Real-Time Water Animation and Rendering using Wavefront Parameter Interpolation

Master's thesis in Complex Adaptive Systems

Gustav Olsson

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Master's thesis 2017

Real-Time Water Animation and Rendering using Wavefront Parameter Interpolation

Gustav Olsson

Department of Computer Science and Engineering
Computer Graphics Research Group
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Real-Time Water Animation and Rendering using Wavefront Parameter Interpolation
Gustav Olsson

Supervisor: Erik Sintorn, Department of Computer Science and Engineering
Advisors: Fredrik Larsson and Jan Schmid, DICE
Examiner: Ulf Assarsson, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Computer Graphics Research Group
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Water waves refract around a natural jetty in a virtual depiction of the Mediterranean coast implemented in the Frostbite [2017] game engine. The water is simulated and rendered using the presented algorithm.

Typeset in LaTeX
Gothenburg, Sweden 2017

Real-Time Water Animation and Rendering using Wavefront Parameter Interpolation
Gustav Olsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Realistic simulation and rendering of water is a challenge within the field of computer graphics because of its inherent multi-scale nature. When observing a large body of water such as the sea, there are small waves and perturbations visible close to the observer. As the distance increases, the small scale details form large scale wave patterns that may be several kilometers away.

A common approach to rendering large bodies of water in real-time is to simulate deep water waves in a small area and repeat the wave motions across the water surface at different spatial scales in order to minimize repetition patterns. This method gives excellent results at open sea but cannot react to changes in water depth or the terrain of the virtual scene.

In this thesis, an algorithm for rendering large expanses of water that interact with the terrain of the virtual scene in real-time is presented. The proposed algorithm first simulates water waves in a pre-computation step and saves wavefront parameters on a coarse triangle mesh as proposed by Jeschke and Wojtan [2015]. Then, the stored simulation is evaluated and the water surface is rendered in real-time using a novel staggered update scheme. The staggered update scheme effectively improves the rendering performance by a factor of 8 and makes it possible to render water surfaces of up to 16 square kilometers with excellent visual quality and wave patterns at multiple spatial scales.

Keywords: computer graphics, water, ocean, wave, wavefront, terrain, wavefront parameter interpolation, simulation, rendering, staggering

Contents

1 Introduction

1.1 Background

As computers are increasingly being used to model the world, the importance of computer graphics as a visualization tool increases. Rendering believable virtual scenes is important for any application that depicts the world in some way. In addition to being esthetically pleasing, a visualization provides intuition about the phenomenon under observation.

As water is abundant on Earth, there exists a need to render bodies of water in virtual scenes. For outdoor scenes, one would like to capture the grand scale of water behaviors present in everything from the open ocean and coastal areas to rivers and small lakes. In addition, one would like the transition between these areas to be natural and without seams.

Realistic simulation and rendering of water within the field of computer graphics is a difficult task. For real-time applications such as computer games or visualization software, a balance between realism and performance has to be made where the result is convincing to an observer while at the same time allowing for interactive frame rates (30-60 Hz). Graphical simulations of large expanses of water typically restrict the simulation to 2 dimensions (a top-down view) and model deep water waves on the water surface using *linear wave theory* [Airy, 1841] (also known as *Airy wave theory*). In linear wave theory, a water wave is represented by its *phase*, *amplitude*, *angular frequency* and *wavenumber* and the water surface is assumed to be a sum of waves evaluated as sinusoidal functions.

In order to render an ocean, one simplifies the model by assuming that all waves are deep water waves that travel with constant speed independent of water depth. The Fast Fourier Transform is then used to generate and evolve a height map that offsets the geometry of the water surface to approximate the ocean [Tessendorf, 2001]. To achieve real-time performance, several small height maps are typically tiled across a large expanse of the ocean. The height maps are layered on top of each other at different spatial scales in order to minimize repetition patterns. This approach is excellent at capturing wind-driven deep water waves but is limited when it comes to other types of waves. In particular, the major downside with this approach is that the water simulation can not interact naturally with coastlines or objects that are submerged in the water, nor react to changes in water depth.

In reality, water waves refract when they move from one depth to another in the same way light refracts as it moves from one medium to another. The reason water waves refract is that the speed of any point along a wave depends on the depth of the water at that location. If the speed of points along the wave crest diverge, the wave refracts and the wave crest bends. Thus, the crest curves of water waves typically bend towards areas of shallow waters since that is where the speed is low. In addition to refraction, water waves reflect when they collide with hard surfaces such as cliffs or stones that are submerged in the water. The reflecting waves superimpose on top of the ambient waves and cause ripples on the water surface that are sometimes as important to the appearance of coastal scenes as ambient wind-driven waves.

