KTH Computer Science and Communication

Real-time Interactive Water Waves

Water Wave Simulation for Computer Games

BJÖRN OTTOSSON

Master's Thesis at CSC
Supervisor: Lars Kjelldahl
Examiner: Lars Kjelldahl

Abstract

In this master thesis report a new method for simulating waters surface waves is presented. The method is well adapted for real-time applications and has been developed with computer games in mind. By simulating the water surface at several different resolutions simultaneously using a construction similar to Laplacian Pyramids dispersion is handled approximately resulting in a complex behaviour. The simulation is also extended with a dynamic level of detail method and phenomenological models for boundaries and high frequency waves. This method is prototyped inside the Frostbite™ engine developed at EA™ DICE™ and runs at 3 ms per time step on a single core of a Intel™ Xeon™ processor with high quality results.

Referat

Interaktiva vattenvågor i realtid

Denna rapport presenterar resultaten för ett examensarbete om simulering av vattenvågor. En ny metod för att simulera vattenvågor presenteras. Denna metod är anpassad för realtids-tillämpningar och har utvecklats med datorspel i åtanke. Genom att simulera en vattenyta med flera upplösningar samtidigt i en pyramidstruktur hanteras dispersion approximativt. Simuleringen varierar dynamiskt upplösning beroende på avstånd till observatören och för att öka detaljnivån finns ett fenomenologiskt detaljsystem. En prototyp har utvecklats inuti spelmotorn Frostbite™som utvecklats av EA™ DICE™. Prototypen simulerar ett steg på 3 ms men en kärna hos en Intel™ Xeno™-processor med bra resultat.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Simulating water realistically is of great interest in computer graphics since it is such an important part of our world. It is also a very complex problem. Today realistic computer generated water is seen in many movies. This not the case for computer games. The simulations used in movies are not computed in real-time and a single frame may take several minutes or hours to compute (24 frames are needed per second) and may use a cluster of computers. In video games this is not possible, instead the simulation has to be computed in real-time on a single computer or video game console. Because of this only a 1/24th of a second is available per frame to achieve 24 frames per second. Also only a fraction of a the computer's or the console's processing power is available since to rest of the game has to be run at the same time. This is a difference of several orders of magnitude. Because of this water is rarely simulated in games today and when it is in a very simple fashion.

In this thesis we will focus on surface waves on water that as a whole is in rest. Examples of this are ponds, lakes and pools. The goal is to investigate how accurately this can be done in a real time context and find a method that is usable in games on this generation of consoles. The method should be easy to control from a user perspective: an artist should be able to tweak as much as possible for artistic use in a computer game. It should also be predictable in performance. In games consistent performance is more important than simulation accuracy: keeping the frame-rate is more important than keeping the simulation physical. Another important area of focus is level of detail. In order to efficiently use the computing power resolution should be adapted to the distance to the observer. Waves far away need less resolution than waves close since they will occupy less screen space. Parallelization is also important to utilize the parallel nature of modern computer processors and video game consoles.

An important area of research is investigating what simplification are useful and provide realistic results. In this thesis we limit our investigation to linear wave theory since it accounts for important properties of water such as wave speed dependency on wavelength and water height. It is also an interesting starting point since several com-

mon approximation may be derived from it. Another advantage linear wave theory provides is an intuitive understanding of wave behavior that is hard to obtain with other approaches.

Using linear wave theory we present a method for simulating water surface waves that approximates the height and wavelength dependency of water waves. In this way it captures the complexity of water waves in nature. Our method is well suited for level of detail techniques and is highly parallizable.

Chapter 2

Background

One of the hardest problems in computer graphics is fluid-simulation. Large scale (millions of particles) offline fluid simulations have not been possible until recently with software such as RealFlow [rea]. Large scale real-time simulations are still not feasible. To simulate fluid dynamics Navier-Stokes equations have to be solved. Usually simulation is restricted to incompressible Newtonian fluids. This yields the incompressible Navier-Stokes equations (2.1) and the volume conservation equation(2.2):

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = -\nabla p + \mu\nabla^2\mathbf{v} + \mathbf{f} \tag{2.1}$$

$$\nabla\cdot\mathbf{v} = 0 \tag{2.2}$$

Where $\mathbf{v}$ is the flow velocity, ρ is the fluid density, p is the pressure, μ is the dynamic viscosity and $\mathbf{f}$ represents body forces such as gravity. These equations are usually solved with smoothed-particle hydrodynamics (SPH) for fluids and 3D grid-based methods for gases. For real time solvers only modest particle numbers (e.g.. 65 000 [Gre08]) and grid sizes (e.g.. 128x128x32 grid [KID10]) can be solved. These results are from simulations utilizing the full processing power of a high-end PC. In real-time applications such as games only a fraction of the computing power is available: simulation size has to be even less. In order to simulate more detailed fluids in real-time vast simplifications have to be made.

2.1 The Height Field Model

One way of simplifying the simulation is to only consider the surface of the fluid and to model it as a height field. This means that the surface height (h) is modeled as a function of horizontal spatial coordinates (x, y). This effectively reduces the problem size by a dimension. A height field correctly represents waves on for example an ocean surface, a pond, a river etc as long as the waves are fairly small. Splashes and breaking waves can not be represented however since they violate the height field assumption. Nevertheless this is a useful representation and height field approach will be the focus of this paper.

There has been a lot of research on this description of water surfaces. We will focus on the results of linear wave

theory and the shallow water equation. Studying other higher order models such as the Boussinesq model for water waves are possible future areas of research.

2.1.1 Linear Wave Theory

Linear wave theory describe waves as small perturbation of the water surface and ignores effects of water current (advection). Also viscosity is assumed to be zero. Because of this linear wave theory does not correctly handle rivers, steep waves etc. There are also several other assumption that are out of the scope for this brief introduction. These assumption results in the following expression for the water speed, c:

$$c = \sqrt{\frac{g}{|\mathbf{k}|} \tanh(kH)} \tag{2.3}$$

Where g is the acceleration by gravity, k the angular wavenumber and H the water depth. For shallow and deep water respectively, c is reduced to:

$$c = \sqrt{gh}, \lambda \gg H \tag{2.4}$$

$$c = \sqrt{\frac{g}{|\mathbf{k}|}}, \lambda \ll H \tag{2.5}$$

Equation (2.5) is known as the dispersion relation for ocean waves and is responsible for the characteristic look of ship wakes (Kelvin wakes) and is therefore critical to simulation of realistic ocean waves.

2.1.2 The Shallow Water Equations

The shallow water equations are the results of a different set of approximations. Just as in linear wave theory viscosity is assumed to be zero. The most important assumptions are that the water height is assumed to be significantly less than the wavelength of water waves and that the flow is assumed to be constant vertically. Because of this the shallow water equations only simulate large water waves correctly - small waves will have a small wavelength compared to the water height and therefore contradict the wavelength assumption. Contrary to linear wave theory current is simulated; rivers etc can be simulated. Also the sharpening of waves as they reach shallower water is handled. The resulting equations are:

$$\frac{d\mathbf{v}}{dt} + g\nabla h + (\mathbf{v} \cdot \nabla)\mathbf{v} = 0 \tag{2.6}$$

$$\frac{dh}{dt} + (h + b)\nabla \cdot \mathbf{v} = 0 \tag{2.7}$$

Where $\mathbf{v}$ is the horizontal flow, g is the acceleration by gravity and h the water height from a reference level h_0 and b the water depth from h_0. Worth noting is that if we remove the advection term,$(\mathbf{v} \cdot \nabla)\mathbf{v}$, from (2.6) assume that $h \ll b$ the shallow water equations reduce to the linear wave theory model for shallow water, that is the wave speed is given by (2.4).

2.1.3 Turbulence

Neither linear wave theory nor shallow water equations model turbulence. However turbulence plays an important role in realistic simulation of water surfaces. Many situations common in games result in a great deal of turbulence. A person running through water is such a situation. Turbulence is created when there is a large difference in flow velocity

within a short distance. Understanding of turbulence is limited and it is therefore hard to do accurate simulations without solving the full Navier-Stokes equations. There are however several phenomenological models. See [FS06, MK99]. One of the simplest methods is to model turbulence by diffusion. In turbulent areas the flow field will be close to random which results in diffusion, much like Brownian motion. Physical models for turbulence are beyond the scope of this paper. In section 2.2 a brief overview of turbulence models for computer graphics is given.

2.2 Related Work

This literature review is limited to work relevant to real-time water surface simulations with rigid body interaction. For a complete overview of fluid simulation in computer graphics see [Sch07, Igl04, BMF07]. Section 2.2.1 contains an overview of 2d height field techniques, section 2.2.2 an overview of research on controlling simulations and phenomenological methods for adding details to simulations such as turbulence, foam and particles and section 2.2.3 an overview of different methods for interaction.

2.2.1 Height Field Methods

An early approach for simulating water surfaces uses a Fourier approach and propagates waves in the frequency domain. Waves are stored as amplitude and phaseangle for each wavenumber. Propagation is done by updating the phaseangle according to the dispersion relation for ocean waves. While accurate modeling the dispersion relation it is difficult to combine this method with interaction. Because of this it is often used for ambient waves. For an overview see [Tes04b]. Recently this algorithm has been implemented on GPU [Fin04].

