omega\mu_{i,j}}$, where $\mu_{i,j} = \max_{k=1}^{3} \mu^k_{i,j}$. $\mu^k_{i,j}$ is the absolute value of the eigenvalue with maximum magnitude of the Green strain of (s^k, t^k) at (i,j). The Green strain can be computed as $\frac{1}{2}(\mathbf{D}\mathbf{D}^T - I)$, where $\mathbf{D} = \begin{bmatrix} \frac{\partial s^k}{\partial x} & \frac{\partial s^k}{\partial z} \\ \frac{\partial t^k}{\partial x} & \frac{\partial t^k}{\partial z} \end{bmatrix}$. $\Omega > 0$ controls how fast the small waves disappear with stretching, for which we use values between 0.5 to 2 in our examples. The final displacement for a given position is $\omega \sum_{k=1}^{3} w_k F(s^k, t^k)$, where ω, s^k, t^k are bi-linearly interpolated from the grid, $w_k = \frac{1}{3}(1 - cos(2\pi\frac{t - t^k_0}{\tau}))$, t^k_0 is the last time the set k is regenerated. The phase shift ensures a constant blending weight. The choice of τ trades off the distortion of the texture coordinates against the coherency of the waves. We use $\tau \leq 3$ seconds in our examples. One could also adapt a texture coordinate regeneration method presented by Neyret [Ney03] to not have to make this tradeoff, but it is more expensive.

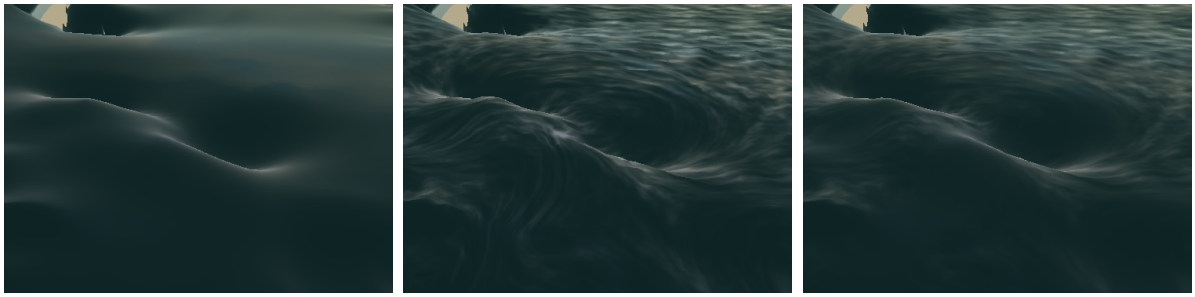


Figure 7: *Left: No FFT Mid: FFT constant weight ($\Omega = 0$) Right: FFT varying weight ($\Omega = 1$)*

2.5.3. Particles Rendering

We render the splash particles using the screen space technique presented by Van der Laan et al. [VdLGS09] with the modification of using a bi-lateral filter with support radius defined in world space [RD10], so that the fluid appearance is view-independent. For the final rendering of the particle surface, we use the same pixel shader as the one for rendering the height field.

Spray particles are rendered as an elongated ellipse along the direction of their velocity to emulate the motion blur effect as in [VdLGS09]. Foam particles are rendered as diffuse disks with normals perpendicular to the height field water surface.

3. Results and Discussion

We implement our method both on the CPU as well as on the GPU using CUDA. All timings are done on an Intel Core i7, 2.67 GHz with 4GB of memory and NVIDIA GTX480. We use NVIDIA's PhysX SDK for rigid body, soft body, and cloth simulation. The size of the grid and the number of particles used for each example are stated in Table 1. We use $\Delta t = 16.66ms$ in all examples, except the boat riding example, where we use $\Delta t = 20ms$. Table 2 show the timings for various examples. For scenes indicated with the letter G, all the steps are executed on the GPU except for the solid coupling. Porting the solid coupling part to the GPU is part of future work. For the CPU cases, the height field simulation and the particles simulation are multi threaded.

A drawback of our method is that although the height integration step conserves volume, the coupling and the overshooting reduction do not. However, we found that volume loss is less crucial in these situations and a global mass adjustment technique such as the one proposed by [Hes07] could be employed if desired. Another drawback is that our stability enhancements cannot guarantee unconditional stability. However, they allow us to explicitly integrate with a reasonable time step in all of our examples.