Thus, the natural next step towards increased realism is to take the terrain of the virtual scene into account when simulating the water.

1.2 Purpose

The aim of this thesis is to develop an algorithm for simulating and rendering large expanses of water that interact with the terrain of the virtual scene. Furthermore, the algorithm must allow rendering of the water surface in real-time on current graphics hardware.

1.3 Problem Statement

The thesis will determine how to:

1. Robustly simulate waves that react to the terrain in a variety of different virtual scenes
2. Render the simulation in real-time on current graphics hardware while maintaining excellent visual quality

Robustly simulate waves that react to the terrain in a variety of different virtual scenes The proposed algorithm will be used in a production environment where there are high demands on the stability and predictability of the simulation. The simulation should not just produce accurate results for one or two example scenes, it must perform reliably across a wide range of possible inputs crafted by artists with little knowledge of the inner workings of the algorithm.

Render the simulation in real-time on current graphics hardware while maintaining excellent visual quality Within the field of computer graphics, performance and visual quality are tightly coupled and the proposed algorithm for rendering the ocean needs to strike a balance between the two. The rendering of the simulation must reach a frame rate of at least 60 Hz in order for the proposed algorithm to be usable in practice. In addition, the visual quality of a rendered frame must be excellent with no visible aliasing or noise.

1.4 Limitations

The scope of the thesis is limited to the animation of waves on the water surface and the basic rendering of the water surface geometry.

While there are numerous aspects that need to be considered when making virtual water believable, the intention is not to write a thorough dissertation on all of them. In order to facilitate pre-computation of the simulation, the terrain of the virtual scene with which the waves interact is considered static and the effects of dynamic objects submerged in the water are omitted. Non-linear effects of water waves such as breaking waves and consequently foam and spray are not considered. Finally, the way light interacts with the volume of water and how it affects the shading of the water surface is not considered.

2 Previous Work

Water surface rendering Bruneton et al. [2010] show how to accurately illuminate and render the ocean at all scales in real-time. Aliasing is eliminated using a hierarchical representation that combines surface geometry, normals and BRDF in order to sample waves at the appropriate level of detail, respecting the Nyquist limit.

There are several ways of constructing the geometry of the water surface. Kryachko [2005] use a simple radial grid mesh centered around the camera and sample the wave function at each vertex. The radial grid is densely sampled at the center and becomes gradually more sparse as the distance from the camera increases. Johanson and Lejdfors [2004] introduce the concept of a projected grid to achieve a uniform resolution of vertices in screen space. A rectangular grid is constructed in screen space and projected from the camera onto the water plane. Each projected vertex is then used to sample the water surface, yielding samples of even spacing in screen space and non-linear spacing in world space.

In order to offset the water surface, several authors turn to trochoid wave profiles [Tessendorf, 2001, Finch, 2004, Bruneton et al., 2010] originally discovered by Gerstner [1809]. A trochoidal wave is an exact solution to the Euler fluid equations for deep water gravity waves [Bruneton et al., 2010] where fluid parcels move in closed circles.

Waves in homogenous media Tessendorf [2001] show how deep water waves can be evolved in the Fourier domain in order to generate realistic height and normal maps of the water surface. With the advent of programmable graphics hardware, their research has become the current state of the art for real-time water animation and rendering of large expanses of water.

Waves in heterogeneous media Waves propagating in heterogeneous media are studied extensively in many areas of research and several simulation schemes for wave propagation have been developed. Rawlinson et al. [2008] provides a detailed review of existing simulation schemes in the context of seismology. For a more general overview, see Runborg [2007].

Fournier and Reeves [1986] and Peachey [1986] were the first to show how wave refraction close to shores can be achieved by varying the phase speed of waves. They use the concept of a "wave train" and a "wave component", respectively, to describe a wave with a particular angular frequency, amplitude and propagation direction. The phase of each point along the wave is numerically integrated along a straight line path in the direction of propagation to produce a grid of phase values. The grids are then used during rendering to evaluate the surface height of the water at any location in the world. These models capture wave refraction but are limited as the propagation direction of a wave is held constant and only first-arrivals are accounted for, i.e., only a single value of the multi-valued phase function is computed.