One other approach for simulating water height fields is to use grids and finite differences to solve simplified 2D shallow water equations where the flow is assumed to be zero. This approach was used by [KM90]. [CL95] use a similar method and solve compressible 2D-Navier Stokes equation and extend the finite difference approach to handle object interaction by introducing variable boundary conditions. The height of the water is generated using the pressure.

Recent work has focused on solving the full shallow water equations including current using a Semi-Lagrangian approach[LvdP02, HHL$^+$05]. [Kal08] simulate shallow water equations with an adaptive grid to do fluid simulations with a scale of several hundred meters for use in open world games. Semi-Lagrangian approaches suffer from dissipation due to the discretization. Research has been done to reduce this [KLLR05].

Another recent development is the incorporation of dispersion in interactive simulations. [Tes04a] and [Lov02, Lov03] do this by convolution.[Day09] use a multi-resolution frequency domain approach. Their simulation is done using an adaptive grid. The waves are propagated in the Fourier domain using the deep water dispersion relation. The resolution is changed in run-time to so that the simulation has high resolution near the player and low resolution far away.

Particle systems have also been used for water simulation. Recently 2D particle systems have been used to simulate water surfaces. [LH10] use SPH in 2D to animate a height field. [YHK07] intro-

duce the wave particles, particles representing a small disturbance of the water surface, and use them to solve a simple wave equation. The particles are splatted to a height field for rendering. A big advantage of this approach is that the simulation is concentrated to the areas with perturbations and barely any computation is needed in areas without waves. However, with lots disturbances more and more wave particles have to be created which results in bad worst case performance. Wave particles are in this wave very similar to ordinary particle systems.

[Cor08] extend the wave particle approach to handle flow by coupling it with a low resolution flow simulation. [Cor07] couple a low resolution SPH simulation with a surface wave simulation.

2.2.2 Control and Detail

Increasing particle numbers and grid resolution result in major performance degradation. Therefore a lot of work has been made regarding up-sampling and adding details to lower resolution simulations. Up-sampling a fluid without turbulence is fairly easy to do. To make a low resolution simulation more detailed high frequency turbulence has to be added. This can be done either by adding noise [BHN07, KTJG08] or by simulating the generation of turbulence at a higher resolution [PTSG09]. Height field simulation lack foam and splashes. Work has been done on how to add this to existing simulations. [JG01] add foam from particles and steep waves and render it by blending in a foam texture. They also generate particles from the height field simulation to simulate splashes. [Vla10] use flow to animate a texture with good result.

Fluid simulations are generally hard to control and predict due to many parameters (such as grid subdivision, static geometry, interacting objects etc) with often large unexpected consequences. Therefore work has been made at controlling fluid animations while keeping the overall appearance dynamic. For SPH simulations the notion of guide particles has been introduced [TKPR06]. With this approach individual particles are allowed to move freely but the local mean of particle motion is restricted. [NCZ+09, Nie10] propose a method for guiding grid-based fluid simulations. Neither of these approaches are directly applicable to height field simulations.

2.2.3 Interaction

There are mainly two different ways of interacting with a height field. Either the height field and flow are modified directly or the boundary conditions are modified. The first approach is used in [CL95, Tes04a] to simulate the motion of rigid objects. In the approach used by [Tes04a] the height of the water is increased in front of a moving object and decreased behind it. When an object is submerged the height is increased above it corresponding to the volume of the object and when an object emerges the height is decreased. This approach can also be used to add and remove water. The other approach is to use handle interaction by applying pressure. When an object is floating in the water pressure is applied to the height field resulting in a lower height below the object.

[Tes04a] also use what they refer to as an obstruction mask. This mask con-

tains which parts of the water surface that is obstructed. This is used to create rigid boundaries against the obstructed areas. This is not physically correct but results in plausible reflections against dynamic objects.

Chapter 3

Overview

The aim of this chapter is to give a simple overview of the complete system. The purpose of this is to give the reader an intuitive understanding before diving into theoretical details. Therefore this chapter will be less formal than the rest of the report and will not contain many references. More theoretical details and motivation of design choices can be found in the following chapters: chapter 4 och chapter 5.

3.1 Variable Wave Speed

As we saw in chapter 2 an important property of water is that the wave speed is dependent on wave length. This is known as the dispersion relation of water waves. The result of this property is the characteristic look of water surface waves and the reason behind phenomenon such as Kelvin wakes. Linear wave theory provides a highly simplified description of surface waves. While being heavily simplified it still manages to capture the dispersion property. In the linear wave theory model wave speed increases until wave length and water height is roughly the same and then remains constant.

The water surface can be represented by a grid. Given this representation it is easy to simulate waves with constant speed. This can be done by Euler stepping the wave equation:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \nabla^2 h \tag{3.1}$$

It can be showed that the variable wave speed of linear wave theory can be simulated by replacing $c^2\nabla^2$ with a convolution, L:

$$\frac{\partial^2 h}{\partial t^2} = L * h \tag{3.2}$$

This is similar to the method of [Tes04a]. While correctly accounting for the dispersion relation this method is not optimal for use in a real time context. The size of the convolution operator L is dependent on the largest possible wave length. To represent a 1 m wave with a grid resolution of 5 cm the convolution operator needs roughly 20 grid cells wide to handle variable wave speed at that wave length. The convolution is not separable either so 400 cells have to be summed to calculate the contribution to one pixel. [Day09] alleviate this by carrying out the convolution in the Fourier domain at the cost of doing a forward and

inverse Fourier transform in each step. We present a different solution.

Our method is the following: h is split into different parts $(h_1, h_2, h_3, \ldots, h_k)$ representing different ranges of wave lengths. We will refer to these as sub grids. h_1 contains wave lengths down to some wave length λ, h_2 wave lengths between λ and $\frac{\lambda}{2}$, $h_3$wave lengths between $\frac{\lambda}{2}$ and $\frac{\lambda}{4}$, and so on. With this representation the shortest wavelengths of h_i are twice those of h_{i+1} ($i \in \mathbb{Z}, i > 0$). Because of this h_i can be simulated at half the resolution of h_{i+1}. The total height h can be retrieved by interpolating and summing h_i for all possible i.

The big advantage of this approach is that all the wave lengths contained in a grid are fairly small compared with the grid cell width. Because of this the convolution operator L can be approximated with a small convolution kernel. Within a sub grid the wave speed will not vary as much as within the whole range of frequencies and simple approximation of L can be used. By splitting the grid into several sub grids the total number of grid cells being simulated increase but the computation per cell is drastically reduced.

All the sub grid together will be referred to as a pyramid.

3.2 Level of Detail

To make the simulation fast without sacrificing detail it is important to control the resolution of the grid so that areas of high importance get higher resolution than unimportant areas. When viewed through a perspective projection the water close to the observer will require higher resolution since it occupies more of projected space. As the observer moves different areas will need higher resolution.

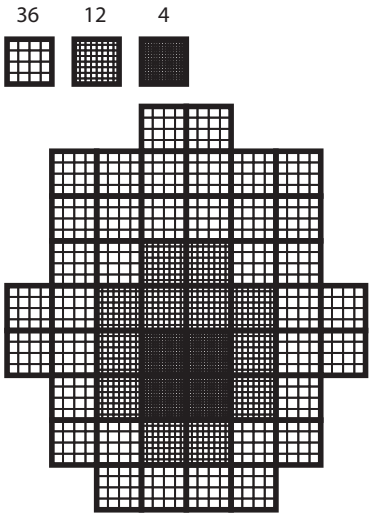

Figure 3.1. Level of Detail. Example distribution of different resolution pyramids.

Our system solves this by having pool of pyramids of different resolutions all with the same width (6 m for example). These are dynamically distributed to simulate water close to the observer width a high resolution and water further away with a higher resolution. See figure 3.1 for an example distribution.

3.3 Interaction

Interaction is handled by displacing the height field. When an object is pulled out of water the water height is decreased and when pushed down height is increased. See figure 3.2. Using our wavelength decomposition into sub grids makes this more complex. Only waves with correct wavelength should be added to a sub grid. This can be achieved by bandpass filtering the displacement with a filter corresponding to the wavelengths

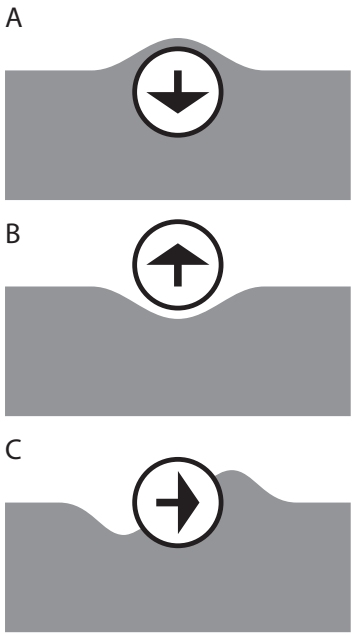

Figure 3.2. Example of Object Interaction. A, B and C show an object moving in different directions (arrow) and the corresponding change in height-field (gray).

represented in the sub grid.

To simplify the interaction the intersection between an object and the water surface is approximated by an ellipse. Instead filtering the ellipse numerically an approximate analytical filter is used.

Chapter 4

Theory and Techniques

In this chapter we describe the techniques used to solve Linear Wave Theory approximately in real time. We describe various approximations made and their implications. We describe how interaction and boundary conditions can be handled. Also a system for adding details to a lower resolution simulation is described.