When a dense object falls into the water and fully submerges, there is usually a secondary splash called Rayleigh jet due to the collapse of water over the cavity formed by the object. We are currently experimenting with a method to simulate this phenomena. Another area of future work is in rendering of the water represented by the height field and the particles seamlessly and efficiently.

	Boat	WaterF	PML	Beach	Ocean
Grid	900x135	128x128	128x128	128x128	256x256
Par	250K	56K	2K	220K	83K

Table 1: *Grid size and average number of foam, spray, and splash particles active in a given frame.*

	Total	HF	Gen	Par	Cou
BoatC1	94.50	42.25	5.11	16.18	2.46
BoatC2	76.75	25.69	5.11	9.91	2.31
BoatC3	73.88	23.25	5.11	8.18	2.24
BoatC4	69.75	19.06	5.09	7.55	2.31
BoatG	18.05	2.19	1.36	1.15	2.32
WaterFG	7.97	0.75	0.82	0.44	0.00
PMLG	3.78	0.78	0.13	0.19	0.00
BeachG	9.88	0.75	1.87	0.90	0.00
OceanG	13.13	1.28	1.16	0.56	3.13

Table 2: *Timing for various examples in milliseconds. Total is the frame time including rendering. HF includes advection, height integration, velocity integration, PML, overshooting reduction and FFT texture coordinate advection. Gen includes all particle generation steps. Par includes particles simulation. Cou is the coupling of solids with fluid. The lines BoatC1...C4 contain the timings of the boat demo running on the CPU with 1-4 threads. The other lines show timings on the GPU. PML shows the all absorbing boundaries case. Beach shows the timings for the case with overshooting reduction.*

References

[ATBG08] ANGST R., THÜREY N., BOTSCH M., GROSS M.: Robust and Efficient Wave Simulations on Deforming Meshes. *Computer Graphics Forum 27 (7)* (October 2008), 6, 1895 – 1900. 2, 3

[BAB09] BURRELL T., ARNOLD D., BROOKS S.: Advected river textures. *Comput. Animat. Virtual Worlds 20*, 2–3 (2009), 163–173. 2

[Bri05] BRIDSON R.: Shallow water discretization, Lecture notes Animation Physics. University of British Columbia, 2005. 3

[Bri08] BRIDSON R.: *Fluid Simulation for Computer Graphics*. A K Peters, 2008. 2

[CL95] CHEN J. X., LOBO N. D. V.: Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graph. Models Image Process. 57*, 2 (1995), 107–116. 2

[CS09] CORDS H., STAADT O.: Real-time open water environments with interacting objects. In *Proceedings of Eurographics Workshop on Natural Phenomena (EGWNP)* (2009). 2

[FR86] FOURNIER A., REEVES W. T.: A simple model of ocean waves. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 75–84. 2

[Hes07] HESS P.: *Extended Boundary Conditions for Shallow Water Simulations*. Master's thesis, ETH Zurich, 2007. 2, 3, 9

[HHL*05] HAGEN T. R., HJELMERVIK J. M., LIE K.-A., NATVIG J. R., HENRIKSEN M. O.: Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory 13*, 8 (2005), 716–726. 2

[HNC02] HINSINGER D., NEYRET F., CANI M.-P.: Interactive animation of ocean waves. In *Proc. SCA* (2002), pp. 161–166. 2

[HW04] HOLMBERG N., WÜNSCHE B. C.: Efficient modeling and rendering of turbulent water over natural terrain. In *Proc. GRAPHITE* (2004), pp. 15–22. 2

[Joh08] JOHNSON S. G.: Notes on perfectly matched layers. online mit course notes, July 2008. 4

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *Proc. SIGGRAPH* (1990), pp. 49–57. 2

[LO07] LEE R., O'SULLIVAN C.: A fast and compact solver for the shallow water equations. *Virtual Reality Interations and Physical Simulation 1* (2007), 51–58. 2, 8

[LvdP02] LAYTON A. T., VAN DE PANNE M.: A numerically efficient and stable algorithm for animating water waves. *The Visual Computer 18*, 1 (2002), 41–53. 2, 3

[MB96] MAX N., BECKER B.: Flow visualization using moving textures. In *Proc. LCAS/LaRC Symposium on Visualizing Time-Varying Data* (1996), pp. 77–87. 8

