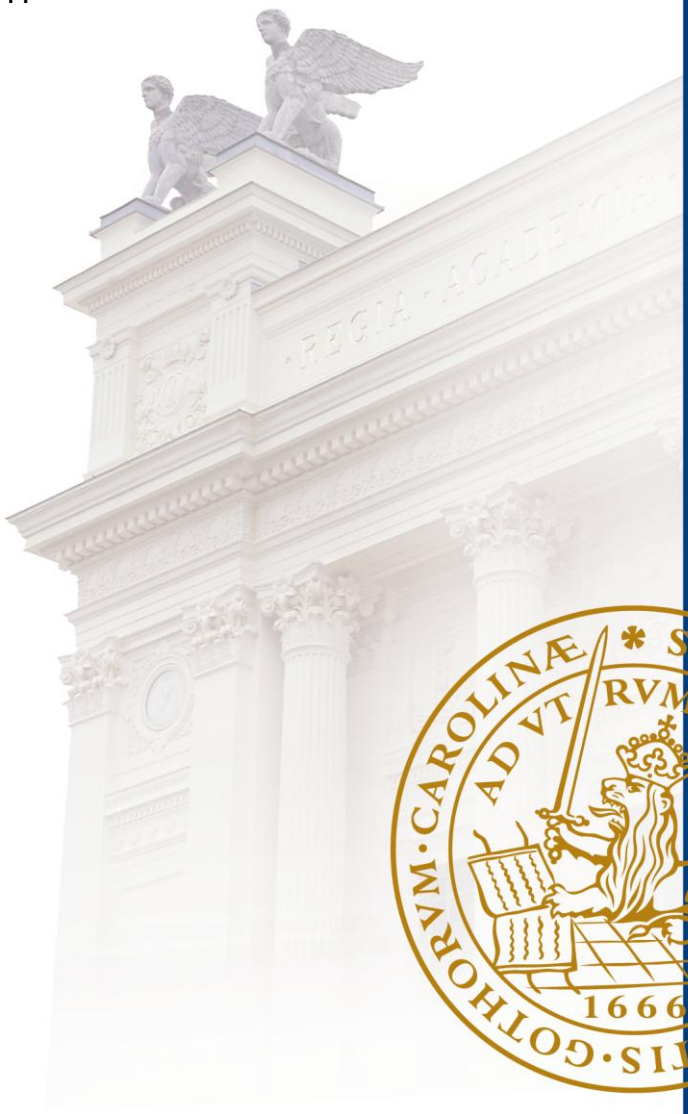
Master's Thesis

# Deterministic Ocean Waves

Fredrik Larsson

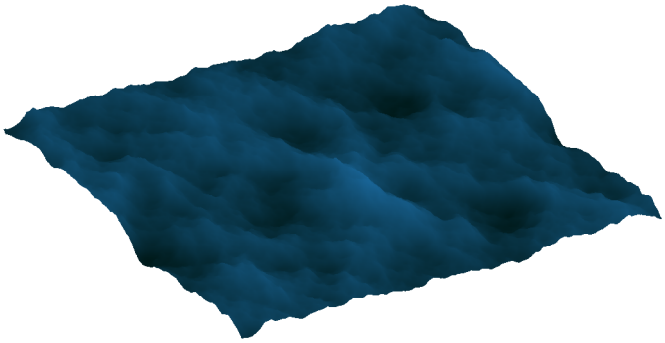
Department of Computer Science
Faculty of Engineering LTH
Lund University, 2012



ISSN 1650-2884
LU-CS-EX: 2012-18

M.Sc. Thesis in Computer Science

# Deterministic Ocean Waves



Fredrik Larsson
Lund University
Institute of Technology, LTH
Department of Computer Science

**Advisors:**
Torbjörn Söderman, EA DICE
Björn Ottosson, EA DICE
Michael Doggett, Lund University

August 20, 2012

**Abstract**

In this master's thesis a system for deterministic ocean wave simulation is presented. The target application is interactive multi-client computer games, where the system can render a visual representation of the ocean surface as well as provide surface data to other systems, such as a buoyancy physics simulation. Ocean waves caused by wind above the water surface are simulated by propagating waves in the Fourier domain, where a Fast Fourier Transform algorithm is used for efficient computations. Additional larger individual waves are represented by a basic particle system in an algorithm inspired by Wave Particles. A prototype implementation was integrated into an existing system for interactive water waves in the Frostbite engine developed by EA DICE. The simulation can, with reasonable quality, achieve sub-millisecond run times on current generation gaming consoles and PC hardware.

**Sammanfattning**

I denna masteruppsats presenteras ett system för deterministisk oceanvågsimulering. Målapplikationen är interaktiva datorspel för multipla klienter, där systemet kan rendera en visuell represenation av havsytan och dessutom kan tillhandahålla data till andra system, så som fysiksimuleringar för flytande objekt. Havsvågor som orsakas av vind ovanför havsytan simuleras genom vågpropagering i Fourierdomänen, där en algoritm för snabb Fouriertransform används för att uppnå tillräcklig prestanda. Större individuella vågor representeras av ett partikelsystem där en algoritm som inspirerats av Vågpartiklar tillämpats. En prototyp har implementerats i ett existerande system för interaktiva vattenvågor i spelmotorn Frostbite, som utvecklas av EA DICE. Simulering kan med rimlig kvalitet uppnå beräkningstider på under 1 millisekund på denna generations spelkonsoler och PC-hårdvara.

# Contents

# List of Figures

# Chapter 1

# Introduction

When simulating and rendering a physically correct world, sooner or later one usually needs to address the issue of representing and visualizing a body of water. Such a body of water may occur in varying sizes and appearances ranging from small rain puddles to vast stormy oceans. Water simulations have over the last decades seen numerous takes, both in interactive applications, off-line image generation and simulation of fluid dynamics. Common to all methods is the aim of achieving a high level of physical accuracy while maintaining low computational cost. Simulation with interactive frame rates, which is the topic of this thesis, is of course more leaned towards keeping the needed computational effort low and is satisfied with results at a plausible level.

## 1.1 Scope

In this thesis, the simulation of ocean waves is discussed, with the target application being multi-client computer games. The goal is to represent two types of waves occurring on the ocean surface: ambient waves and larger individual waves. The ambient waves should provide a simulated representation of a water surface of oceanic proportions under given weather conditions. The ability to manually place large individual waves on top of the simulated surface can be used to design specific gameplay events, for example a dam breaking or a tsunami.

One use of the simulation results in this application is to render a visual representation of the water surface. This involves supplying the underlying rendering system with a mesh representation of the simulation results. Another use is to provide other systems in the application with responses to water level queries. For example, a physics simulation system may need such information in order to simulate the motion of an object floating in the water body.

A multi-client game usually involves a server application in addition to

the clients. It is essential that the water surface affects the game world equally on all involved clients, including the server. To achieve this, the ocean surface needs to be fully deterministic across all clients, for any given point in time. The server and clients are connected via a network, usually the Internet, which imposes further issues in terms of latency and synchronization, something that also needs to be addressed by the system.

This thesis is focused on finding a suitable model for the problem formulation given above, and presenting a prototype implementation of it. The main focus is obtaining an efficient implementation of a simulation of ambient wind-driven waves that can potentially be incorporated in today's games. The implementation of individual wave entities is complemental and presented as a proof of concept. The implementation will be incorporated into the Frostbite engine, developed by EA DICE, where it needs to integrate with other sub-systems of the engine. For this reason, the available resources for the system are highly restricted, which introduces further performance requirements. The ocean wave simulation also needs to integrate with an existing system for water interaction. The interaction simulation is local to each client and is used for generating and propagating waves caused by disturbances of the water surface.

For any component in a game engine it is essential that a high enough level of control is exposed to the user. The user here is an artist or game designer, who must be able to tweak the ocean waves to obtain the visual appearance and game experience they want. Furthermore, it is also important to be able to adjust the fidelity of the simulation for devices on both ends of the performance scale.

## 1.2 Outline

A brief presentation of previous research in the area is discussed in chapter 2, where a number of different approaches for both ocean wave simulation and interactive wave simulation are discussed. The restrictions and features offered by the target system for the reference implementation are presented in chapter 3. These two chapters are used as a motivation for the chosen method, which is outlined in chapter 4. The reference implementation of the method is presented in chapter 5, with the results in chapter 6. Finally, a discussion on the results and suggestions for future improvements concludes the report in chapter 7.

# Chapter 2

# Background

## 2.1 Fluid Simulations

Any work concerning simulation of fluids is almost obliged to mention the Navier-Stokes equations. These are a set of partial differential equations describing the movements of a fluid. The equations are of non-linear nature, and thus very difficult to solve. In the field of computational fluid dynamics (CFD) various numerical methods are applied in order to solve the equations. The methods of CFD are often applicable for off-line simulations of fluid phenomena and require large amounts of computational power. If interactive frame rates are the target, the methods from CFD must be discarded in favour of ones of less computational intensity. As in many other situations in real-time computer graphics, one is forced to resort to approximations of more or less severity to get a good balance between computational time and result plausibility.

As will be shown in this chapter, many of the methods aiming for interactive frame rates reduce the otherwise three-dimensional problem to a similar one in two dimensions. This reduction means that only waves present on the water surface are simulated, discarding phenomena such as vertical flow and turbulence.

Many methods presented here also use a heightfield to represent the water surface. A heightfield is a map from a two-dimensional point $\boldsymbol{p}$ to a corresponding vertical displacement. As the heightfield is animating, the map also depends on time, meaning that a function $h(\boldsymbol{p}, t)$ is the mathematical representation of the heightfield. The heightfield representation introduces further restrictions, for example the inability to represent breaking waves.

## 2.2 Ocean Surface Simulation

A well known and early model of the ocean surface was presented by Fournier and Reeves, which utilizes the theory of *Gerstner waves*. In the Gerstner

model, water particles exhibit elliptical stationary orbits around their resting points. This gives the waves their characteristic shape with sharp crests and wide troughs [FR86]. The model assumes that the ocean surface can be described by a sum of a number of such waves. For this reason, as the number of waves increases the method quickly becomes expensive because of the need to evaluate a high number of sine functions each frame.

A more intricate solution, which successfully handles a high number of waves is Tessendorf's Fourier domain method. His work is based on the findings of Mastin, Watterberg and Mareda who use a statistical approach for their model [MWM87]. The model originates in oceanographic research, in which observations of ocean surface lead to the derivation of a frequency spectrum for surface waves. Using this method, the wave propagation can be performed efficiently in the Fourier domain, to be later transformed to the spatial domain using a Fast Fourier Transform (FFT) algorithm. This method has been successfully used in both off-line renderings in movies [Tes04b], and also in interactive applications [Mit07].