Ts'o and Barsky [1987] introduce the concept of "wave-tracing" and let the propagation direction of a point on the wave change according to Snell's law. The wave-tracing scheme is analogous to ray-tracing for arrival time computations within the

field of geometrical optics and seismology and conventional ray-tracing for light that is used for image generation in Computer Graphics. For rendering, wave heights are evaluated and stored on a regular grid as rays pass close to the grid points. This method allows for refraction and sharp changes in propagation direction in addition to multiple arrivals of a wave, i.e., many values of the multi-valued phase function are captured. However, in areas where rays diverge, there will be insufficient ray coverage and the method will produce patches of undisturbed water on the surface.

Gonzato and Le Saëc [1997] improve on the work of Ts'o and Barsky [1987] by treating the wave as a wavefront and achieve a nearly constant spatial resolution of rays across the water surface by subdividing and collapsing neighboring rays. This propagation scheme is known as wavefront construction in physical space within the wavefront tracking literature and was originally proposed by Vinje et al. [1993]. Gonzato and Le Saëc [2000] extend their method to capture wave reflection and diffraction and render the water surface by ray-tracing the wavefronts stored at regular time intervals.

In order to reduce the memory requirements of previous methods, Jeschke and Wojtan [2015] propose a two-dimensional unstructured coarse triangle mesh as a means to store the recorded wave data. Reconstruction of the stored wave data is handled through a novel interpolation scheme that sidesteps the Nyquist limit and allows high-frequency waves to be captured even by a loosely tessellated coarse mesh. The coarse mesh is spatially adaptive and constructed by first generating a point-set covering the water domain using Poisson disk sampling with the disk radius varying with the distance to the seabed boundary and then constructing the Delaunay triangulation of these points.

In order to interpolate wave data in wavefront construction schemes, inverse bilinear interpolation must be used. Quilez [2010] describes a simple algorithm for computing the inverse bilinear interpolation of a set of four points.

Osher et al. [2002] introduce the concept of a *bicharacteristic strip* in *reduced phase space* in order to solve for the multi-valued phase function using a level set approach within the field of geometric optics. A self-intersecting wavefront in world space becomes a non-self-intersecting curve in reduced phase space that is known as the bicharacteristic strip. Hauser et al. [2006] compare an Eulerian and a Lagrangian approach to solving the multi-valued phase function and show how the resolution of Lagrangian wavefronts can be maintained in reduced phase space.

Mesh generation Cook [1986] introduced Poisson disk sampling in the field of Computer Graphics and showed how it can be used to reduce aliasing. Bridson [2007] developed a linear-time algorithm for Poisson disk sampling in arbitrary dimensions. Poisson disk sampling can be extended to use variable radii and proofs on the properties of the resulting point-set and subsequent Delaunay triangulation can be made [Mitchell et al., 2012].

The Delaunay triangulation and its dual, the Voronoi diagram, have been studied extensively within the field of Computational Geometry and Computer Graphics and Aurenhammer [1991] provides a thorough dissertation on the subject. Incremental flipping is a common algorithm for constructing the Delaunay triangulation from a set of points that is proven to work in arbitrary dimensions [Edelsbrunner

and Shah, 1992]. Brown [1979] first showed that there is a relationship between n-dimensional Voronoi diagrams and n+1-dimensional convex hulls through a stereographic projection. Edelsbrunner and Seidel [1985] built upon Brown's work and showed that the Delaunay triangulation of an n-dimensional point-set is equivalent to an orthographic projection of the triangles of the convex hull of the points lifted to a n+1-dimensional parabola.

The convex hull is an important concept within the field of Computational Geometry. In addition to being closely related to other important concepts such as the Delaunay triangulation and the Minkowski set, convex hulls can be used to simplify complex problems. Gilbert et al. [1988] showed that convex hulls and the Minkowski set can be used for fast intersection and distance calculations between complex objects. There are numerous algorithms for computing the convex hull of a point-set. Barber et al. [1996] developed an iterative algorithm that works in arbitrary dimensions and has seen widespread adoption but is susceptible to precision errors when the input is not in general position. Gustafsson [2013] proposes a promising topology-based algorithm that is robust against inputs not in general position. This algorithm for generating three-dimensional convex hulls seems to be related to incremental flipping for constructing the two-dimensional Delaunay triangulation.

See Ericson [2004] for a comprehensive summary of geometric concepts in the field of Computer Graphics.