4.1 Dispersion as Convolution

In this section we will provide a theoretical derivation of the results of section 3.1. While being similar to the derivation in [Tes04a] our derivation is based on Linear Wave Theory rather than Bernoulli's principle and also uses different set of variables: we use the time derivative of h, [Tes04a] use a velocity potential for h.

Simple waves with constant speed can be modeled by the wave equation:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \nabla^2 h \tag{4.1}$$

This ordinary differential equation can easily solved by discretization and Euler stepping in time. However, this requires c to be constant with respect to wavenumber since all wavelengths are handled in the same way. As we recall from section 2.1.1 wave speed relates to water height from bottom, H, and angular wavenumber, k, in the following way:

$$c^2 = \frac{g}{|\mathbf{k}|} tanh \frac{|k|}{H} \tag{4.2}$$

This expression can not be directly used for integrating (4.1) because of its dependence on k. However, by transforming (4.1) into the Fourier domain (4.2) can be directly used.

$$\mathcal{F}\left(\frac{\partial^2 h}{\partial t^2}\right) = c^2 (i\mathbf{k})^2 \mathcal{F}(h) \tag{4.3}$$

Where $\mathcal{F}$ is the Fourier transform. Insertion of (4.2) gives:

$$\mathcal{F}\left(\frac{\partial^2 h}{\partial t^2}\right) = -\frac{g}{|\mathbf{k}|} tanh \frac{|\mathbf{k}|}{H} |\mathbf{k}|^2 F(h) \tag{4.4}$$

Applying the inverse Fourier transform gives the following result, where $\mathcal{F}^{-1}$ is the inverse Fourier transform and $*$ the convolution operator:

$$\frac{\partial^2 h}{\partial t^2} = L * h \tag{4.5}$$

where L is defined as:

$$L = \mathcal{F}^{-1}\left(-g\,|\mathbf{k}|\,tanh\frac{|\mathbf{k}|}{H}\right) \qquad (4.6)$$

This equation can now be solved by computing L and solving (4.5) numerically. While feasible, [Tes04a] does this, solving (4.5) directly requires a lot of computation because of the convolution. If waves of unlimited size are allowed L has to be computed accurately in the whole frequency range resulting in a large kernel. If the range of wavenumbers is limited L can be approximated to be correct within that range and to have a small kernel size. Ideally wavelengths should also be fairly small compared with the grid size to avoid computation on a higher frequency than needed.

4.2 Laplacian Pyramids

Laplacian Pyramids, first introduced in [BA83], have been successfully used to solve various problems within computer graphics. Laplacian pyramids decomposition is a way of representing an image, much similar to wavelets. The basic idea is very simple. To represent an image I using a Laplacian pyramid an iterative algorithm is used. An overview of the algorithm is given in figure 4.1 and an example decomposition in figure 4.2.

In essence this means that each image in the Laplacian pyramid only contains a limited amount of frequencies. Also the wavelengths stored in each image are small compared with the pixels size and the original image is retrieved by summing all images in the pyramid. We can thus conclude that the Laplacian pyramid is a great match for the requirements posed in the end of section 4.1.

Usually, a Laplacian is constructed only once, some calculation are performed on it and then it is converted back into an ordinary image. One way of implementing wave propagation using Laplacian pyramids. Each frame the pyramid is constructed, the waves are propagated and the resulting pyramid is then summed. The results are then drawn and displacements of the water surface are generated. The reason the decomposition algorithm has to be run each frame is to allow for interaction through height displacements. This is a potential bottle neck of the algorithm. The decomposition algorithm requires each image in the pyramid to be down-sampled, blurred and then up-sampled. We therefore propose an implicit construction of the Laplacian pyramids. The basic idea is that instead of running the decomposition algorithm each frame on the whole water surface only the displacements are decomposed. To simplify the decomposition further we propose a new method in section 4.5.

4.3 Approximating L

To take advantage of the water height field representation introduced in 4.2 we need an approximation for L. The goal for this approximation is to accurately approximate L for wavenumbers between j and $2j$ for some values j where j satisfies $j = \frac{C}{h}$ where h is the grid cell width and C is some constant. We have used wavelengths between $6h$ and $12h$. This means that the range of frequencies relative to grid cell width represented in a grid are independent of grid size. The approximation should also be fast to compute.

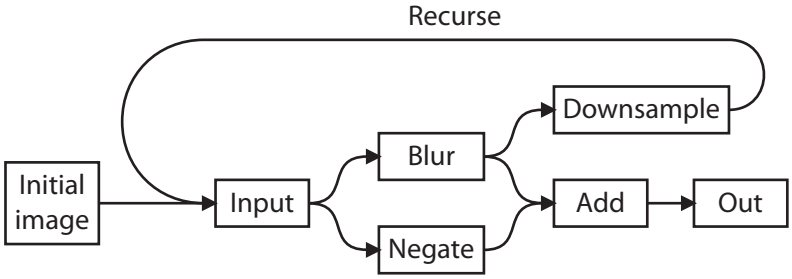

Figure 4.1. Laplacian Pyramid Algorithm.

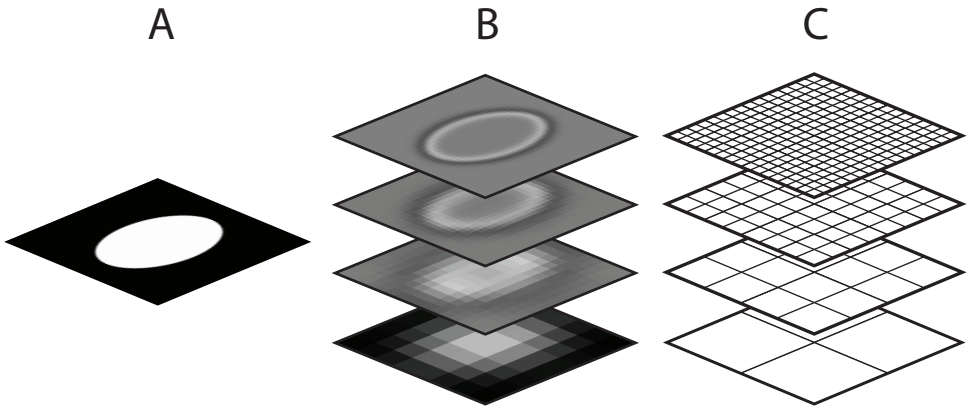

Figure 4.2. Example Decomposition. A shows the original height-field. Black is -1 and white is 1. B the resulting decomposition. 50% gray is 0, white is 1 and black -1. C shows the resolution of the different grids (at a lower resolution than B to exaggerate)

A third requirement is that the approximation should be symmetric. The reason for this is to make the wave speed equal in all directions. We believe that this property is more important than the accurate wavenumber wave speed relation since it is required to keep waves circular.

For this reason a kernel based on a linear combination of a separable Gaussian kernel and an impulse was chosen. The Gaussian kernel is the only kernel that is both rotationally symmetric and separable. This means that a 2D convolution can be made by a 1D convolution first vertically and then horizontally. This means that an $n \times n$ convolution only needs $2n$ reads and 2 writes per cell to be computed instead of n^2 reads and 1 write for a non-separable kernel.

We express this kernel as:

$$A_1\delta(r) + A_2 e^{-B^2 r^2} \tag{4.7}$$

Where δ is the Dirac delta function, r is the distance to center and A_1, A_2, B are weights. To choose the weigths we transformed locally 4.7 into the Fourier domain 4.8.

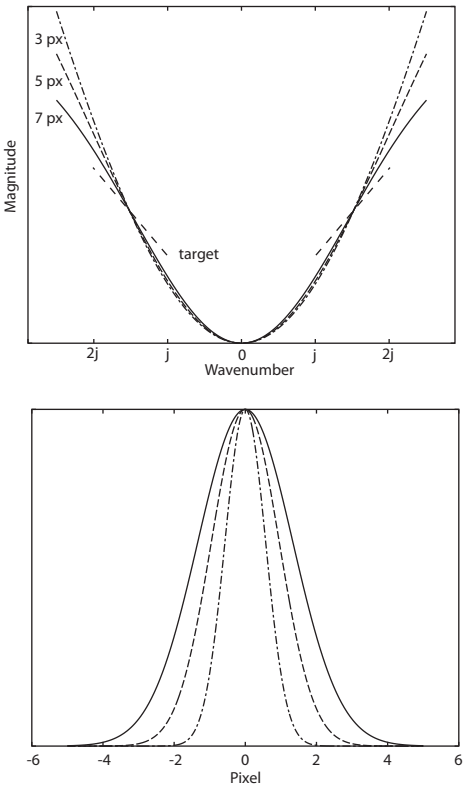

Figure 4.3. Approximation of L. The first plot show L in the Fourier domain. The second plot shows the corresponding Gaussian functions.