Another approach worth mentioning is to use fractal noise to generate the heightfield. With Perlin noise of different frequencies and amplitudes a plausible ocean surface can be acquired. In combination with a level of detail algorithm, Yang et al. used this method for rendering an unbounded ocean [Yan+05]. This method lacks the foundation of a statistical observation as the previously mentioned method has. However, as mentioned by Yang et al., this method has the benefit of being easily implemented while maintaining low computational cost.

As an extension to the heightfield representation, the method described by Thüerey et al. successfully simulates breaking waves. By using shallow water simulations, the characteristic steepening of waves as they approach a shore or reef is simulated. Then, by detecting and tracking the steep wave fronts, a set of particles are spawned that can be used to represent a patch of wave that spills down in front of the wave [Th07].

In most cases only a small part of the ocean surface is visible in screen space at a given time, yielding it unnecessary to render most of it. A popular method of limiting the rendered surface is to use a *projected grid*, which is described in detail in Claes Johansson's master thesis [Joh04]. The basic idea is to use a uniformly spaced grid in screen space, which is later projected onto the water surface. In world space, this yields a fine grid resolution close to the viewer, and coarser further away towards the horizon. This approach has also been used successfully by numerous other researchers [YHK07; CS09].

A good example of a practical application of water simulation techniques for computer games can be found in Naughty Dog's title *Uncharted*. The method described is highly artist driven, and not so much a simulation method, but is still interesting because it describes actual production use of techniques described here. The authors discarded the methods of Tessendorf described above, with the motivation that the parameters offered by the sim-

ulation were difficult to interpret and tweak to get the desired appearance of the water surface. Instead, a few larger Gerstner waves are used to create the large billowing waves of the ocean, with smaller high-frequency waves superimposed on top of those. Other techniques applied involve representing large individual waves, where a wave-shaped B-spline curve is used to define the shape of a wave. This wave can then be individually spawned and animated as desired. Also, a mask encoding flow and local deviations in amplitude is used for further tweaking of the resulting water surface [GOH12].

## 2.3 Interactive Water Surfaces

The already present water interaction simulation in the target system of this thesis' prototype implementation is based on linear wave theory and uses convolution to propagate waves over a heightfield [Ott10]. Tessendorf describes a similar algorithm called *iWave* [Tes04a], and because of the similarities the two methods are comparable in terms of visual and computational performance [Ott10]. The simulation has been further optimized and parallelized to run efficiently on current gaming consoles, and use a quadtree based LOD (Level of Detail) algorithm as described in Lennartsson's thesis [Len12].

Also based on linear wave theory, Cords and Staadt describe a method for highly detailed simulation of interactive waves. They introduce moving grids to efficiently limit the simulation area, instead of using a LOD scheme. For rigid-body physics, a particle-based approach is used which realistically simulates the object-liquid and liquid-object interaction. The method is combined with the Fourier-based ocean wave simulation method by Tessendorf to simulate the ambient ocean surface [CS09].

Another useful simulation method is to use the shallow water equations. These are simplifications of the Navier-Stokes equations making them usable in real-time applications. Chentanez and Müller describes a method which is based on these equations. In addition, they use a hybrid approach which extends the otherwise limiting heightfield representation. By using a particle based simulation in situations where the heightfield is not enough, breaking waves, waterfalls, vortices and other phenomena can be simulated [CM10].

Using Lattice-Boltzmann methods originating from CFD, Geist et al. present a slightly different simulation method. The method employs a two-dimensional discrete lattice in which mass can flow between neighboring points. Their wave propagation method uses a collision matrix to encode how the mass flows between points. In contrast, the previously mentioned iWave algorithm relies on convolution for propagation but the two methods are comparable in terms of needed computational power [Gei+10]. The authors implemented their method using the compute capabilities of modern graphics cards to obtain interactive frame rates.

A slightly different approach in simulation of interactive surface waves is the concept of *wave particles* which was originally proposed by Yuksel, House and Keyser. This model takes a large number of simultaneous interactions into consideration and successfully simulates phenomena such as wave reflection with low computational effort. The method involves using a simple particle system, where the particles represent perturbations on the water surface. The particles can be treated independently from each other, meaning large amounts of particles can be present simultaneously in the system. Using the graphics processor, the particles can be converted to a heightfield using a simple wave form function [YHK07].

# Chapter 3

# Overview

This chapter gives a description of the premises for the proposed system. The local water interaction simulation that the system integrates with is described, along with existing engine features such as rigid body physics simulation and networking.

## 3.1 Local Interactive Simulation

The water interaction simulation consists of a number of steps which are outlined below. The intention is not to give a full description, but rather an overview which is needed for the following chapters. The following steps make up the algorithm, and are described in more detail in the following sections.

1. Determine which area of the water surface to simulate by considering the client's world position.
2. Find which objects are currently inside the volume of water and displace the heightfield appropriately.
3. Animate the heightfield using results from previous steps.
4. Render the total heightfield.

The simulation is run locally on each connected client with no data being interchanged between clients, and the server does not run the simulation at all. For this reason it is not possible for the results of this simulation to influence objects in the global game world, and the results are only used for local visual effects.

### 3.1.1 Level of Detail

The simulation requires a discrete grid to represent the heightfield, where finer resolution grids give a simulation of higher fidelity and the possibility

to simulate waves with shorter wavelength. However, finer resolution grids also leads to higher computational cost per area unit of water surface. To overcome this, it is useful to limit the resolution in areas where the simulation does not add any significant improvement in visual realism. The less important pieces of surface area are typically ones that are far away from the client's point of view. In the existing simulation system the approach used to implement such a scheme is to equip each simulated body of water with a quadtree, which gives a hierarchical representation of the water body's surface area. Each node in a quad tree contains a data structure here called a *cell*, which represent a square piece of water surface. The root node of each quadtree is a cell with the same dimension $D_0$, as the entire water surface. Descendant cells at level $i > 0$ of the quadtree have the dimension $D_i = \frac{D_{i+1}}{2}$.

In order to give predictable memory usage a constant number of cells is allocated when the system starts. This means that each tree node only has an implicit cell, and a cell is only explicitly added to the hierarchy when needed. In each step of the simulation the quadtree is updated by prioritizing cells according to the client's point of view. Cells which get a high enough priority are added to the tree hierarchy, while those with a priority lower than a user defined threshold are removed. Next, by using a user defined range of dimensions $R = [D_s, 2D_s, \ldots, ND_s]$ the simulation system can attach grids to the cells which have a dimension $D \in R$. These grid-equipped cells are in a later step those to be considered by the actual simulation.

A common problem when dealing with LOD systems is *popping*. This occurs when swapping between different levels of detail and destroys the illusion of a seamless mesh or texture. To overcome this, the interactive water system does not instantly remove or add grids to the hierarchy, but instead fades the values in the grid towards or from zero over time. This introduces inertia and the need for frame-to-frame persistence of LOD data.

### 3.1.2 Heightfield Generation and Propagation

When the grids to use for the simulation have been determined, the next step is to apply disturbances to the heightfield from the previous step. This is done by first finding all objects that are currently inside the water volume. Then, by using the object's velocity and its approximate shape the heightfield is disturbed, taking care to preserve the water body's total volume. Next, the heightfield is propagated by convolving each grid causing the waves to animate over time. Extra care is also taken along the borders of each grid, as the waves generated by a disturbance must be allowed to travel between grids. For that reason, each grid has a border region which is copied between neighboring cells. The underlying theory of this procedure is not in the scope of this thesis, but is discussed in greater detail in the

works of Ottosson and Lennartsson [Ott10; Len12].

The final step before rendering the heightfield is to apply a bicubic upsampling step. This can be seen as adding the results of a low-resolution simulation to its children of higher resolution. The higher resolution grids have twice the resolution per area unit which calls for the need of upsampling. Another feature of the existing simulation system is that the borders of heightfields in quadtree cells which do not have a gridded sibling are linearly faded to 0. The intention of this is to avoid gaps in the resulting mesh, which would otherwise be evident.

### 3.1.3 Rendering

The very same quadtree based approach as described above is used to obtain more efficient rendering of the heightfield. A gridded cell with simulation data is used to create vertices with uniform horizontal distance and with its vertical position displaced according to the heightfield. The cells which do not have simulation data will each be rendered as a simple flat quadrilateral with vertices in the corners of the cell. For rendering purposes the normal of each vertex is also needed. The approach used by the interactive water system is to approximate the normals on the GPU. The heightfield resulting from the simulation is rendered to a two-dimensional texture which is later used by the shader to compute an approximate surface normal.

## 3.2 Physics Simulation

Modern game engines often have extensive systems for simulation of physical phenomena, such as rigid body dynamics. For the purpose of this thesis the interesting part of the rigid body physics simulation is the module handling floating objects. Among the information needed by this simulation is the water surface height $h(\boldsymbol{p}, t)$, where $\boldsymbol{p}$ is a point in world space coordinates and $t$ is the simulation time. During an update pass of the physics simulation the function needs to be evaluated for several different points $\boldsymbol{p}$, which is proportional to the number of objects floating in the body of water.

### 3.2.1 Networking and Prediction

In a networked environment, latency is an inevitable problem that needs to be addressed. Latency can be as high as several hundred milliseconds, which, if not handled appropriately, will cause unacceptable delays for the clients. One way of dealing with latency issues is to utilize a client-side prediction mechanism which can effectively hide such delays. As an example, if a client issues a command to move forward in the game world, the command would first need to reach the server, where the position is updated. At a later time, the updated position is sent to all clients. The delay before the client receives

the updated position is approximately equal to its round-trip latency to the server, which is often large enough to be noticeable.

With prediction however, the client instantly predicts the effect of a command before sending it to the server. Assuming an authoritative server, there is a possibility that the predicted result is not equal to the correct result computed on the server. This may occur because of commands from other clients not having arrived to the client in question yet, and means the prediction took place with invalid premises. Such a misprediction situation can be solved in various ways, for example temporally interpolating a position from the predicted position and the actual position. Implications to keep in mind when employing the described prediction scheme is that objects controlled by the local client will live ahead of the authoritative server time. For the purpose of the system presented here, the implications are that there is a possibility that the water height function needs to be evaluated for times in the future with respect to the actual time.

# Chapter 4

# Method

## 4.1 Ocean Waves and the Fast Fourier Transform

Given the already present representation of the water surface, the ocean surface simulation method described by Tessendorf was deemed suitable for simulation of wind-driven waves in this application. The system in place is equipped with a quadtree organized set of uniform grids with equal resolutions, making it a good fit for incorporating this method. A heightfield representation of the water surface is already in place, and the chosen method can be incorporated with little interference with the original implementation. As the method has been used successfully in both movie productions and games, the method can provide the physical realism required.