Ocean wave spectra In oceanic research, one is interested in the shape of the sea surface in different weather conditions and several ocean wave spectra have been developed to describe the state of the ocean given different parameters such as wind speed or fetch. Pierson and Moskowitz [1964] experimentally find a spectrum for fully developed seas which gives the energy density of the sea as a function of angular frequency and wind speed.

3 Theory

The next section will introduce the theory necessary to follow the reasoning in subsequent sections.

3.1 Linear Wave Theory

Linear wave theory is a linear model for surface waves on a body of water. It was first formulated by George Bidell Airy in the 19th century [Airy, 1841] and is also commonly known as *Airy wave theory*.

The model states that the water surface can be approximated by a sum of sinusoidal functions:

$$\eta(\vec{x}, t) = \eta_0 + \sum_{i=1}^{N} a_i \sin(\omega_i \phi_i(\vec{x}) - \omega_i t) \tag{1}$$

where $\eta(\vec{x}, t)$ is the water height at point $\vec{x}$ at time t, n_0 a constant offset, N the number of waves and a_i, ω_i, ϕ_i the amplitude, angular frequency and phase function of the i:th wave [Jeschke and Wojtan, 2015].

An important realisation is that two waves with the same angular frequency, oscillating at the same rate, can be different due to the phase function. Consider a stationary world space point. Two waves, with the same angular frequency, pass through the point (the point travels the length of an entire wave period) in the same amount of time but the lateral speed at which they pass may be different. If wave A travels much faster than wave B, wave A will seem stretched out in relation. On the other hand, if wave A travels much slower than wave B, wave A will seem compressed. Thus, for a wave, one can say that the angular frequency denotes the temporal rate of the oscillations and the phase function denotes the spatial rate of repetition. See Figure 1 for an illustration of this concept.

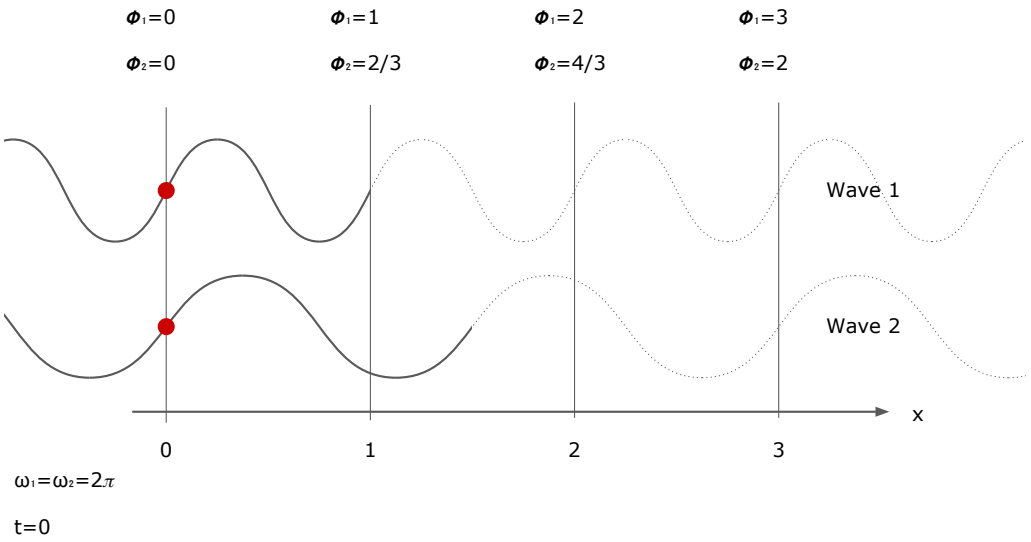


Figure 1: An illustration of two waves with the same angular frequency, $\omega_1 = \omega_2 = 2\pi$, but different phase functions, $\phi_1(x) = x$ and $\phi_2(x) = \frac{2}{3}x$. While both waves travel through a stationary world space point (denoted by the red dot) in the same amount of time, the phase speed of wave 2 is higher than that of wave 1.

The *phase speed*, c_i, of a wave is the spatial rate at which a point on the wave, say the crest, travels through the world under the *high frequency approximation*. It is given by the following equation:

$$c_i = \frac{\omega_i}{k_i} \tag{2}$$

where k_i is the *wavenumber* of the i:th wave [Jeschke and Wojtan, 2015]. The wavenumber denotes the number of times a wave repeats itself in one spatial unit. It is related to the *wavelength*, λ_i, the spatial extent of a wave period, by $k_i = 2\pi/\lambda_i$.