[MFC06] MAES M. M., FUJIMOTO T., CHIBA N.: Efficient animation of water flow on irregular terrains. In *Proc. GRAPHITE* (2006), pp. 107–115. 2

[MWM87] MASTIN G. A., WATTERBERG P. A., MAREDA J. F.: Fourier synthesis of ocean scenes. *IEEE Comput. Graph. Appl. 7*, 3 (1987), 16–23. 2, 8

[MY97] MOULD D., YANG Y.-H.: Modeling water for computer graphics. *Computers & Graphics 21*, 6 (1997), 801–814. 2

[Ney03] NEYRET F.: Advected textures. In *Pro. SCA* (2003), pp. 147–153. 8

[OH95] O'BRIEN J. F., HODGINS J. K.: Dynamic simulation of splashing fluids. In *Proc. Computer Animation* (1995), p. 198. 2

[Pea86] PEACHEY D. R.: Modeling waves and surf. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 65–74. 2

[RD10] ROUSLAN DIMITROV M. S.: SPH fluid rendering, in submission to siggraph talk, 2010. 9

[SFK*08] SELLE A., FEDKIW R., KIM B., LIU Y., ROSSIGNAC J.: An unconditionally stable MacCormack method. *J. Sci. Comput. 35*, 2-3 (2008), 350–371. 3

[SM09] SÖDERSTRÖM A., MUSETH K.: Non-reflective boundary conditions for incompressible free surface fluids. *SIGGRAPH Talks* (2009). 4

[SP09] SOLENTHALER B., PAJAROLA R.: Predictive-corrective incompressible sph. In *Proc. SIGGRAPH* (2009), pp. 1–6. 2

[TDG00] THON S., DISCHLER J.-M., GHAZANFARPOUR D.: Ocean waves synthesis using a spectrum-based turbulence function. In *Proc. CGI* (2000), p. 65. 2

[Tes99] TESSENDORF J.: Simulating ocean water. SIGGRAPH course notes, 1999. 2, 8

[TMFSG07] THUREY N., MULLER-FISCHER M., SCHIRM S., GROSS M.: Real-time breakingwaves for shallow water simulations. In *Proc. Pacific Conf. on CG and App.* (2007), pp. 39–46. 2, 6, 7

[TSS*07] THÜREY N., SADLO F., SCHIRM S., MÜLLER-FISCHER M., GROSS M.: Real-time simulations of bubbles and foam within a shallow water framework. In *Proc. SCA* (2007), pp. 191–198. 6

[vBBK08] ŠŤAVA O., BENEŠ B., BRISBIN M., KŘIVÁNEK J.: Interactive terrain modeling using hydraulic erosion. In *Proc. SCA* (2008), pp. 201–210. 2

[VdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *Proc. I3D* (2009), pp. 91–98. 9

[WMT07] WANG H., MILLER G., TURK G.: Solving general shallow wave equations on surfaces. In *Proc. SCA* (2007), pp. 229–238. 2, 3

[YHK07] YUKSEL C., HOUSE D. H., KEYSER J.: Wave particles. In *Proc. SIGGRAPH* (2007), p. 99. 2, 5

[YNBH09] YU Q., NEYRET F., BRUNETON E., HOLZSCHUCH N.: Scalable real-time animation of rivers. *Proc. Eurographics 28*, 2 (2009). 2

Appendix

3.1. PMLs for z-moving waves

$$h_{i,j} \mathrel{+}= (-\gamma_{i,j}(h_{i,j} - h_{\text{rest}}) + \psi_{i,j})\triangle t \tag{21}$$

$$w_{i,j+\frac{1}{2}} \mathrel{+}= -\frac{1}{2}(\gamma_{i,j+1} + \gamma_{i,j}) w_{i,j+\frac{1}{2}} \triangle t \tag{22}$$

$$\psi_{i,j} \mathrel{+}= -\lambda_{\text{update}} \psi_{i,j} \frac{(u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j})}{\triangle x} \triangle t \tag{23}$$

$$\psi_{i,j} \mathrel{\times}= \lambda_{\text{decay}} \tag{24}$$

We evolve the field ψ only in the regions where we want to damp a z-moving waves. γ plays the same role as σ in Section 2.1.4.