For the rest of this section, the theory behind Tessendorf's method is described in detail. The theory is also adapted to better fit the implementation aspects, such as meeting performance requirements, integration with the present system, and being able to use the simulation deterministically in a multi-client environment. Another intention of this theoretical section is to identify and extend the set of simulation parameters available, so that the system can provide sufficient user control of the simulation.

### 4.1.1 Features

The method produces a tiled discrete heightfield of the desired resolution. The tiling feature means that the generated heightfields can be placed side by side to get a continuous water surface of the desired dimension. However, if the rendered water surface area is much larger than the tile, the repeating tile pattern causes unwanted artefacts.

An ocean surface with a high level of realism will need to obtain the characteristic choppy look of ocean waves. The method can achieve this by creating an additional two-dimensional displacement field which can be used to add perturbations to the points of the otherwise uniform grid. Moving

grid points horizontally towards crests, and thus away from troughs will simulate this behavior.

The method is also fully deterministic, which is crucial in this application. The determinism comes from the fact that for any time $t$, the resulting heightfield is only dependent on a fixed set of initial parameters. Also, for computing a heightfield at time $t = t_0$ no information is needed from previous results for times $t < t_0$, meaning no frame-to-frame persistence of data is required.

### 4.1.2 Notation

The theory described in this section operates in a three-dimensional space, where the $y$-axis of the space's coordinate system is pointing upwards. The simulated water surface thus lies in parallel with the $xz$-plane, with the base water height $y = y_0$. A wave travelling on the $xz$-plane can be expressed using a function of a point $\boldsymbol{p} = (p_x, p_z)$ on the plane and the time $t$:

$$\Phi(\boldsymbol{p}, t) = A\cos(\boldsymbol{k} \cdot \boldsymbol{p} - \omega t + \theta)\,,$$

where $A$ is the amplitude of the wave, $\omega$ is the angular frequency and $\theta$ its phase. In the expression above, the vector $\boldsymbol{k} = (k_x, k_z)$ is the *wave vector*, which points in the waves' propagation direction and has the magnitude $|\boldsymbol{k}| = \frac{2\pi}{\lambda}$, inversely proportional to the wavelength $\lambda$. This number $k = |\boldsymbol{k}|$ is also called the waves' *wavenumber*. Using this notation, the water surface height is the function

$$h(\boldsymbol{p}, t) = y_0 + \sum_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{p}, t)\,, \tag{4.1}$$

which is the sum of the base water height and all waves present on the water surface.

Another convenient notation for waves in general comes from Euler's formula. The formula gives an expression for a sinusoidal wave in terms of complex-valued functions

$$A\cos(\omega t + \theta) = \mathrm{Re}\{Ae^{i(\omega t + \theta)}\} = \mathrm{Re}\{Ae^{i\theta} \cdot e^{i\omega t}\}\,.$$

In the above, the complex number $Ae^{i\theta}$ encodes the amplitude and phase of the wave, and is commonly called its *complex amplitude* or *phasor*.

As stated before, the simulation works by generating smaller areas of tiling water surface. These tiles are discrete grids with a resolution $R_x \times R_z$ discrete samples and cover an area of $D_x D_y\,\mathrm{m}^2$. The resolution and dimension of the tiles can be chosen depending on application, but for the rest of this theoretical section a square tile with $R_x = R_z = R$ and $D_x = D_z = D$ will be used for simplicity's sake. Also note that, since an FFT

algorithm will be applied to the discrete grid, it is usually convenient to keep $R$ a power of two.

The described method works on a discrete heightfield, which is given by the water surface height function 4.1 evaluated at the points $\boldsymbol{q} = \frac{D}{R}\boldsymbol{r}$, where $\boldsymbol{r} = (m, n)$ is an index vector with $m, n \in [0, \ldots, R-1]$.

### 4.1.3 Wave Distribution

Looking at the expression in 4.1 the summation region is all wave vectors $\boldsymbol{k}$ present on the water surface. This section describes how to generate a set of such wave vectors and also how to obtain the corresponding amplitude and phase. Operating in the frequency domain a wave vector is assigned to each of the $R^2$ discrete grid points. With $m$ and $n$ being row and column indices, a wave vector for that grid point is:

$$\boldsymbol{k}_{mn} = \frac{2\pi}{D}(n - \frac{R}{2}, m - \frac{R}{2}) = \frac{2\pi}{D}(\boldsymbol{l} - \frac{\boldsymbol{R}}{2}), \tag{4.2}$$

where the vectors $\boldsymbol{R} = (R, R)$ and $\boldsymbol{l} = (n, m)$ are used for convenience. The result of this is a discrete field of wave vectors with the magnitude of the vectors increasing, and thus with wavelength decreasing, radially from the center of the field.

Oceanological research has lead to the observation that the complex amplitudes for an ocean surface fluctuate spatially in the frequency domain. The only factor that influence the waves' fluctuations is the wind velocity $\boldsymbol{w} = (w_x, w_z)$. The fluctuations can then be described by the *Phillips spectrum* [Tes04b]

$$P_h(\boldsymbol{k}) = A\frac{e^{-1/(kL)^2}}{k^4}|\hat{\boldsymbol{k}} \cdot \hat{\boldsymbol{w}}|^2,$$

where $A$ is a scalar constant, the $\cdot$ operator denotes the dot product of two vectors, and $\hat{\boldsymbol{v}}$ denotes the unit length counterpart of a vector $\boldsymbol{v}$. The dot product term will cause waves travelling in a direction perpendicular to the wind to be cancelled out, while leaving those propagating in a direction closer to the upwind and downwind directions.

This spectrum is what will determine the end result of the simulation, and from a user's perspective it does not provide many knobs to turn. For this reason, a slightly altered Phillips spectrum is used in the proposed system. With this altered spectrum the wave distribution can be *narrowed* towards the wind direction, giving a more evident directionality of the waves. This can be done by increasing the power of the dot product term. It is also possible to scale waves travelling in either direction, to further increase the directionality. By defining a scaling function $d(x)$ and power function $p(x)$ a modified Phillips spectrum is:

$$P'_h(\boldsymbol{k}) = \frac{e^{-1/(kL)^2}}{k^4}C(\hat{\boldsymbol{k}} \cdot \hat{\boldsymbol{w}}), \tag{4.3}$$

where $C(x) = d(x) \cdot |x|^{p(x)}$. Here, a simple step function was chosen for the scaling

$$d(x) = \begin{cases} d_+ & \text{if } x \geq 0, \\ d_- & \text{if } x < 0, \end{cases}$$

and analoguously for the power $p(x)$.

Using the Phillips spectrum for the given set of waves, an initial field of complex amplitudes can be generated with

$$\tilde{h}_0(\boldsymbol{k}) = \frac{\xi}{\sqrt{2}} \sqrt{P'_h(\boldsymbol{k})}\,,$$

where $\xi = \xi_r + i\xi_i$ is a complex number and $i$ is the complex unit $i^2 = -1$. $\xi_r$ and $\xi_i$ are independent draws from a gaussian distribution with mean 1 and standard deviation 0. This set of complex amplitudes constitutes the phases and amplitudes of $R^2$ waves on the water surface. These amplitudes only need to be computed and stored once for the simulation, and are used in subsequent steps for animating the surface.

### 4.1.4 Wave Dispersion

To realistically animate water surface waves, the velocity with which a wave propagates needs to be replicated. The simple property that gives this behavior is concluded in the *dispersion relation* for water waves [Tes04b], which couples a waves' wavelength to its angular frequency. Using the above notation the relation comes down to:

$$\omega^2(k) = gk \tanh(kd)\,,$$

where $d$ is the water depth and $g$ is the gravitational constant. For the open ocean where $d$ is usually very large compared to the wavelength, the hyperbolic tangent term will approach 1, yielding the even simpler relation

$$\omega^2(k) = gk\,.$$

For the purpose of this simulation this means that for any wave $k$, the angular frequency and thus the propagation velocity is given, and also constant.

Now, using the dispersion relation, the complex amplitudes for a given time $t$ can be computed with

$$\tilde{h}(\boldsymbol{k}, t) = \tilde{h}_0(\boldsymbol{k}) e^{i\omega(k)t} + \tilde{h}_0^*(-\boldsymbol{k}) e^{-i\omega(k)t}\,. \tag{4.4}$$

Here, $^*$ denotes the complex conjugate operator. This expression maintains the reality condition for Fourier transforms, which states that if a function is strictly real-valued, the imaginary part of the Fourier transform exhibits odd symmetry:

$$\mathrm{Im}\left\{f(t)\right\} = 0 \;\Rightarrow\; F(-s) = F^*(s)\,,$$

where $F(s) = \mathcal{F}\left\{f(t)\right\}(s)$, and $\mathcal{F}$ is the Fourier transform.

### 4.1.5 Transforming

The results so far are still in the frequency domain, and need to be transformed to the spatial domain in order to evaluate the water surface height function 4.1. This transformation is obtained by [Tes04b]

$$h(\boldsymbol{q}, t) = \sum_{\boldsymbol{k}} \tilde{h}(\boldsymbol{k}, t) e^{i\boldsymbol{q}\cdot\boldsymbol{k}}, \tag{4.5}$$

which closely resembles the definition of the inverse 2-dimensional Discrete Fourier Transform (DFT) of size $N \times N$

$$f_{\boldsymbol{k}} = \frac{1}{N^2} \sum_{\boldsymbol{n}} F_{\boldsymbol{n}} e^{i\frac{2\pi}{N}\boldsymbol{k}\cdot\boldsymbol{n}},$$

where the sum is nested over both indices $\boldsymbol{n} = (m, n)$, $m, n \in [0, \ldots, N-1]$. Recalling the expression for $\boldsymbol{k}$ in equation 4.2 and that $\boldsymbol{q} = \frac{D}{R}\boldsymbol{r}$, equation 4.5 can be rewritten to instead use the summation variable $\boldsymbol{l}$, and parameter $\boldsymbol{r}$. With this substitution, the expression is in fact the DFT, with a shift of $\frac{R}{2}$ in each dimension of the input signal.

$$\begin{aligned} h(\boldsymbol{q}, t) &= \sum_{\boldsymbol{l}} \tilde{h}(\boldsymbol{q}, t) e^{i\frac{2\pi}{R}\boldsymbol{r}\cdot(\boldsymbol{l}-\frac{\boldsymbol{R}}{2})} = \\ &= \left[\sum_{\boldsymbol{l}} \tilde{h}(\boldsymbol{q}, t) e^{i\frac{2\pi}{R}\boldsymbol{r}\cdot\boldsymbol{l}}\right] e^{-i\pi\frac{\boldsymbol{R}}{R}\cdot\boldsymbol{r}}. \end{aligned}$$