Pixels	B	A_1
7	$0.53 \cdot h$	$-5.91 \cdot g \cdot j_{mean}$
5	$0.74 \cdot h$	$-10.2 \cdot g \cdot j_{mean}$
3	$1.24 \cdot h$	$-26.0 \cdot g \cdot j_{mean}$

Table 4.1. Approximation of L: Values for B and A_1

$$\frac{A_1}{\sqrt{2\pi}} + \frac{A_2}{\sqrt{2}B} e^{-\frac{k^2}{4B^2}} \tag{4.8}$$

The weights were determined by testing various values for B and comparing 4.8 with $\mathcal{F}(L)$. When k approaches zero 4.8 approaches $\frac{A_1}{\sqrt{2\pi}} + \frac{A_2}{\sqrt{2}B}$. Therefore $A_2$was chosen so that $\frac{A_2}{\sqrt{2}B} = -\frac{A_1}{\sqrt{2\pi}}$ to keep the simulation stable at low frequencies. A_1 was chosen visually to make 4.8 and $\mathcal{F}(L)$ close in the area of interest, between j and $2j$. We start by choosing A_1, A_2, B for $\mathcal{F}(L)$ with infinite water depths. To make our approximation independent of pixels size and average wavelength we parametrize it based on pixel width grid cell width h and mean wavenumber, j_{mean}. A_2 is not calculated analytically. Instead it is calculated numerically once the Gaussian kernel has been discretized. This is to ensure stability with the discretized kernel. The calculation is done by numerically integrating $I = e^{-B^2 r^2}$ and setting $A_2 = \frac{A_1}{I}$. See figure 4.3 and table 4.1.

In order to handle non infinite water depths we need values for A_1, A_2, B for water depth limited wave speeds. For computational simplicity B is not allowed to vary with depth so that a sub grid will not need multiple B values. Instead we simply clamp A_1 and A_2 to avoid the wave speed from reaching over the shallow water speed.

4.4 Interaction

For interaction simple linear displacement of the height field was chosen for its simplicity. The displacements have to be decomposed to be distributed to the different sub grids in the Laplacian pyramid. Computing the intersection between an arbitrary body and the water surface is complex. Because of this a simple approximation of the intersection is used. All interacting bodies are modeled as ellipsoids and their resulting intersections as ellipses. This makes it possible to do an approximate analytical decomposition rather than numerical de-

composition as we shall see in section 4.5. A horizontally moving body is modeled by increasing the water height in front of it and decreasing it behind. To do this the intersection ellipse is used to modify the height field twice. It is moved forward and used to increase water height and backward to decrease water height. When a body is moved vertically through the water surface the ellipse is used to change the water height dependent on the change of submerged volume.

4.5 Bandpass Filtered Ellipses

In order to generate bandpass filtered ellipses we derive an approximate analytical expression. This expression is correct for ellipses which are considerable larger than the size of the band pass filter.

The basic idea is the following: Let $f(d)$ be the edge response function of a band pass filter . An ellipse can locally be approximated by a line segment. Let $d(x, y)$ be the signed distance function to the ellipse. Then $f(d(x, y))$ will locally represent the bandpass filtered ellipse, given that the filter is much smaller than the line segments, so that curvature does not start to play a role. If we can find an expression for $d(x, y)$ over the whole ellipse $f(d(x, y))$ will have an approximation for the band pass filtered ellipse valid for large ellipses.

We will use two properties of d in our derivation: its gradient is perpendicular to the boundary of the ellipse and the gradient's magnitude is one.

An ellipse is usually described by the following equation:

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1 \tag{4.9}$$

Where x and y are orthogonal coordinates in a coordinate system defined so that the direction of x and y coincide with the major and minor axis of the ellipse and r_x and r_y are the radii along the x and y directions respectively. To find d we start by finding an expression of the form:

$$x^2 + y^2 = r^2(x, y) \tag{4.10}$$

We therefore propose $r^2(x, y)$ of the form:

$$r^2(x, y) = \frac{r_x^2 r_y^2 (x^2 + y^2)}{r_y^2 x^2 + r_x^2 y^2} \tag{4.11}$$

We need to prove that 4.9 and 4.10 are equivalent:

$$x^2 + y^2 = \frac{r_x^2 r_y^2 (x^2 + y^2)}{r_y^2 x^2 + r_x^2 y^2} \tag{4.12}$$

$$\Leftrightarrow 1 = \frac{r_x^2 r_y^2}{r_y^2 x^2 + r_x^2 y^2} \tag{4.13}$$

$$\Leftrightarrow r_y^2 x^2 + r_x^2 y^2 = r_x^2 r_y^2 \tag{4.14}$$

$$\Leftrightarrow \frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1 \tag{4.15}$$

Using this expression we can find an approximation for d : $\sqrt{x^2 + y^2} - \sqrt{r^2(x, y)}$. This expression equals d when the ellipse is a circle since it reduces to the distance from origin minus the radius. For ellipses it is inaccurate since the gradient of $x^2 + y^2$ is not perpendicular to the boundary of the ellipse. To

compensate for this we divide with the gradient of x^2+y^2 projected on the normal of the ellipse $(\frac{x}{r_x^2}, \frac{y}{r_y^2})$. This gives the following results.

$$\frac{1}{\sqrt{\frac{x^2}{r_x^4}+\frac{y^2}{r_y^4}}}\left(\frac{x}{r_x^2},\frac{y}{r_y^2}\right)\cdot\frac{1}{\sqrt{x^2+y^2}}(x,y)$$

$$=\frac{1}{\sqrt{\left(\frac{x^2}{r_x^4}+\frac{y^2}{r_y^4}\right)(x^2+y^2)}}\left(\frac{x^2}{r_x^2}+\frac{y^2}{r_y^2}\right)$$

$$=\frac{r_y^2x^2+r_x^2y^2}{\sqrt{\left(r_y^4x^2+r_x^4y^2\right)(x^2+y^2)}}$$

$$=\left(\frac{r_y^4x^4+2r_x^2r_y^2x^2y^2+r_x^4y^4}{r_y^4x^4+(r_x^4+r_y^4)x^2y^2+r_x^4y^4}\right)^{1/2}$$

$$=\left(1+\frac{(r_x^2-r_y^2)^2x^2y^2}{(r_y^2x^2+r_x^2y^2)^2}\right)^{-1/2}$$

Using this we can find an expression for d:

$$d=\frac{\left(\frac{x^2}{r_x^2}+\frac{y^2}{r_y^2}\right)^{1/2}-\left(\frac{r_x^2r_y^2(x^2+y^2)}{r_y^2x^2+r_x^2y^2}\right)^{1/2}}{\left(1+\frac{(r_x^2-r_y^2)^2x^2y^2}{(r_y^2x^2+r_x^2y^2)^2}\right)^{1/2}} \tag{4.16}$$

Laplacian pyramids use Gaussian filters as their basic low pass filter. To make the analytical bandpass filtering possible we need a band pass filtered step function to use as a basis. We have three requirements for this function:

- Fast to calculate
- Short tail
- Good frequency response

The chosen function was selected by testing various simple function, consisting of only addition, multiplications and a single division and comparing them with a Gaussian filter and a perfect band pass filter. See figure 4.4 for a comparison. Based on its simplicity and short tail we chose:

$$f(d)=\frac{ad}{1+a^6d^6} \tag{4.17}$$

4.6 Level of Detail

In computer games predictable memory and computation requirements are important. A game has to run at a high frame rate for the experience to be enjoyable. It is therefore better to sacrifice quality for performance. Because of this we require our level of detail algorithm to be close to constant in both memory and computation independent of observer and water configuration.

As we shall see the Laplacian pyramid approach is very well suited for adaptive resolution. The simulation is done by simulating several sub grids $(h_1, h_2, h_3, \ldots, h_i)$. Different resolution is easily obtained by varying i. For example 5 sub grids could be used close to the observer, 4 further away and 3 even further away etc.

In order to achieve constant memory and performance we decided to base our level of detail system on a pool. The pool contains pyramids of different resolutions. These are dynamically distributed to achieve varying level of detail. To make the change of resolution easy, the area which should be simulated

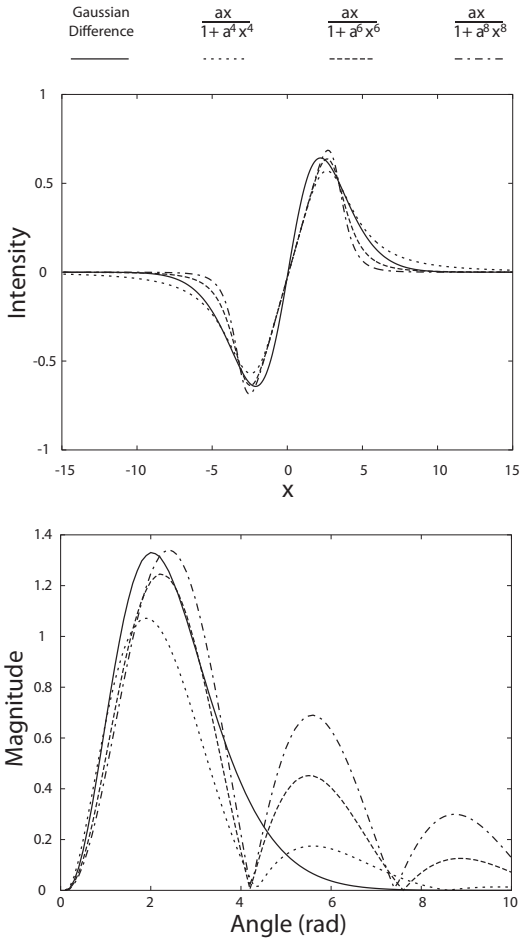

Figure 4.4. Bandpass Filtered Edges. First plot shows the edge response of different functions. Second plot the frequency plot of the corresponding bandpass filter. a is a scaling factor. $a = 0.3$ was used in the example.