By further expansion, and using that $e^{-i\pi} = -1$ and $\frac{\boldsymbol{R}}{R} = (1, 1)$ the shifted input is compensated for by multiplying with an alternating sign factor $(-1)^{m+n}$:

$$\begin{aligned} h(\boldsymbol{q}, t) &= \left[\frac{1}{R^2}\sum_{\boldsymbol{l}} \tilde{h}(\boldsymbol{k}, t) e^{i\frac{2\pi}{R}\boldsymbol{r}\cdot\boldsymbol{l}}\right] R^2(-1)^{(m+n)} = \\ &= \mathcal{F}^{-1}\left\{\tilde{h}(\boldsymbol{k}, t)\right\} R^2(-1)^{(m+n)}. \end{aligned} \tag{4.6}$$

### 4.1.6 Additional Data

By using even more Fourier transforms, additional required data can be obtained [Tes04b]. To be able to render the heightfield it is essential to have access to the surface normals of the heightfield. Recalling the Fourier transform of a functions derivative

$$\mathcal{F}\left\{f'(t)\right\}(s) = 2\pi i s\, F(s),$$

and applying this to the field of complex amplitudes

$$\nabla h(\boldsymbol{q}, t) = \sum_{\boldsymbol{k}} i\boldsymbol{k}\, \tilde{h}(\boldsymbol{k}, t) e^{i\boldsymbol{k}\cdot\boldsymbol{q}},$$

yields the partial derivatives in the $x$ and $z$ directions. Having obtained the derivatives, the surface normal at the grid point $\boldsymbol{q}$ is

$$\boldsymbol{n}(\boldsymbol{q}, t) = (-\nabla h_x(\boldsymbol{q}, t), 1, -\nabla h_z(\boldsymbol{q}, t)) .$$

Finally, to obtain the choppy look of ocean waves, the horizontal displacement vectors also need to be computed. Again, by using even more Fourier transforms, the two-dimensional displacement vector field is computed with

$$\boldsymbol{D}(\boldsymbol{q}, t) = \sum_{\boldsymbol{k}} -i\hat{\boldsymbol{k}}\, \tilde{h}(\boldsymbol{k}, t) e^{i\boldsymbol{k}\cdot\boldsymbol{q}} .$$

The final horizontal position of the grid points is thus $\boldsymbol{q} + \boldsymbol{D}(\boldsymbol{q}, t)$. A pitfall when computing the displacement field is that it is possible for the displacement of adjacent points to overlap. While it is possible to detect such situations, the simple solution is to bias the displacement with a factor $\lambda$ which can be set by the user, yielding $\boldsymbol{q} + \lambda \boldsymbol{D}(\boldsymbol{q}, t)$.

## 4.2 Wave Entities

To represent large individual waves an algorithm inspired by the wave particles, as proposed by Yuksel et al. was selected. While the method is originally proposed as a simulation method for interactive water surfaces [YHK07], the method has features which makes it suitable in this situation as well.

### 4.2.1 Particle System

For representing the waves on a water surface a very simple particle system is used. For efficiency reasons, the particles do not interact with other particles in the system and most of a particles' properties remain constant during its lifetime. An advantage to draw from this, is that if the initial position $\boldsymbol{p}_0$ for a particle is known, the position at any other time $t > t_0$ kan be easily determined with

$$\boldsymbol{p}(t) = \boldsymbol{p}_0 + (t - t_0)\boldsymbol{v}, \tag{4.7}$$

where $\boldsymbol{v}$ is the velocity of the particle. This makes the behavior of each particle completely deterministic over time. Another advantage to exploit from this is that there is no need to update each particle position every frame. Instead, the position is computed as needed with the expression in 4.7, greatly reducing the number of memory writes [Yuk10, p. 89].

The properties needed for finding the position of each particle is thus initial position, creation time and velocity. Furthermore, the wave particles must maintain two more properties: amplitude and radius, which define the shape of the wave as the particles are later converted to the height field.

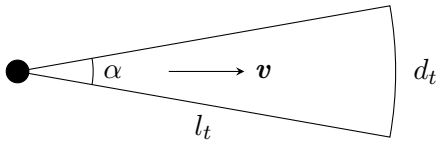



Figure 4.1: Approximation of time for subdivision event for particle with velocity $\boldsymbol{v}$ and dispersion angle $\alpha$.

Also, to represent expanding wavefronts, another parameter is introduced which is the dispersion angle of a particle. For radially expanding waves, new wave particles must be spawned in order to preserve the shape. The distance between particles on the same wavefront should optimally never exceed half the particle radius [YHK07], which is why subdivision of particles is necessary. Naïvely approached, the problem of finding subdivision candidates can become expensive and introduce dependencies between particles. A better approach is to instead treat every particle independently, which can be done by introducing a few assumptions regarding particle properties yielding an efficient analytical solution. Considering a particle on a wavefront with speed $v = |\boldsymbol{v}|$, dispersion angle $\alpha$ and radius $r$, the distance travelled by the particle in $t$ seconds is $l_t = t\,v$. The length of the arc given by the angle $\alpha$ and a circle of radius $l_t$ is $d_t = \alpha l_t = \alpha t v$, and is a rough approximation of the distance between two neighboring particles, see figure 4.1. Solving for $t$ with $d_t = \frac{r}{2}$ yields the time of the subdivision event:

$$\frac{r}{2} = \alpha\, t\, v \;\Leftrightarrow\; t = \frac{r}{2\, v\alpha}\,.$$

From the acquired time the subdivision event can be scheduled, removing the need for finding candidates each update. The subdivision means three particles will spawn, one at the original particle's position and two on each side of that. The dispersion angle and amplitude of the new particles are one third of the original, and the direction of the new outer particles is rotated by $\frac{\alpha}{3}$ and $-\frac{\alpha}{3}$ respectively.

### 4.2.2 Particles as a Heightfield

Having subdivided particles accordingly, the particle system needs to be turned into the heightfield representation discussed earlier. This can be done by properly selecting a waveform function $W_i(\boldsymbol{x})$, yielding the heightfield function

$$h(\boldsymbol{p}, t) = y_0 + \sum_i a_i W_i(\boldsymbol{p} - \boldsymbol{p}_i(t))\,,$$

where $a_i$ is the amplitude and $p_i(t)$ is the position of wave particle $i$. Yuksel et al. suggests using the following waveform function

$$W_i(\boldsymbol{x}) = \frac{1}{2}\left(\cos\left(\frac{\pi|\boldsymbol{x}|}{r_i}\right) + 1\right)\Pi\left(\frac{|\boldsymbol{x}|}{2r_i}\right),$$

where $\Pi(x)$ is the rectangle function with $\Pi(x) = 1$ for $-0.5 < x < 0.5$, $\Pi(x) = 0.5$ for $x = \pm 0.5$ and 0 otherwise. For simplicity's sake it is usually convenient to keep a constant radius $r_i = r$, which also gives a constant waveform function $W_i(\boldsymbol{p}) = W(\boldsymbol{p})$. With this simplification Yuksel et al. describe how to perform the heightfield conversion efficiently using a convolution on the GPU. Each particle is rendered as a point primitive to a render target, which is later filtered with the waveform function as a kernel. The filtering can be further optimized by using one-dimensional approximations in each direction of the otherwise two-dimensional filter.

To obtain the choppy look of waves, just as with the method described in the previous section, a two-dimensional vector field can be obtained for displacing the horizontal position of the heightfield. This field is given by

$$\boldsymbol{D}(\boldsymbol{p}, t) = \sum_i \boldsymbol{L}_i(\boldsymbol{v}_i) a_i W_i(\boldsymbol{p} - \boldsymbol{p}_i(t)),$$

where

$$\boldsymbol{L}_i(\boldsymbol{u}) = -\sin\left(\frac{\pi|\boldsymbol{u}|}{r_i}\right)\Pi\left(\frac{|\boldsymbol{u}|}{2r_i}\right)\hat{\boldsymbol{u}}$$

### 4.2.3 Water Interaction

To use the described particle system for simulating water interaction, wave particles with proper amplitude and velocity must be spawned accordingly. The authors of the wave particle algorithm describes how to accomplish this by using a GPU-based technique. As the wave entities described in this thesis will be primarily artist generated, these techniques are not in the scope of this thesis.

# Chapter 5

# Implementation Details

This chapter is meant to give a description of the implementations of the algorithms presented earlier. Various aspects of rendering the ocean surface are also discussed. The implementation here refers to a reference implementation made in the Frostbite engine and running on a PC under the Microsoft Windows operating system, and on Sony's Playstation 3 gaming console. The reference implementation was integrated with an existing system for water interaction, but the simulations are not tightly coupled and could be enabled and disabled independently. A simple prototype for quick experimentation with the chosen method was also implemented in Matlab.

## 5.1 Overview

Combining the two different techniques discussed in chapter 4, the simulated wind driven waves and wave entities can be superimposed to form the final heightfield function

$$H(\boldsymbol{p}, t) = y_0 + \sum_k \tilde{h}(\boldsymbol{k}, t)\, e^{i\boldsymbol{p}\cdot\boldsymbol{k}} + \sum_i a_i W_i\left(\boldsymbol{p} - \boldsymbol{p}_i(t)\right) . \tag{5.1}$$

The reference implementation consists of one system for the server and one for the client. The two implementations contain significant differences, but as it turns out much of the code from the server-side can be reused on the client. The only information that the server needs to provide is water heights for different points in time and space, or more precisely, evaluations of the function in equation 5.1.

The client system needs to provide the exact same functionality as the server, but additional work is also needed which is related to rendering. It is assumed that the heightfield queries arrive at a different frequency than the rendering frame rate, and for that reason the simulation data for wind-driven waves used for queries and rendering are computed separately. This means more computations on the client, but has the advantage that the two

sets of simulation data can be processed with different parameters. One example of where this advantage is used in the reference implementation, is that the simulation data used for physics queries does not need surface normals, nor is it equipped with any horizontal displacement data. As the simulation data used for rendering has a horizontal displacement, there will be a discrepancy between the water height returned by the queries and the rendered mesh.

For easy integration with the interactive water system, the LODing of the rendered heightfield use the same quadtree based approach. To be able to use this approach, the heightfield obtained from the simulation will need to be resampled to lower resolutions. In the reference implementation 3–4 LODs were used. As mentioned in section 3.1.2, the grids used by the interactive water simulation have a border region on each side. This border is not used while rendering the heightfield, which means that for a border of size $B$ a mesh representation of such a grid has only $(R-2B+1)\times(R-2B+1)$ vertices. Here the $+1$ comes from the fact that the borders of the mesh must match for tiling purposes. In order for the data from both simulations to match, a scaling of the ocean simulation data is applied for it to fit the unbordered grid area. The scaling uses bilinear downsampling to reach the desired resolution, which is later copied to fill an entire $R \times R$ grid, with maintained border properties.