(this is defined by the user) is split into equal sized squares (for example 6 x 6 m in size). We refer to these as simulation cells. The pyramids are at run-time distributed to the simulation cells so that cells close to the observer get high resolution pyramids while cells further away get lower resolution. This results in constant performance independent of simulated area. However, the bigger area needed to be simulated the lower resolution it will be, relative to distance to the observer. When there are so many cells that there are not enough pyramids the cells furthest away will not be simulated.

4.7 Phenomenological Generation of Details

Real world water has waves down to just a few millimeters in size. With our approach and current processors it is impossible to achieve this amount of detail in real-time. Because of this an interesting area to study is phenomenological addition of details. Phenomenological approaches have been used successfully to add details to smoke simulations [BHN07, KTJG08]. It is therefore interesting to consider similar methods for height field based water. [KTJG08] use wavelet noise to modify the flow of a low resolution simulation. This method is not applicable since our simulation assumes zero flow. Instead a method that modifies the height-field is needed. An early approach at generating waves for computer games is to render a billboard with a circular wave on it and animate the billboard scaling. While an interesting approach it is hard to integrate with the rest of the simulation. Ideally the detail addition could be done mainly in a

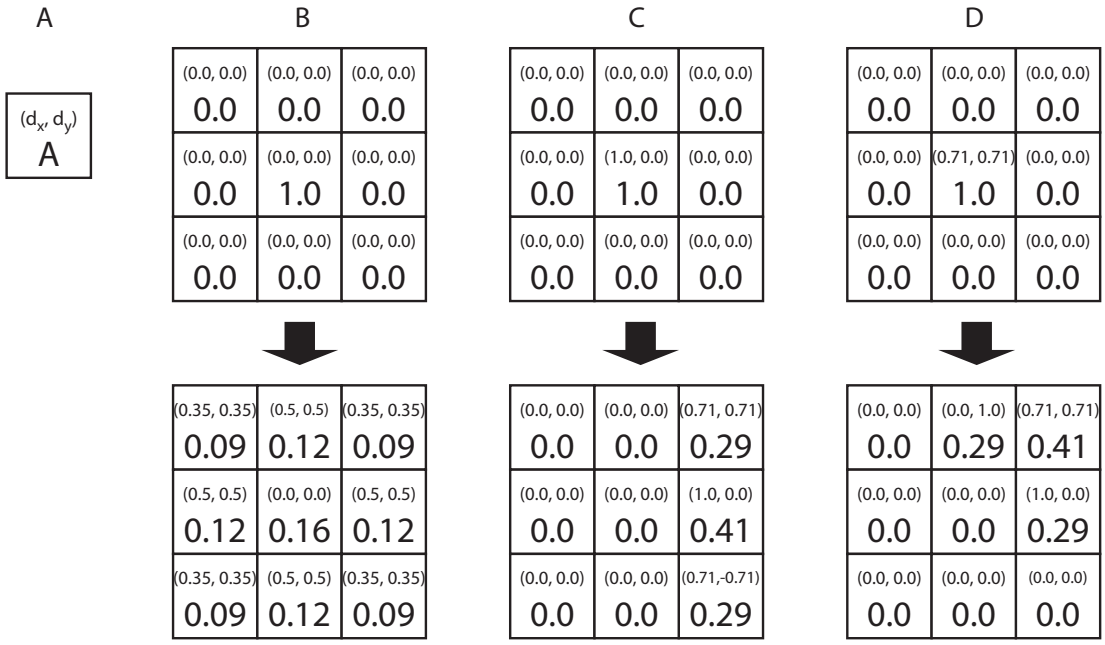

Figure 4.5. Propagation of Detail Waves. A explains the meaning of the different values. B, C and D show example propagations.

pixel shader.

Instead we propose a method for modeling the existence of waves rather than a height field. The idea is that instead of propagating waves information about wave direction and amplitude is propagated. Simulation is done in a grid. Each grid cell contains the wave energy, E, and a 2d vector containing the average direction of the waves, d, this vector is not normalized. Because of this the d contains information about how directional the waves are. If $|d| = 0$ no direction is favored , if $|d| = 1$ they are completely directional. This representation is clearly inaccurate and violates the superposition principle. However our results using the approach are promising.

A method of propagation is needed to use this representation. We base the propagation on three simple observations:

- If $|d| = 1$ they should propagate in that direction and maintain $|d| = 1$.
- If $|d| = 0$ one possible reason for this is that the waves are traveling in all directions. In that case they should propagate in all directions and become more and more directional.
- Sum of E for all cells should be conserved.

Based on this E is propagated in two ways dependent on $|d|$. E is split into two parts: directional $|d|E$ and non-directional $(1 - |d|)E$. The non-directional part is diffused equally in all directions. The directional part weighted by the scalar product between d and the direction to the adjacent cell. Energy is only moved in the directions where the

scalar product is positive. See figure 4.5 for examples. Propagation speed is controlled by the amount of energy being propagated.

E and d are then exposed in a pixel shader to generate the actual details.

4.8 Phenomenological Boundary Conditions

Correct handling of boundary conditions is a hard problem. [Tes04a] uses a phenomenological approach and simply sets the water displacement to zero where there is no water. This results in a rigid boundary condition which generates reflected waves. However these waves are completely unphysical since the water height is kept constant at borders. A better approximation would be a free boundary condition. Using this boundary condition is more complicated. Therefore we use a simple approximation: water displacement is modeled by diffusion where there is no water. Physically this is very inaccurate and may generate both energy and mass. In practice it is a cheap way of generating plausible boundary interactions and also has a dampening effect on wave amplitude. The diffusion is cheap to calculate since we already calculate L which can directly be used to calculate the diffusion.

This model does not take non linear effects of water boundaries. A large wave generates smaller waves when interacting at a boundary. In order to model these effects we allow waves in low resolution sub grids to move into higher resolution sub grids near borders. This also compensates for the energy loss of the boundary condition approximation.

Chapter 5

Implementation

We have created two implementations of the methods and algorithms described in chapter 4. One implementation is created using GNU Octave to create easy analyzable results. We have also implemented the algorithm of [Tes04a] for comparison. The Octave implementation is made directly from the equations presented in 4. The Octave implementation does not feature the LOD system. The second implementation is made using C++ in the Frostbite engine. This implementations is made to see how our algorithm works in a real computer game engine and is highly optimized. The Frostbite implementation is used for performance evaluation.

The rest of this chapter is focused on the Frostbite implementation and how our algorithm is optimized and adapted to the game engine architecture.

5.1 Update method

We solve eq. 4.5 using Euler integration in the following way:

$$v_{n+1} = v_n + \Delta t\,(L * h_n)$$

$$h_{n+1} = h_n + \Delta t\, v_{n+1} + \Delta h_{displacement}$$

Where n is the last step, $n+1$ is the step to be calculated, v the velocity,Δt the change in time and $\Delta h_{displacement}$ is the change in water height due to interaction. To solve this the height and velocity is stored in two buffers, one for the current frame and one for the previous. The two buffers alternate role so that the first buffered is updated from the second and in the next frame the second from the first.

To make sure the simulation is stable Δt is clamped at a maximum value Δt_{max}. This results in slower wave speed for large Δt rather than unstable simulation.

5.2 Parallelism

To fully utilize the Parallelism of modern computers and game consoles it is important to take into account when implementing high performance code. There are several kinds of parallelism. Two important types, which we will focus on, are vectorization and processor parallelism. Vectorization refers to utilizing Single In-

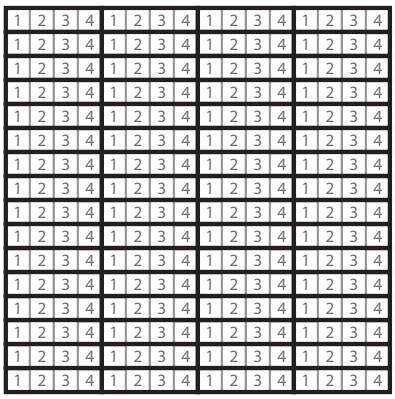

Figure 5.1. Vectorization.

struction Multiple Data (SIMD) capabilities. Processor parallelism to the usage of multi core architectures to run code on several processors in parallel.

If fully utilized parallization can provide huge speed ups. A Cell processor, which is used in PlayStation 3, has 6 independent processors that each are capable of doing 4 instructions on 32 bit floating point number. This is makes it possible to process 24 numbers at once.

The Frostbite engine is using a job model to handle processor parallelism. This means that to parallelize a task it has to be divided into a number of independent jobs which are then scheduled to the many different cores. Frostbite also has a cross platform vector library to simplify vectorization.