For the simulation of wind-driven waves, the theory from section 4.1 is used to form the following algorithm:

1. Construct an $R \times R$ field of wave vectors $\boldsymbol{k}$
2. Compute Phillips spectrum $P'_h(\boldsymbol{k})$
3. Compute initial amplitudes $\tilde{h}_0(\boldsymbol{k})$
4. Apply dispersion relationship $\omega(k)$ to get $\tilde{h}(\boldsymbol{k})$
5. Transform to spatial domain to get heightfield $h(\boldsymbol{p}, t)$

Items 1 to 3 is the initialization part of the algorithm which only needs to be executed once any of the parameters change. The subsequent steps are performed on each update step of the simulation.

## 5.2 Simulation Tasks

To fully utilize the multi-core architectures of modern PCs and gaming consoles, the simulation splits its workload among a number of tasks. At every update step of the simulation, the jobs are started to later be synchronized before the rendering is done. The ambient wave simulation has been split into three different tasks: `update`, `precomp` and `prepareLod`, and an additional initialization procedure. The `prepareLod` task is only used by the

client simulation. For updating the wave particle system a subdivision task is also required, this task is however run on the main thread because of the ability to implement it very efficiently.

Running the tasks on a Microsoft Windows PC involves starting a thread which executes the task's code. By relying on the operating system's scheduling the multiple threads can be allowed to run in parallel on a multi-core processor. To avoid the overhead of spawning new threads, a pool of worker threads can be used.

The architecture of Sony's Playstation 3 is slightly different though, and multi-threading will not cause significant performance improvements compared to running all task in sequence on the same thread. The Playstation 3's is equipped with a Cell processor, consisting of one Power Processor Element (PPE) and 8 Synergetic Processor Elements (SPE). The PPE is a 64-bit PowerPC, reduced instruction set computer (RISC), which runs the operating system and controls the SPEs. The SPEs are 128-bit RISC processors designed for smaller data-rich processing tasks and are the main workhorses of the platform. Each SPE executes its instructions on a local memory store, in contrast to the PPE which operates on main memory. To transfer data back and forth between the local store and main memory, the SPEs use direct memory accesses (DMA), which can be performed asynchronously to reduce latency [CBE09].

To utilize the SPEs for the simulation, each task needs to be compiled into a separate small program that are started on each update step. The memory addresses of the data to be processed are passed as parameters to the programs, and by using DMA, the data is read from main memory into the local storage. After the processing is complete, the data is written back.

### 5.2.1 Initialization

The initialization task takes the user-defined parameters and computes an initial field of complex amplitudes and a dispersion field. This task needs only to run once when the simulation starts, and if any of the parameters change. Because of this, the task has not undergone any optimizations in the reference implementation as there are no critical time limits to meet. The reason for also outputting the dispersion field is that, while the expression looks harmless, it does contain two square root operations for each wave vector $\boldsymbol{k}$, if one were to compute it as needed. The set of wave vectors is constant as long as the parameters remain unchanged, which yields the opportunity to pre-compute this data.

For the simulation to be deterministic, the initialized data needs to be precisely equal on all involved clients and on the server. The computation of the initial heightfield draws from a random number generator which, for the equality to be fulfilled, must be identical. In this implementation a seeded pseudo random number generator is used, with the seed being part of the

common simulation parameters that are of course identical on all clients.

### 5.2.2 Updating the Heightfield

The update task contains two distinct computational sub-tasks, where the second depends on the results of the first. The first task consists of applying the dispersion to the initial grid of complex amplitudes $\tilde{h}_0$. The dispersion is done according to equation 4.4 and the dispersion relationship $\omega(k)$. Because equation 4.4 maintains the complex conjugation property, the bottom half of the resulting set of amplitudes $\tilde{h}(\boldsymbol{k}, t)$ will be the complex conjugate of the horizontally and vertically flipped upper half. By exploiting this fact, the number of needed iterations for this procedure is $R^2/2$.

To guarantee identical results between clients, the clocks used for simulation must be synchronized between server and clients. This is handled by the underlying networking system of the engine and not part of this implementation. It works by having each client separately maintain its own timer, which are continuosly corrected by the server, should it start to drift. A time correction from the server may cause unexpected jumps in the simulation time for a client and thereby a jump in the heightfield animation. To avoid such jumps, the clock is not instantly skipped to the correct time when a correction arrives, but instead accelerated by a factor proportional to the time error.

The second sub-task is to transform the results from the previous task to the spatial domain. This is done using an FFT algorithm, discussed further in section 5.5. In total, 5 inverse FFTs are performed where one produces the heightfield, two are used for horizontal displacement, and the last two are for the computation of surface normals.

### 5.2.3 Precomputation of Heightfields for Point Sampling

To accommodate for the point sample queries of the water surface height, this task precomputes heightfields and stores them for later use. This performs a subset of the operations in the `update` task, where only the heightfield is computed with no additional horizontal displacement or normal data. This means only one inverse FFT operation is executed for each instance of the `precomp` task. The buffering scheme is described in more detail in section 5.3.

### 5.2.4 Level of Detail Preparation

The `prepareLod` task is a preparation of the data before rendering the heightfield. The task uses the data which is output from the previously described `update` task to prepare a single LOD of lower resolution. This task performs the down-sampling described in the overview section above to

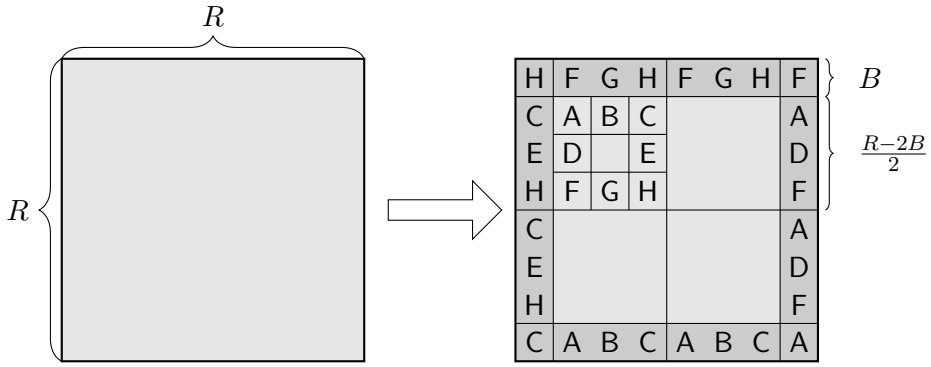



Figure 5.1: Downsampling and border copy of original heightfield of resolution $R$ to level of detail $l = 1$. The original heightfield is down-sampled and stored in the upper left region of the target heightfield. Next, it is copied to the remaining three regions and the lettered sub-regions are copied to the corresponding area of the border

reach the desired resolution

$$R_{target} = \frac{R - 2B}{l + 1},$$

where $l$ is the requested LOD and $l = 0$ is the highest resolution LOD. After obtaining the low-resolution heightifield it is copied to fill the topmost row of the $R \times R$ tile. Next, that row is copied to fill the rest of the tile. While doing so, the borders are also copied to preserve the same data layout as used by the interactive simulation. This procedure is depicted for $l = 1$ in figure 5.1. The LOD preparation tasks can run in parallel with one task instance per LOD, as there are no dependencies between them.

## 5.3 Heightfield Sampling

It is assumed that for a point in time $t_0$, the water height may be queried for a range of times $t_0 - S \leq t < t_0 + S$, where the times are discrete, uniformly spaced time steps. To quickly be able to respond to such queries, the system maintains a buffer of heightfields from past and for future time steps instead of computing them on-demand. The maximum time $t_{max}$ for which a query has been answered is recorded, and with a buffer of size $2S$, the buffer ensures that the heightfields for times $t_{max} - S \leq t_{max} < t_{max} + S$ are in the buffer. If the initial assumption holds, a sufficiently large buffer will guarantee that all the heightfields needed to respond to the queries are already in the buffer. However, to be safe, a fallback on-demand computation of a heightfield can be made if needed. At every update step of the simulation, $t_{max} - t_{buf}$ new

heightfields are computed and buffered, where $t_{buf}$ is the maximum time of all currently buffered heightfields.

Simply fetching a buffered heightfield is not enough to respond to a query. As a query is made for an arbitrary point $\boldsymbol{p}$ in space, the heightfield fetched from the buffer needs to be sampled in some fashion. First of all, if the point is outside the bounds of the entire water surface, no further work is needed and the sampling procedure is complete. If it is inside, the offset position into the water surface is determined with $\boldsymbol{p}_s = \boldsymbol{p} - \boldsymbol{p}_0$, where $\boldsymbol{p}_0$ is the corner position of the entire water surface. This position is then used to compute the position in the tile's coordinate system $\boldsymbol{q} = \frac{R}{D}\boldsymbol{p}_s$. The final water height is finally bilinearly interpolated from four samples from the discrete grid points closest to $\boldsymbol{q}$.

Superimposing individual waves is performed by simply iterating over all active wave particles and summing evaluations of their waveform functions.

## 5.4 Simulation Parameters

There are several parameters that affect the outcome of the ocean simulation. Some parameters influence the general appearance of the surface waves while others can be used to adjust fidelity. Increasing fidelity, as usual, also means increasing computational cost and thereby decreasing performance.

The first two parameters to set are the simulation resolution and dimension. The resolution of course has a direct relation to the computational cost. The implication of choosing a higher simulation resolution is that more wave vectors will fit into the grid. The increased number of wave vectors will expand the heightfield outwards, and thus give waves with higher wave numbers. Recalling that the wave number is inversely proportional to the wavelength, this will give more small surface waves. A resolution above around $2048 \times 2048$ will give such small wavelengths that numerical precision becomes an issue [Tes04b]. During the work of this thesis the resolutions were usually set to $64 \times 64$ or $32 \times 32$. The dimension parameter only controls the size in the world that a simulation tile covers and thus does not add any additional computational costs for the simulation.

The modified Phillips spectrum in equation 4.3 provides an additional four parameters that can be set by the user. Upwind and downwind power, controls how much the propagation directions of waves will be spread out around the wind direction. Upwind and downwind scaling factors controls the amplitude of waves travelling in the corresponding direction.

## 5.5 The Fast Fourier Transform

Evaluating a DFT by definition, while maintaining interactive frame rates, is infeasible as the complexity of such an operation is $\mathcal{O}(N^2)$ in the one-

dimensional case and at best $\mathcal{O}(N^3)$ in two dimensions. Instead we resort to an FFT algorithm, such as the traditional one, proposed by Cooley-Tukey. This effectively reduces the complexity of the problem to a more manageable $\mathcal{O}(N \log N)$ in one-dimension and thus $\mathcal{O}(N^2 \log N)$ in the two-dimensional case [FJ05]. For the first prototype implementation a simple, yet naïve, implementation of the Cooley-Tukey algorithm was used to verify the results. However, this unoptimized implementation still did not provide sufficient performance, especially on consoles, which led to the use of a third-party library specialized for FFT computations.

### 5.5.1 Using FFTW

The problem faced in this thesis is to efficiently compute a transform of fixed and moderate size. For this reason it would be possible to write low-level optimized code that is tailored for that particular problem. This solution however, is not very flexible. As mentioned, it is desirable to be able to adjust simulation fidelity depending on platform, which means lots of work needs to be put into writing similar low-level optimized code for those problems as well.

For these reasons a more flexible solution, to use the third-party library FFTW, was chosen. FFTW stands for *The Fastest Fourier Transform in the West* and is a collection of C routines released both under the GNU GPL license and other non-free proprietary licenses. FFTW works with a set of *codelets*, which are machine generated code sequences, specialized on performing one particular set of computations. By testing different combinations of the codelets, FFTW can find an optimal solution for the given problem. The speed of computations is comparable with "hand-optimized solutions" [FJ05]. With the FFTW software package, it is also possible to generate new codelets if needed.

## 5.6 Wave Particles

### 5.6.1 Particle Data

To avoid runtime allocation of memory on the heap, it is necessary to maintain a buffer of constant size for the particle data. Each particle carries a property which tells if it is active and should be included in the simulation. To avoid searching the buffer for available particles when spawning new ones a simple wrapping counter tells the next index to choose. This will cause particles, which are potentially still alive, to be overwritten. However, the buffer can be maintained to keep the particles roughly sorted by creation time, meaning the overwrites will only hit old particles.

The data needed for each particle is: creation time, initial position, direction, amplitude and dispersion angle. A choice can be made whether

to allow varying particle radii or use a constant global value. Allowing individual radius properties, in combination with waveform evaluation using a convolution kernel is problematic. This would need numerous convolutions with different kernels and would significantly increase memory usage. The reference implementation does not use convolution, which allows the use of individually set radii.

### 5.6.2 Subdivision

The subdivision time table of the wave particle system can be implemented efficiently by choosing the right data structures [Yuk10, p. 90-92]. In the reference implementation, a map data structure was used to map the time of a subdivision event to a list of particles. To avoid dynamically allocating memory for the particle lists, a linked list structure was used. By equipping the particle data with a pointer to the next particle, this scheme is readily implemented. Thus, the time table is a map from discrete time step to a wave particle pointer, pointing at the tail of the subdivision list.

Ideally, all particles subject for subdivision at a point in time should be ordered sequentially in memory to improve memory access times. Subdividing a particle, involves creating two new particles on either side of the original one, which is not optimal for memory layout. To overcome this, the original particle is relocated to where the two new particles are created in the memory buffer. This also reduces the risk of overwriting particles before it has been subdivided [Yuk10, p. 94].

## 5.7 Rendering

The heightfields obtained from the simulation of wind-driven waves are well suited for the already present rendering. As described in section 3.1.1, the quadtree structure will provide a number of grids of equal resolution, but different dimension. To apply the height field, the different LODs of the simulation data can be matched with a quadtree level, giving a range of levels of where to superimpose the heightfield. The level matching can be performed by finding the quadtree level which has grids of a dimension which most closely matches the dimension of the simulation data. An example quadtree hierarchy is depicted in figure 5.2. To apply the height field to grids below the range of levels, the present system's upsampling algorithm is used to bicubically upsample the data from the level above.

To render the wave particles, a naïve solution would be to for every vertex in every mesh grid evaluate the waveform function for every wave particle. This solution was deemed too expensive, which lead to a more efficient, yet still naïve, two-pass solution. By first determining which particles affect which grid, the complexity is reduced. Considering a wave particle's

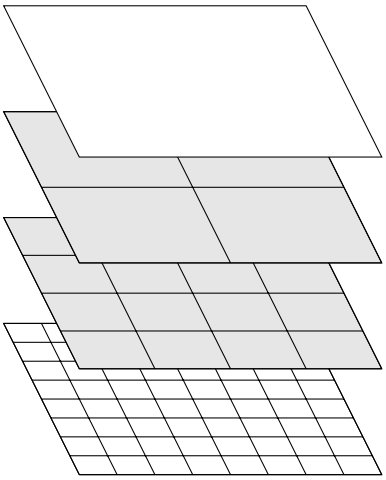


Figure 5.2: Example quadtree hierarchy for a water surface. Using 2 LODs, the simluation data for wind driven waves would be inserted at the levels shaded with gray color, with the highest LOD in the grids of the bottommost of these levels. For quadtree levels below the LOD range, the data from above levels are upsampled to fit the grids.

position and the inflated two-dimensional bounding box of a grid, a point-box intersection test can tell which particles affect a given grid. The bounds of a grid must be inflated by the constant particle radius $r$ in each direction to successfully cover the influence of all particles, which is exemplified in figure 5.3. All particles affecting a grid is appended to a list, which in the second pass provides each grid with the particles to consider. The second pass is also easily parallelized, for further performance increase.

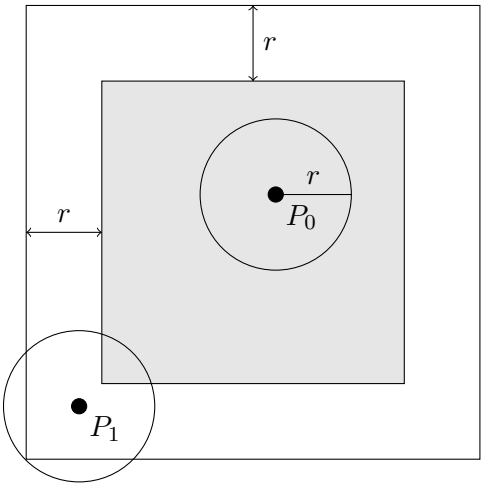



Figure 5.3: Example grid sampling of a wave particle system with two particles: $P_0$ and $P_1$, and the grid shaded in gray. The outer square represents the inflated bounding box which is needed to catch influences from all relevant particles.

# Chapter 6

# Results

This chapter presents the results of the implementation of the algorithms. The default set of parameters used, unless otherwise stated, is shown in table 6.1.

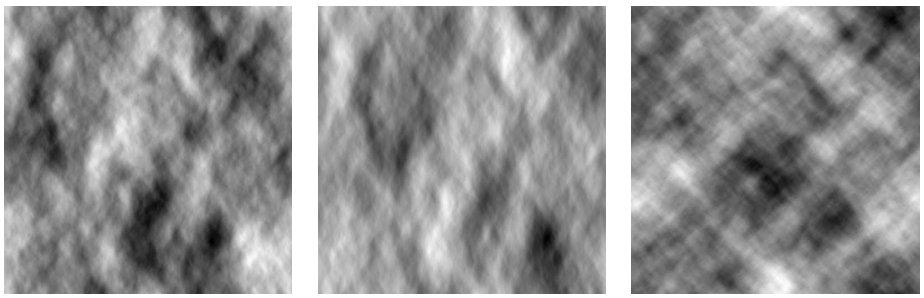
| | |
|---|---|
| Resolution | $R = 512$ |
| Dimension | $D = 1500\,\mathrm{m}$ |
| Wind direction | $\hat{\boldsymbol{w}} = (w_x, w_z) = (1, 0)$ |
| Wind speed | $\lvert\boldsymbol{w}\rvert = 30\,\mathrm{m/s}$ |
| Directionality | $p_- = p_+ = 2$ |
| Scale factor | $d_- = d_+ = 1$ |

Table 6.1: Default simulation parameters used for the results presented in chapter 6.

## 6.1 Renderings

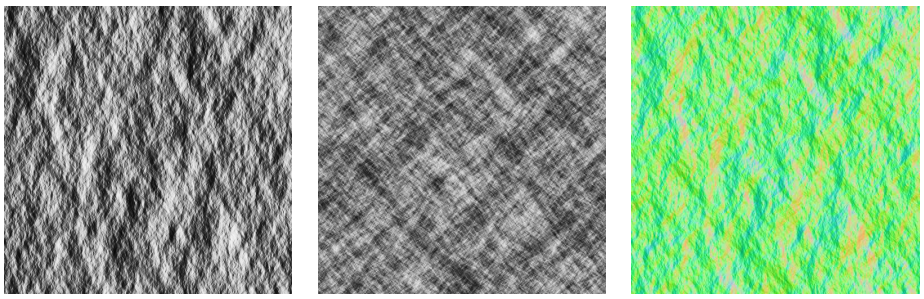
Using a 2-dimensional grayscale image, the heightfield and horizontal displacement maps can be represented. Such a representation is shown in figure 6.1. Observing only this representation, it is not clear in which direction the waves are propagating. But, by also taking the derivatives into account, the direction becomes more evident. Similar renderings of the derivatives in the $x$ and $z$ directions, along with the resulting normal map are shown in figure 6.2.

The final result can be seen in figure 6.3, where an entire water body is rendered with basic shading. Looking closely at this figure, cracks can be noticed between different LODs as the border vertices of the mesh patches does not match. A repeating pattern is also evident caused by the tiled heightfield with a rather low resolution. The rendering in figure 6.4 shows the wireframe of the same water body.



(a) (b) (c)

Figure 6.1: Example two-dimensional representations of (a) height field, (b) horizontal displacement in $x$ direction, and (c) horizontal displacement in $z$ direction. The parameters used are those presented in table 6.1.



(a) (b) (c)

Figure 6.2: Heightfield derivatives rendered as a two-dimensional map. (a) is the $x$ derivative, and (b) the $z$ derivative. In (c) the resulting normal map is rendered in color with the red, green and blue channels representing $x$, $y$ and $z$ components respectively. Simulation parameters are listed in table 6.1.

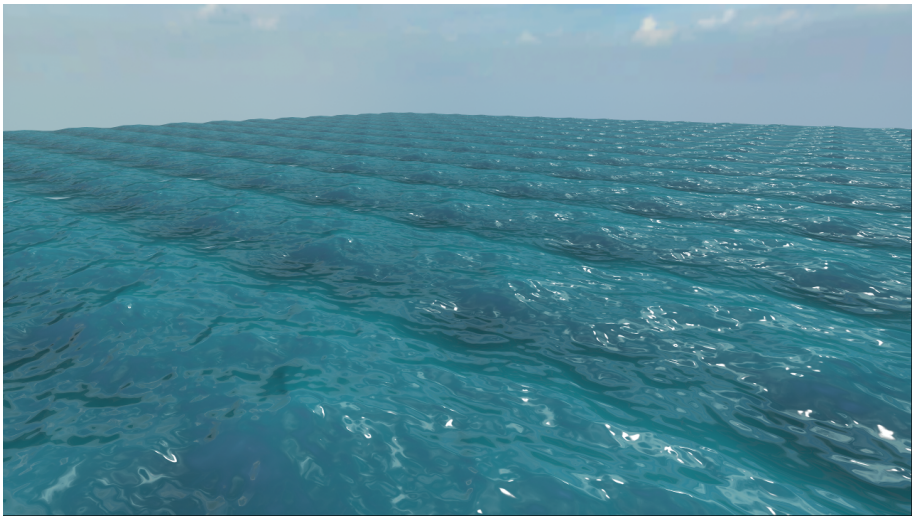


Figure 6.3: Rendering of ocean surface with basic shading. The heightfield resolution used was $64 \times 64$ which explains the obvious repeating pattern caused by tiling. Cracks between different LODs are also evident on closer examination.