5.2.1 Vectorization

To utilize SIMD capabilities the sub grids are stored as vectors of length four aligned in the x-direction as shown in figure 5.1. Most parts of the simulation is trivial to vectorize since it does not depend on neighbor cells. The two parts that are harder are up-sampling and convolution. Up-sampling and convolution in the y-direction is unchanged. In the x-direction it can be calculated by matrix multiplication. Since up-sampling and convolution are similar we only show how convolution is done in more detail. Calculating the convolution in the x-direction with a 9 cells wide vector $\mathbf{w}(w_0, w_1, \ldots, w_8)$ in an height field represented by the 4 long column vector $h_{x,y}$ (each x represents 4 values, y represents 1 value) can be done in the following way:

$M_{before} \times h_{x-1,y} + M_{middle} \times h_{x,y} + M_{after} \times h_{x+1,y}$

Where

$$M_{before} = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 \\ 0 & w_0 & w_1 & w_2 \\ 0 & 0 & w_0 & w_1 \\ 0 & 0 & 0 & w_0 \end{bmatrix}$$

$$M_{middle} = \begin{bmatrix} w_4 & w_5 & w_6 & w_7 \\ w_3 & w_4 & w_5 & w_6 \\ w_2 & w_3 & w_4 & w_5 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix}$$

$$M_{after} = \begin{bmatrix} w_8 & 0 & 0 & 0 \\ w_5 & w_6 & 0 & 0 \\ w_6 & w_7 & w_8 & 0 \\ w_5 & w_6 & w_7 & w_8 \end{bmatrix}$$

5.2.2 Processor parallelism

To be able to use the job model for parallelism the simulation has to be split into smaller problems. The Cell processor consists of 7 processing units. 6 of these, called Synergistic Processing Units (SPU's), do not have direct access to main memory and only have 256 kB of cache for both code and data. Because of this jobs can only access 256 kB simultaneously. To meet these requirements wave propagation is split into two part: a simulation step and a border copy step. All grids contain a border outside the simulated area. This border is used to calculate the Laplacian in the propaga-

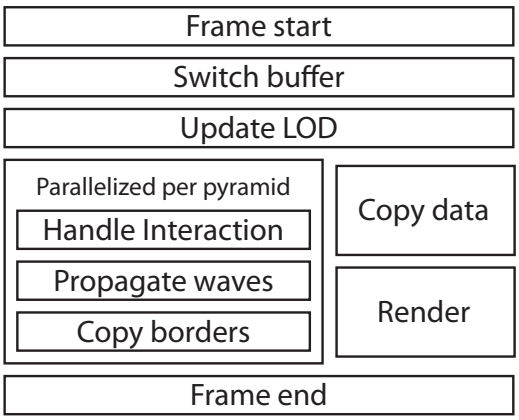

Figure 5.3. Algorithm Overview. The frame starts at the task "Frame start" and progresses downwards. Tasks beside each other are performed in parallel.

tion step. In the border copy step two neighbors are processed simultaneously and their border are filled with the simulated data from their neighbor.

The height-field has to be copied to the GPU for rendering. In order to do this in parallel with simulation the copying is done on the data calculated in the last frame. This results in a slight time difference between simulation and rendering. However this has not been noticeable in practice.

5.3 Algorithm Overview

Figure 5.3 contains an overview of the work done during a frame.

5.4 Client and Server

The multiplayer games at EA DICE™ use a client server model. Critical systems such as player movement, vehicle physics simulation, large scale destruction etc run on the server and are mirrored on the client. Effects and less important physics objects such as smoke, newspaper blowing in the wind, debris from destruction etc are only simulated on the client. We will refer to objects only existing on the client as client side objects and objects being simulated at the server and mirrored at the client as server side objects.

For performance reasons it was decided that the water simulation should only run on the client. If the simulation was to be run on the server level of detail methods could not have been used and lots of network traffic had been needed. Because of this a server side simulation would have had substantially reduced resolution.

Client side objects can interact fully with the water simulation. They both generate waves and respond to them. Server side objects do not have access to the height field simulation and can therefore not be affected by waves. Instead server side objects use the resting water height. Since server side objects are mirrored onto the client they can generate waves in same way client side objects do. This makes it less obvious that server objects are not affected and in practice it is barely noticeable unless large waves are present (comparable to the interacting object in size).

5.5 Rendering

This is only a sort overview of the rendering since this was not the focus of this thesis. This is an important area of future work.

To render the height field a vertex buffer and a texture map is used. The vertex buffer contains a triangle representation of all the grids and the phe-

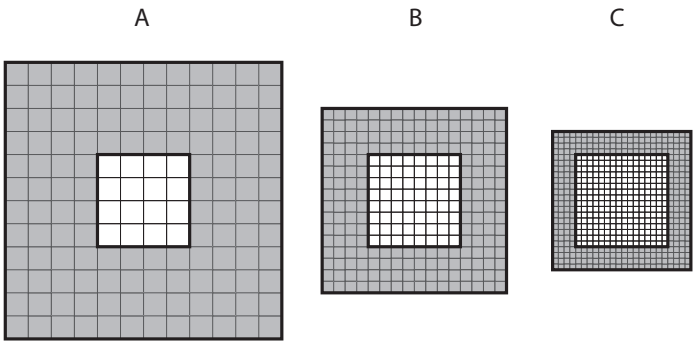

Figure 5.2. Border sizes for Different Grid Resolutions.

nomenological detail model. The texture map the height values which are used for normal calculation. For simplicity all grids are represented in the whether they are simulated or not. However, only the grids that are currently active are rendered.

The vertex buffer is generated at a lower resolution than simulation to increase performance of the rendering while the texture map is generated at full resolution. The textures for all the grid are stored in a single texture using a texture atlas approach.

The normals are generated from the texture map by computing the gradient. In addition to this details are added in the vertex buffer using the phenomenological detail simulation. The direction is used to blend in an animated texture moving in that direction. The energy is used to scale the amplitude of the animated texture. The texture is animated using the method of [Vla10].

In order to provide smooth transitions between LOD levels the highest resolution grid in a pyramid are faded towards any neighbors with lower resolution.

5.6 Physics Interaction

The Frostbite engine has existing systems for handling rigid body physics and buoyancy. Our simulation interacts with this system in three ways.

- The bottom height is determined by checking vertical rays for intersection with static geometry
- Interacting bodies are updated with the current water height at their position to handle buoyancy
- Interacting bodies are approximated with an ellipsoid to find an intersection with the water surface to generate waves. See 4.4.

Chapter 6

Results

6.1 Octave Implementation

These are the results of the GNU Octave implementation. Wave speeds where calculated by measuring zero crossings of a standing wave. The iwave implementation uses a kernal constructed by a numerical inverse Fourier transform of eq. (4.6).

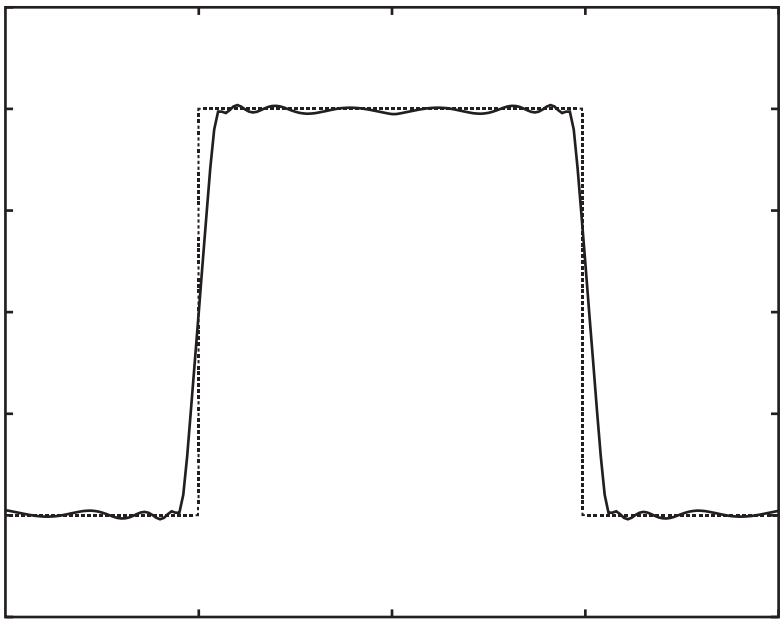

Figure 6.1. Sum of eight bandpass filtered functions.