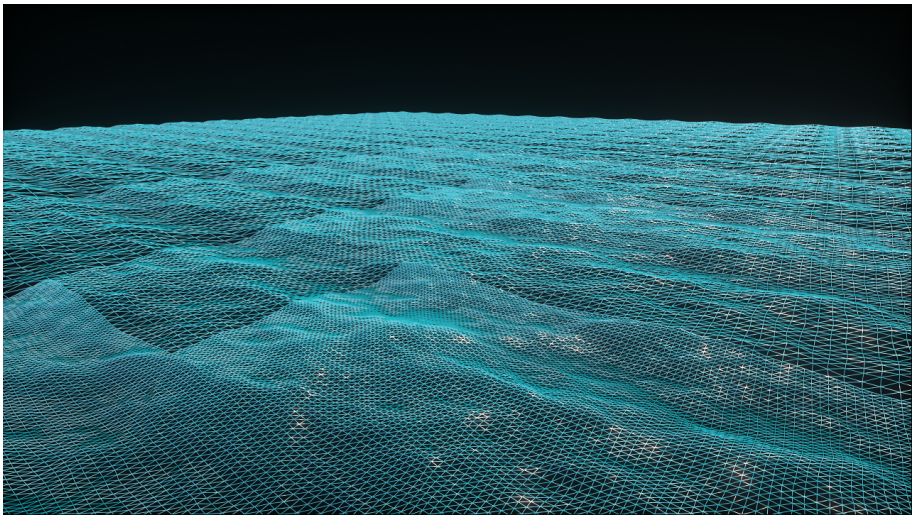


Figure 6.4: Wireframe rendering of the same water body as in figure 6.3. As shown in the rendering, 4 LODs are used.

| | PC | | PS3 | |
|---|---|---|---|---|
| Task | $32 \times 32$ | $64 \times 64$ | $32 \times 32$ | $64 \times 64$ |
| `update` | 0.08 | 0.29 | 0.12 | 0.33 |
| Propagation | 0.06 | 0.21 | 0.08 | 0.24 |
| Transform | 0.02 | 0.08 | 0.04 | 0.09 |
| `prepareLod` | 0.05 | 0.06 | 0.06 | 0.07 |
| `precomp` | 0.05 | 0.15 | 0.06 | 0.16 |
| Propagation | 0.04 | 0.12 | 0.04 | 0.12 |
| Transform | 0.01 | 0.03 | 0.02 | 0.04 |
| Frame time ($\frac{1}{f}$) | 7.69 | 10.0 | 33.3 | 33.3 |
| Simulation CPU Time | 0.29 | 0.58 | 0.42 | 0.77 |
| Wall clock time | 0.13 | 0.35 | 0.18 | 0.40 |

Table 6.2: Timing measurements (in milliseconds) for each of the simulation tasks and sub-tasks. On Playstation 3 (PS3), all tasks are executed on the synergetic processing elements (SPE). The query frequency is $f_q = 30\,\text{Hz}$ and LOD count $l = 4$. Simulation CPU Time is computed by the formula in 6.1, and wall clock time according to 6.2.

## 6.2 Performance

Apart from fulfilling the requirements of deterministically generating a plausible mesh representation of an ocean surface, a key factor of a successful implementation of this method is performance. In table 6.2, processor time for one instance of each of the simulation tasks and sub-tasks is presented for different resolutions, and on two different platforms: a Sony Playstation 3 and a PC. The PC was equipped with an Intel Xeon X5550 with 4 cores at 2.66 GHz and 12 GB of RAM. The number of LODs used during measurements was $l = 4$, and the used query frequency was $f_q = 30\,\text{Hz}$.

On the client, the number of task instances of each task that is run every frame is variable. This is due to the possibility of the frame rate $f$ being different from the frequency of the heightfield queries $f_q$. This means that for each frame, precisely one instance of `update` task needs to be executed, together with one instance of `prepareLod` per LOD. The most common situation in the reference implementation is that $f \geq f_q$, which means that one instance of the `precomp` task executes roughly every $\frac{f}{f_q}$ frame. The total CPU time spent can thus be computed with the formula

$$t_{cpu} = t_{\texttt{update}} + l \cdot t_{\texttt{resmaple}} + \frac{f}{f_q} t_{\texttt{precomp}}\,, \tag{6.1}$$

where $l$ is the LOD count and $t_{\texttt{task}}$ is the CPU time for `task`.

The wall clock time that is needed for the entire simulation can be com-

puted with the following formula:

$$t_{wall} = \max(t_{\texttt{precomp}}, t_{\texttt{update}} + t_{\texttt{prepareLod}}) \,. \tag{6.2}$$

This means that the wall clock time is whichever finishes last of one instance of the `precomp` task and one instance of `update` together with one instance of `prepareLod`. The assumption is made that the tasks are allowed to run in parallel, meaning that `precomp` and `update` can be started at the same time. Also, all `prepareLod` tasks can be executed in parallel, yielding the above expression. Note that the `prepareLod` task can vary slightly in run time depending on which LOD it is computing. These differences were however considered negligible in the performed measurements.

On the server-side the total CPU time and wall clock time are $t_{cpu} = t_{wall} = t_{\texttt{precomp}}$, since only one instance of the `precomp` task is executed every update step.

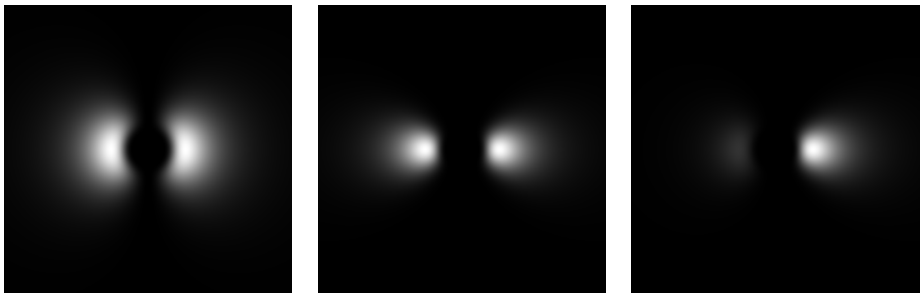
### 6.2.1 Performance Improvements from Using FFTW

Replacing the naïvely implemented FFT algorithm significantly improved performance on all tested platforms. For a simulation resolution of $64 \times 64$, the measured CPU time on PC for the naïve algorithm was 0.55 ms per transform. With FFTW, the corresponding time was improved with one order of magnitude to 0.05 ms. Similar improvements were also evident on the other platforms.

It should be noted, that while there is a gain in performance, static linking with the FFTW library will lead to increased code size of the executable. This could potentially be alleviated by extracting or generating the needed FFTW codelets, and thus only link with those. While sacrificing flexibility, this solution could be used to produce a small amount of high-performance code that is tailored for the given problem and platform. This could potentially also improve performance even further, making it a viable option for final builds of an application as the flexibility offered can be discarded at that point.

## 6.3 Modified Phillips Spectrum

The alternative Phillips Spectrum in equation 4.3 was implemented to increase the user's level of control over the simulation. In figure 6.5 three example spectra are represented as two-dimensional grayscale images. With white color representing high amplitudes, the figure 6.5 (a) show that the maximum amplitudes in the spectrum are in the upwind, and also in the downwind direction for the default spectrum. The spectrum is tweaked with increased directionality in figure 6.5 (b), and with downwind amplitudes scaled down in figure 6.5 (c).



(a) (b) (c)

Figure 6.5: Three different spectra generated with the modified Phillips spectrum with resolution, dimension and wind direction as specified in table 6.1, but with wind speed 5 m/s. White color represents high amplitudes. (a) Default parameters, $s_- = s_+ = 1$ and $p_- = p_+ = 2$. (b) Increased power, $p_- = p_+ = 8$. (c) Scaled amplitudes in downwind direction $p_- = 3$, $p_+ = 8$ and $s_- = 0.2, s_+ = 1$.

## 6.4 Wave Entities

As the reference implementation of a wave particle does not strive for high performance, no detailed performance measurements were conducted on this algorithm. For completeness sake, with a maximum particle count of 4000, a parallelized waveform evaluation step, performs in roughly 5 – 6 ms wall clock time on modern PC hardware. Other steps of the algorithm can be implemented very cheaply and does not add significant cost.

# Chapter 7

# Discussion

## 7.1 Ocean Wave Simulation Using FFT

The presented solution provides a deterministic ocean wave simulation, which is feasible for integration into a real-time application. The simple, yet effective, heightfield representation is well suited for this application as both rendering work and individual point queries can be performed efficiently.

### 7.1.1 Performance

The measurements presented in chapter 6 show promising results in terms of performance. The performance level obtained indicates that a simulation of higher resolution is possible, without significant reduction in overall frame rate. Higher resolutions were not experimented with during the work of this thesis, mainly because of limitations imposed by the underlying interactive water simulation.

The re-sampling and LOD preparation task and the wave propagation task have not undergone significant optimization in the reference implementation. Both these tasks are candidates for vectorization and to thereby put the platforms' single-instruction, multiple data (SIMD) capabilities to use.

### 7.1.2 Spectrum Tweaks

The modified Phillips spectrum gives the opportunity for artists to better control the simulation so that the desired appearance can be achieved. Modifying the spectrum does diverge the method from its foundation in studies of actual ocean surface, but in this application the ability to control the simulation was deemed more important than maintaining the highest possible level of realism.

A problem with the chosen method, is that directly modifying the spectrum can be difficult for someone not well versed in the used method. The implications of a tweak applied by a user can have consequences on the end

result that are difficult to predict, making it a time consuming process to find the right parameters [GOH12]. However, the modified Phillips spectrum is intended to alleviate the customization process.

### 7.1.3 Parameter Changes

The proposed method is not well suited for changes in the parameters. Parameters that are candidates for runtime changes are the wind speed and direction, whereas the resolution and dimension parameters should be kept constant. A change in wind direction or speed would not be instant, but rather a transition over a period of time. A transition in parameter values means that the heightfield needs to be reinitialized for numerous different sets of parameters, which is an expensive effort and could potentially cause jumps in the animation of the heightfield. A better solution would be to, as the parameter changes, initialize a new temporary simulation with the target parameters and blend it in linearly over time. This would cause a smooth transition, with only one needed reinitialization but with twice the required computational effort during the transition.

### 7.1.4 Shores

As the simulation is meant for the open ocean, practical handling of shorelines is an issue that needs to be addressed. In particular, when a rough ocean surface is simulated, a gently sloping shore will cause very unrealistic behavior. If the simulation has the ability to access the geometry that represents the seabed, a simple solution would be to locally apply a water depth dependent scale factor to the heightfield. Experiments have shown that by blending with another procedural wave generator, which is specialized on shore waves, a more realistic shoreline can be obtained.

Another solution, which does not require access to the seabed geometry, is to use an artist painted mask that encodes the scaling factor applied to the heightfield. The mask could potentially encode other data, such as a flow map, providing further simulation control to the user. With flow data, the simulation could also be used to represent larger rivers and the transition between water bodies. Flow maps could also be generated from the terrain geometry, but as this is likely a costly procedure this data generation would need to be done in a pre-processing step.

### 7.1.5 Tiling Artefacts and Cracks

A tile which is much smaller than the total area of viewed water surface can make tiling artefacts evident. Of course, by increasing the tile dimension, this problem could be alleviated. This would however lead to the need for higher simulation resolution in order to maintain the detail provided by the high-frequency waves.

To keep computational cost down, a potential solution would be to perform two heightfield computations of the same resolution but at different dimension. The heightfield of larger dimension could be linearly up-sampled to form a higher resolution heightfield. Several of the smaller heightfields could then be superimposed on the larger to form the final result. The larger heightfield would break up the tiling artefacts created by the smaller. This technique could of course be further extended to use as many heightfields as desired. This would however burden users with more parameters to adjust, to obtain the right appearance.

As is evident from the rendered ocean surface in figures 6.3 and 6.4, vertices along borders between different LODs does not line up correctly, causing cracks in the resulting mesh. A solution to this is to apply some form of stitching along the edges, which is a common method of dealing with such problems.

### 7.1.6 Heightfield Sampling Discrepancy

As mentioned in the implementation chapter, the heightfield queries will not provide query responses that exactly match the rendered mesh. By equipping the buffered heightfields with horizontal displacement information, the queries could be made more exact. However, this would not only mean more complicated sampling of the heightfield, but also more memory usage by the buffer.

In the reference implementation, this discrepancy has not been found to cause problems in terms of simulation accuracy. As buoyant objects of significant density do not float directly on the water surface, but are submerged to some extent, the discrepancy is hidden. The accuracy may become problematic for smaller, light objects, which follow the water surface more precisely.

As the heightfields used for queries are separate from the rendered heightfield, it is also possible to use different resolutions for the two. If a higher resolution is desired for rendering purposes, a low resolution simulation could potentially still be used for queries while maintaining an accurate physics simulation. This is of course interesting for performance reasons, both in terms of computational effort and memory usage.

The rendered mesh will represent the water surface at the local client time, meaning there will be a discrepancy between objects not controlled locally and the water surface. As with the sampling discrepancies described above the buoyancy physics hides this to some extent, avoiding the floating object looking detached from the water surface. This is at least true for moderate network latency. Should high latencies cause unacceptable errors, solutions such as local modification of the water surface around objects could be employed. If the latency can be estimated, two heightfields from different points in time can be radially blended with respect to the local clients camera

position. This means the water surface from the actual time is shown in the distance, and the local time representation is used near the client.

## 7.2 Wave Entities

The implementation of wave particles is only a subset of the entire algorithm as described by the original authors. For example, no reflection step is included in the reference implementation, as it would need additional data from the geometry surrounding the water body not readily accessible.

The described way of handling subdivision events maintains the determinism of the system. Because future positions and subdivisions of any wave particle can be determined another conclusion to draw is that, from a multi-client perspective, only the initial wave particles need to be synchronized over the network. A potential problem worth noting comes from applying the previously described client-side prediction scheme. If the heightfield is evaluated for a future time, subdivision events occurring between the current time and this future time will not have been executed. A potential solution could be to temporarily fast forward the subdivision events to the desired point in time, ensuring an accurate evaluation of the water level. However, in the reference implementation this was not deemed necessary as the error introduced by evaluating a non-subdivided particle system was not noticeable.

### 7.2.1 Other Uses

Using wave particles has proven a flexible solution that could potentially see many applications. A basic wave generator can be implemented to provide large wave trains, with wavelengths far larger than what the ambient simulation can handle. This not only adds more waves to the ocean surface, but could also be used to break up tiling artefacts caused by a low resolution ambient simulation.

Attaching wave particle generators to objects interacting with the water surface could also be an interesting use, giving the ability to simulate the presence of boat wakes and similar effects. Care must be taken in a multi-client environment, as initial positions of wave particles need to be synchronized in order to provide equal simulation results, which leads to increased network bandwidth usage. As objects in the world are already synchronized, the position of these could be used for determining spawn locations for particles. This would however require a deterministic way of finding when such particle spawn events occur, that is insensitive to network latency.

## 7.3 Conclusion

This thesis has presented a system for ocean simulation in real-time applications. The main focus has been to obtain realistic results while maintaining high performance and providing a decent level of user control. The system has been divided into two systems: one for simulation of wind-driven waves and the other for manually controlled individual waves on the ocean surface.

The wind-driven waves use a simulation method presented by Jerry Tessendorf, which is based on statistical observation of ocean surfaces. The method provides, if not the most, at least a very realistic simulation of open ocean surface waves. To improve user control, the wave distribution spectrum used in the method was altered to expose more parameters which affect the outcome of the simulation. In this thesis, the integration of this method into a quadtree-based LOD scheme is also discussed.

Individual waves on the water surface was implemented as *wave entities*, with an algorithm based on Cem Yuksel's Wave Particles. The proposed system lends the basic ideas from the original algorithm: representing waves with a particle system, particle subdivision, and how to obtain the resulting heightfield. Particle generation in the presented system is performed manually by a user, as opposed to the original algorithm which approximate water-object interactions to simulate the wave generation.

To address the steep performance requirements imposed by implementing the system in a game engine, the paralell processing capabilities of the target platform was exploited. To further improve performance, the FFTW library provided a fast implementation of an FFT algorithm that could readily be applied to the simulation data.

# Bibliography


[CBE09] *Cell Broadband Engine Programming Handbook. Including the PowerXCell 8i Processor*. Version 1.12. Apr. 3, 2009. URL: https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/7A77CCDF14FE70D5852575CA0074E8ED/$file/CellBE_Handbook_v1.12_3Apr09_pub.pdf (visited on 05/17/2012).

[CM10] Nuttapong Chentanez and Matthias Müller. "Real-time simulation of large bodies of water with small scale details". In: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '10. Eurographics Association, 2010, pp. 197–206. URL: http://dl.acm.org/citation.cfm?id=1921427.1921457.

[CS09] Hilko Cords and Oliver Staadt. "Real-Time Open Water Environments with Interacting Objects". In: *Proceedings of Eurographics Workshop on Natural Phenomena (EGWNP '09)*. 2009, pp. 35–42.

[FJ05] M. Frigo and S.G. Johnson. "The Design and Implementation of FFTW3". In: *Proceedings of the IEEE* 93.2 (Feb. 2005), pp. 216–231. ISSN: 0018-9219. DOI: 10.1109/JPROC.2004.840301.

[FR86] Alain Fournier and William T. Reeves. "A simple model of ocean waves". In: *SIGGRAPH Comput. Graph.* 20 (4 1986), pp. 75–84. ISSN: 0097-8930.

[Gei+10] Robert Geist et al. "Lattice-boltzmann water waves". In: *Proceedings of the 6th international conference on Advances in visual computing - Volume Part I*. ISVC'10. Springer-Verlag, 2010, pp. 74–85. ISBN: 3-642-17288-1, 978-3-642-17288-5.

[GOH12] Carlos Gonzales Ochoa and Doug Holder. *Water Technology of Uncharted*. Game Developers Conference. 2012. URL: http://www.gdcvault.com/play/1015309/Water-Technology-of (visited on 05/16/2012).

[Joh04] Claes Johanson. "Real-time water rendering". Master's thesis. LTH Lund Institute of Technology, Mar. 2004.

[Len12] Joel Lennartsson. "Data Oriented Interactive Water". Bachelor's thesis. Linköping University, 2012.

[Mit07] Martin Mittring. "Finding next gen: CryEngine 2". In: *ACM SIGGRAPH 2007 courses*. SIGGRAPH '07. ACM, 2007, pp. 97–121. DOI: 10.1145/1281500.1281671.

[MWM87] Gary A. Mastin, Peter A. Watterberg, and John F. Mareda. "Fourier Synthesis of Ocean Scenes". In: *Computer Graphics and Applications, IEEE* 7.3 (Mar. 1987), pp. 16–23. ISSN: 0272-1716. DOI: 10.1109/MCG.1987.276961.

[Ott10] Björn Ottosson. "Real-time Interactive Water Waves". Master's thesis. KTH Royal Institute of Technology, 2010. URL: http://publications.dice.se/attachments/water%20interaction%20ottosson_bjorn.pdf.

[Tes04a] Jerry Tessendorf. "Interactive Water Surfaces". In: *Game Programming Gems 4*. Charles River Media, 2004, pp. 265–275. URL: http://people.clemson.edu/~jtessen/papers_files/Interactive_Water_Surfaces.pdf (visited on 02/08/2012).

[Tes04b] Jerry Tessendorf. *Simulating ocean water. SIGGRAPH 2004 Course Notes*. 2004. URL: http://people.clemson.edu/~jtessen/papers_files/coursenotes2004.pdf (visited on 05/16/2012).

[Th07] Nils Thürey et al. "Real-time Breaking Waves for Shallow Water Simulations". In: *Proceedings of the Pacific Conference on Computer Graphics and Applications 2007* (Oct. 2007), p. 8.

[Yan+05] Xudong Yang et al. "GPU-based real-time simulation and rendering of unbounded ocean surface". In: *Computer Aided Design and Computer Graphics, Ninth International Conference on*. Dec. 2005, 6 pp. DOI: 10.1109/CAD-CG.2005.45.

[YHK07] Cem Yuksel, Donald H. House, and John Keyser. "Wave Particles". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)* 26.3 (2007), p. 99.

[Yuk10] Cem Yuksel. "Real-time Water Waves with Wave Particles". PhD thesis. Texas A&M University, 2010.