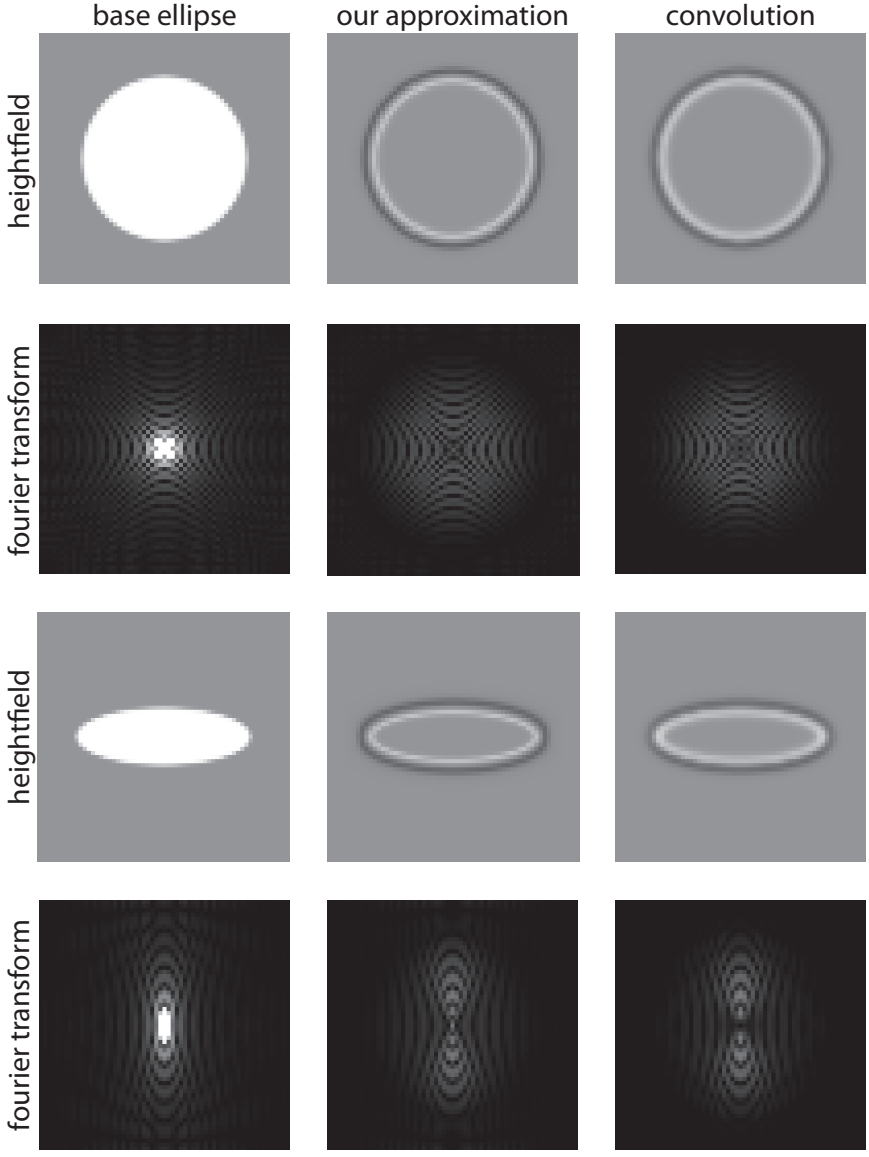

Figure 6.2. Bandpass Decomposition.

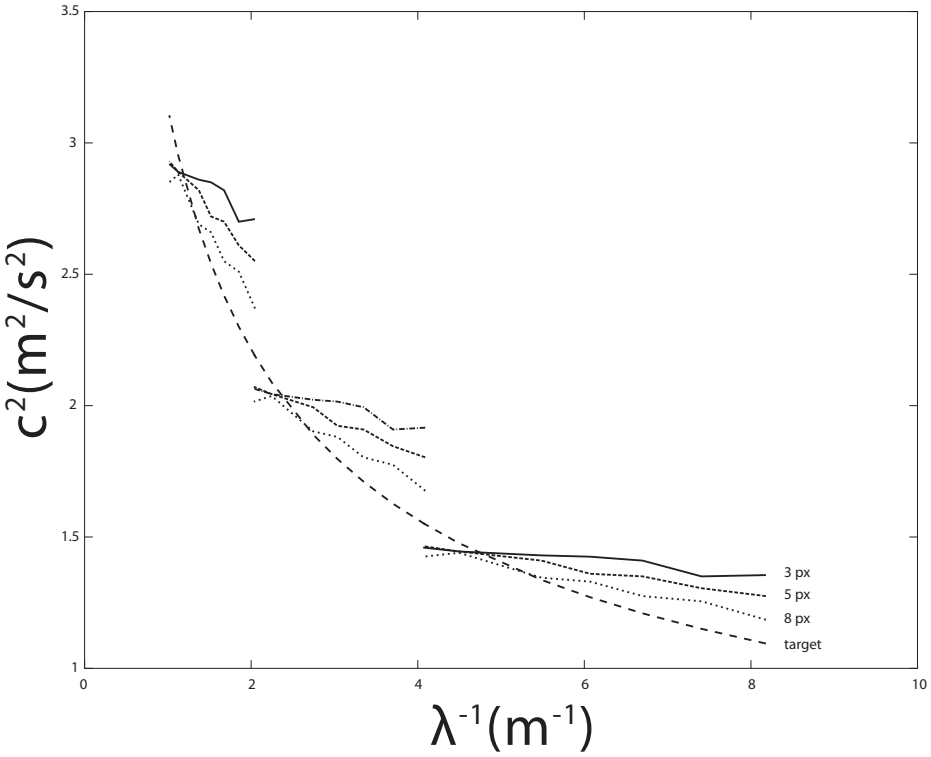

Figure 6.3. Measured wave speed.

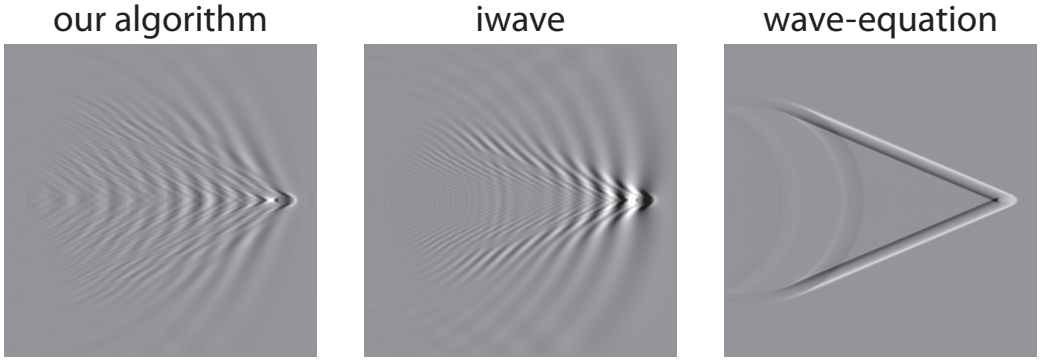

Figure 6.4. Height-field Results Comparison. 50% gray represents 0, lighter gray positive and darker gray negative.

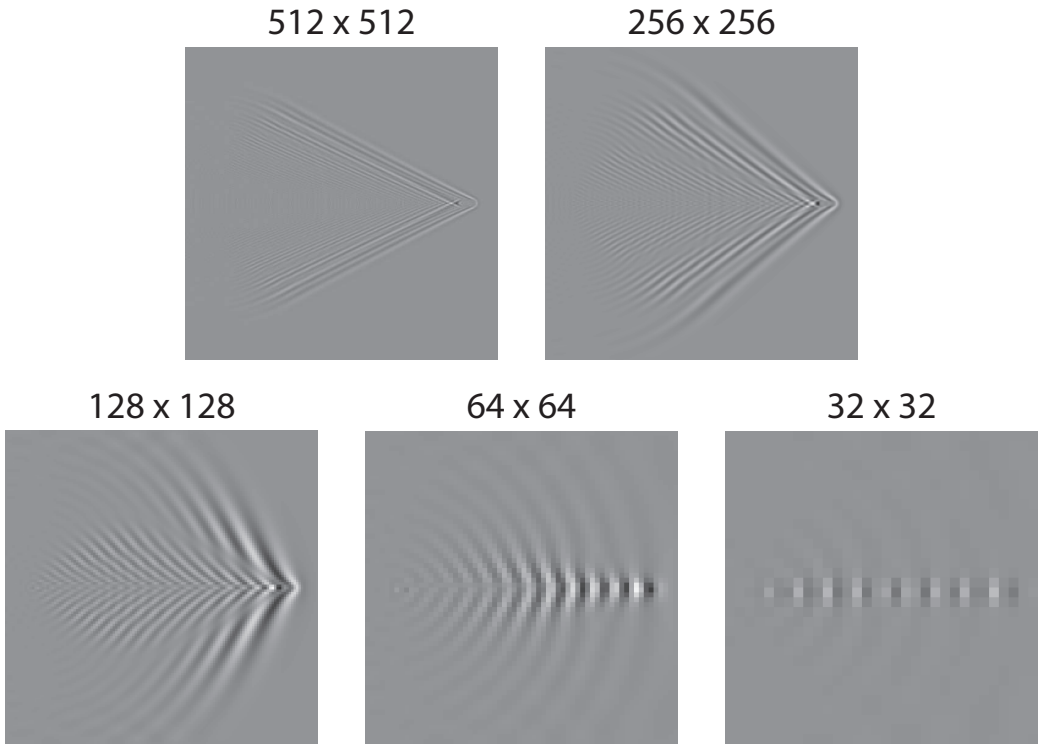

Figure 6.5. Grids in the Pyramid. 50% gray represents 0, lighter gray positive and darker gray negative.

6.2 Frostbite Implementation

These are the results of the optimized implementation. All measurements where made on a 8 core Intel Xeon x5550 processor running at 2.67 GHz with 12 GB of RAM and a Nvidia GeForceGTX 470 graphics card.

Pyramid resolution	Time per frame (ms)
128	6.5
64	2.1
32	0.9
16	0.5

Table 6.1. Time Measurements for 8 Connected Pyramids. Measurements where made in Frostbite with 8 pyramids of different resolutions. The measurements show the time spent on propagation and border copying.

No. of pyramids with resolution				Time per frame spent on	
64x64	32x32	16x16	All	Simulation (ms)	Rendering (ms)
4	12	0	16	2	0.5
6	24	0	30	4	0.7
4	12	36	52	3.1	0.7
6	18	54	78	4.5	1.0
4	16	64	84	4.3	0.9
6	24	96	126	6.5	1.3

Table 6.2. Time Measurements for Various Pyramid Configurations. Measurements where made in Frostbite. The measurements show the time spent on propagation and border copying and the amount of time spent on the CPU to generate vertex buffers and textures.

Chapter 7

Discussion

The method presented in this thesis is a new method for simulating water surface waves. This method is fast enough to be usable in computer and video games and provides an approximation of linear wave theory. In particular both water height and wave length dependency on wave speed is handled which creates complex effects impossible without it.

Formal quality and performance comparisons are important areas which have not been adressed in this thesis. Because of a lack of reference implementaions and because of the dynamic level of detail system performance is hard to compare. Quality is also hard to evaluate since the goal is not numerical accuracy. This could be evaluated by a user study.

7.1 Design Choices

7.1.1 Linear Wave Theory

We have successfully used linear wave theory as a basis for real time water surface waves simulations. Linear wave theory is also used to derive the iWave method which makes it possible to do a qualitative comparison based on the common base.

7.1.2 Wave decomposition

We have presented a novel method for approximating linear wave theory by doing a wavelength based decomposition similar to Laplacian pyramids. This allows heavy approximation of L while requiring 33% more memory than previous methods. Because of the multiresolution approach it is well suited for level of detail methods. We believe that this decomposition has been a success. However, our approximation of L and decomposition algorithm leave room for further developments.

7.1.3 Approximation of L

The Gaussian approximation of L is one of the simplest possible. While being fast to compute it is not an accurate approximation. Because of the goal of being usable in current generation video games and consoles a more expensive approximation is impossible. In figure 6.4 our algorithm does not feature as strong inference pattern (Kelvin wakes) as the iWave algorithm, which correctly solves the linear wave theory dispersion relation. With a better approximation L our results would have been closer to the

iWave results. However, while being less accurate, our method accounts for height dependence.

7.1.4 Level of Detail

The level of detail system presented automatically adapts resolution of the simulation while keeping the performance constant. One big disadvantage of the pool-based method is that the total possible area of the simulation is held constant. This becomes a problem when the observer is moved far from a large water surface. Also as the number of pyramids increase in the pool border copying becomes an larger issue. For 4x4 grids the size of the border is 8 times larger than the grid itself. See figure 5.2.

7.1.5 Interaction

All intersections are modeled as ellipses. In the implementation in the Frostbite engine this has not limited the interactions. Finding accurate intersection for use more advanced system is currently too costly to be of practical use. The ellipse method allows for fairly varied interaction without becoming overly complex.

7.1.6 Phenomenological boundaries and details

The phenomenological boundaries and details improves the visual quality by adding interest and detail. Naturally an accurate high resolution simulation would be better but given the constraints on performance this is not possible.

7.2 Future Research

7.2.1 Different approximations of L

Our approximation of L is fast to compute but not accurate. For future games and other applications a more complex approximation may be of interest. L could potentially be modeled as a sum of several Gaussian functions, two keep the separable property. Another alternative would be to generate a two dimensional kernel computed directly from a windowed frequency response of L.

7.2.2 Hybrid Methods

Hybrid methods are common in fluid simulations. See for example [Cor08]. Using out method in conjunction with another model is an interesting future area of research. One way of doing this is to couple our linear wave theory based method with another height field method such as shallow water equations. Shallow water equations do not account for dispersion effects but handles flow. An interesting use would be to couple a low resolution shallow water equations solution with a high resolution linear wave theory solution. To do this the linear wave theory solver would need an advection step. Another similar possibility is to couple a static laminar flow solution with our solution. to simulate rivers etc which have a close to constant flow.

Coupling our simulation with a turbulence model could also be interesting.

7.2.3 Level of Detail

Researching more advanced LOD methods for our multiresolution simulation is an important area of future research.

The current system uses equal sized pyramids of varying resolution. This limits the algorithm to a maximum area: the total area of all the pyramids. Several other methods were considered but not implemented due to time constraints. A promising approach is to have a pool of equal resolution grids. The pyramid hierarchy is created globally by dynamically varying the scale of the grids. This makes it possible to grid to transition from one scale to another making the LOD algorithm independent of scene scale. The equal size of grid also removes the problem of border copying becoming a larger performance hit for small grids.

7.2.4 Rendering

In order for the simulation to be perceived as realistic in a computer or video game the visualization needs to be realistic. This research is beyond this thesis but is still an important field of research.

7.2.5 Interaction

We have limited ourselves to interactions through ellipses. A more advanced approach using general intersections and numerical bandpass filtering should be investigated. Also a more accurate band pass filter should investigated.

7.2.6 Boundary Conditions

The boundary conditions we have used are phenomenological in their nature. A more scientific approach should be considered.

7.2.7 Performance Comparisons

Comparing performance is important to find the most efficient method. With level of detail methods and different approximations such a comparison is hard to do. Nevertheless such comparisons are important and so far have not been done with height-field based methods. To do such a study it is important to find a good metric of simulation quality.

7.2.8 Other uses

The multiresolution method for solving linear wave theory is applied to simulation for computer and video games. Combined with a better approximation for L it could potentially be used for non-real-time simulations. Another possibility is to use this multiresolution methods with higher order methods and other similar computational problems.

Bibliography

[BA83] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3):46, 2007.

[BMF07] R. Bridson and M. Müller-Fischer. Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 courses*, page 81. ACM, 2007.

[CL95] J.X. Chen and N.D.V. Lobo. Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations. *Graphical Models and Image Processing*, 57(2):107–116, 1995.

[Cor07] H. Cords. Mode-splitting for highly detailed, interactive liquid simulation. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 265–272, New York, NY, USA, 2007. ACM.

[Cor08] Hilko Cords. Moving with the flow: Wave particles in flowing liquids. In *Journal of WSCG (WSCG'08)*, 2008.

[Day09] Mike Day. Insomniac's water rendering system, 2009. retrieved from http://www.insomniacgames.com/tech/articles/0409/files/water.pdf.

[Fin04] Mark Finch. Effective water simulation from physical models. In *GPU Gems*. Addison-Wesley Professional, 2004.

[FS06] G. Falkovich and K.R. Sreenivasan. Lessons from hydrodynamic turbulence. *Physics Today*, 59(4):43, 2006.

[Gre08] Simon Green. Particle-based fluid simulation for games. http://developer.nvidia.com/object/gdc-2008.html, 2008. Retrieved September 10th 2010.

[HHL+05] T.R. Hagen, J.M. Hjelmervik, K.-A. Lie, J.R. Natvig, and M. Ofstad Henriksen. Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory*, 13(8):716 – 726, 2005. Programmable Graphics Hardware.

[Igl04] A Iglesias. Computer graphics for water modeling and rendering: a survey. *Future Generation Computer Systems*, 2004.

[JG01] Lasse Staff Jensen and Robert Golias. Deep-water animation and rendering. http://www.gamasutra.com/gdce/2001/jensen/jensen_01.html, 2001. Retrieved September 10th 2010.

[Kal08] Daniel Kallin. Real time large scale fluids for games. *Linköping Electronic Conference Proceedings (Proceedings of SIGRAD 2008)*, 2008.

[KID10] Jeffrey Kiel, Kumar Iyer, and Sebastien Domine. Taking fluid simulation out of the box: Particle effects in dark void. http://developer.nvidia.com/object/gdc-2010.html, 2010. Retrieved September 10th 2010.

[KLLR05] B.M. Kim, Y. Liu, I. Llamas, and J. Rossignac. Flowfixer: Using bfecc for fluid simulation. In *Eurographics Workshop on Natural Phenomena*, volume 1. Citeseer, 2005.

[KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM.

[KTJG08] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):1–6, 2008.

[LH10] Hyokwang Lee and Soonhung Han. Solving the shallow water equations using 2d sph particles for interactive applications. *The Visual Computer*, 26:865–872, 2010. 10.1007/s00371-010-0439-9.

[Lov02] J. Loviscach. A convolution-based algorithm for animated water waves. In *Eurographics*, volume 2, pages 381–389, 2002.

[Lov03] J. Loviscach. Complex water effects at interactive frame rates. *Journal of WSCG*, 11:2003, 2003.

[LvdP02] Anita T. Layton and Michiel van de Panne. A numerically efficient and stable algorithm for animating water waves. *The Visual Computer*, 18:41–53, 2002. 10.1007/s003710100131.

[MK99] A.J. Majda and P.R. Kramer. Simplified models for turbulent diffusion: Theory, numerical modelling, and physical phenomena. *Physics Reports*, 314(237):574, 1999.

[NCZ+09] M.B. Nielsen, B.B. Christensen, N.B. Zafar, D. Roble, and K. Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 217–226. ACM, 2009.

[Nie10] Michael B. Nielsen. Improved variational guiding of smoke animations. *Computer Graphics Forum*, 29:705–712(8), May 2010.

[PTSG09] Tobias Pfaff, Nils Thuerey, Andrew Selle, and Markus Gross. Synthetic turbulence using artificial boundary layers. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, pages 1–10, New York, NY, USA, 2009. ACM.

[rea] Realflow. http://www.realflow.com/. Retrieved September 10th 2010.

[Sch07] R. Schuster. Algorithms and data structures of fluids in computer graphics. *Unpublished State of the Art Report*, 2007.

[Tes04a] Jerry Tessendorf. Interactive water surfaces. In *Game Programming Gems 4*. Charles River Media, 2004.

[Tes04b] Jerry Tessendorf. Simulating ocean surface. SIGGRAPH 2004 Course Notes, 2004. retrieved from http://tessendorf.org/reports.html September 10th 2010.

[TKPR06] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. Detail-preserving fluid control. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 7–12, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Vla10] Alex Vlachos. Water flow in portal. Advances in Real-Time Rendering in 3D Graphics and Games, SIGGRAPH 2010 Course Slides, 2010. retrieved from http://advances.realtimerendering.com/s2010/index.html September 10th 2010.

[YHK07] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):99, 2007.