The angular frequency, ω_i, and the wavenumber, k_i, of a wave are connected by the *dispersion relation*:

$$\omega_i = \sqrt{(gk_i + \frac{\sigma}{\rho}k_i^3)\tanh(k_i h|_{\vec{x}})} \tag{3}$$

where g is the gravity constant, σ the surface tension, ρ the density of the water and $h|_{\vec{x}}$ the water depth at $\vec{x}$ [Jeschke and Wojtan, 2015]. This equation is an important part of water behavior. It states the relationship between angular frequency, wavenumber and water depth. As the angular frequency remains constant throughout a wave's lifetime [Peachey, 1986, Jeschke and Wojtan, 2015], the relation shows how the wavenumber (and consequently the wavelength) changes with water depth. For a given water depth, a wave with high angular frequency will have a high wavenumber and a wave with low angular frequency will have a low wavenumber.

This is intuitive as in real-life, waves with short wavelengths tend to oscillate quickly whereas waves with long wavelengths tend to oscillate slowly.

Using the dispersion relation, one may express the phase speed c_i as a function of ω_i, k_i and $h|_{\vec{x}}$:

$$c_i = \frac{\omega_i}{k_i} = \sqrt{(\frac{g}{k_i} + \frac{\sigma}{\rho}k_i)\tanh(k_i h|_{\vec{x}})} \tag{4}$$

In this form, it can be seen that the phase speed, i.e., the speed at which a wave propagates, tends to infinity both as $k_i \to 0$ and $k_i \to \infty$, and increases with the depth of the water. In addition, the phase speed is almost independent of water depth for capillary waves (large k_i) and deep water waves (large $h|_{\vec{x}}$) as $\tanh \approx 1$.

Under the high frequency approximation, a wave propagates according to the Eikonal equation,

$$|\nabla\phi| = \frac{1}{c}, \tag{5}$$

and travels through space with phase speed c [Jeschke and Wojtan, 2015]. The difference in phase between two world space points is then the time it takes for the wave to travel between the points. The high frequency approximation is formally valid in the limit when the angular frequency of a wave tends to infinity [Runborg, 2007] and it is a good approximation when the wavelength is small compared to features of the boundary domain. In particular, under the high frequency approximation, a point on a wave travels in a straight line in homogeneous media and waves do not diffract when grazing obstacles. Rawlinson et al. [2008] show the derivation of the Eikonal equation from the elastic wave equation.

When water depth is constant, the phase speed is constant and thus the phase of a wave is a linear function of distance, $\phi_i(x) = x/c_i = k_i x/\omega_i$, and the water surface is a sum of plane waves:

$$\eta(\vec{x}, t) = \eta_0 + \sum_{i=1}^{N} a_i \sin(\vec{k}_i \cdot \vec{x} - \omega_i t)$$

where the wave vector $\vec{k}_i$ denotes the direction of propagation with $\left|\vec{k}_i\right| = k_i$.

However, when the water depth and consequently the phase speed is not constant, the phase function is non-linear and often multi-valued [Jeschke and Wojtan, 2015].

The energy density D_i of wave i is given by:

$$D_i = \frac{(\rho g + \sigma k_i^2)a_i^2}{2}. \tag{6}$$

Finally, energy propagates at the rate of the *group speed*:

$$c_g = \frac{d\omega}{dk}. \tag{7}$$

3.2 The Wavefront

It is straightforward to describe the motion of water waves using linear wave theory for water of uniform depth, as the waves travel at constant phase speed and can be described by plane waves. However, when the seafloor is heterogeneous in depth, the wavenumber and consequently the phase speed of a wave will vary according to the dispersion relation.

When viewing a water wave travelling in a heterogeneous depth field from above, different points on the wave will travel at different speeds and it is convenient to introduce the concept of a *wavefront*. A wavefront is the set of points of constant phase away from a source. The points form a curve in R^2.

As different points along a wavefront travel at different speeds, the wavefront bends towards areas of lower speeds. This is what makes ocean waves that travel chaotically at sea line up with the coastline as they approach shallow waters.

An initially straight wavefront may bend, stretch and even fold over itself as it travels across the water surface. In practice, a wavefront will develop self-intersections even in a simple underlying depth field. An example of this is the *swallowtail pattern* that appears when a straight wavefront enters a circular area of gradually slower speeds and bends over itself [Hauser et al., 2006]. See Figure 2 for an illustration of this phenomenon.

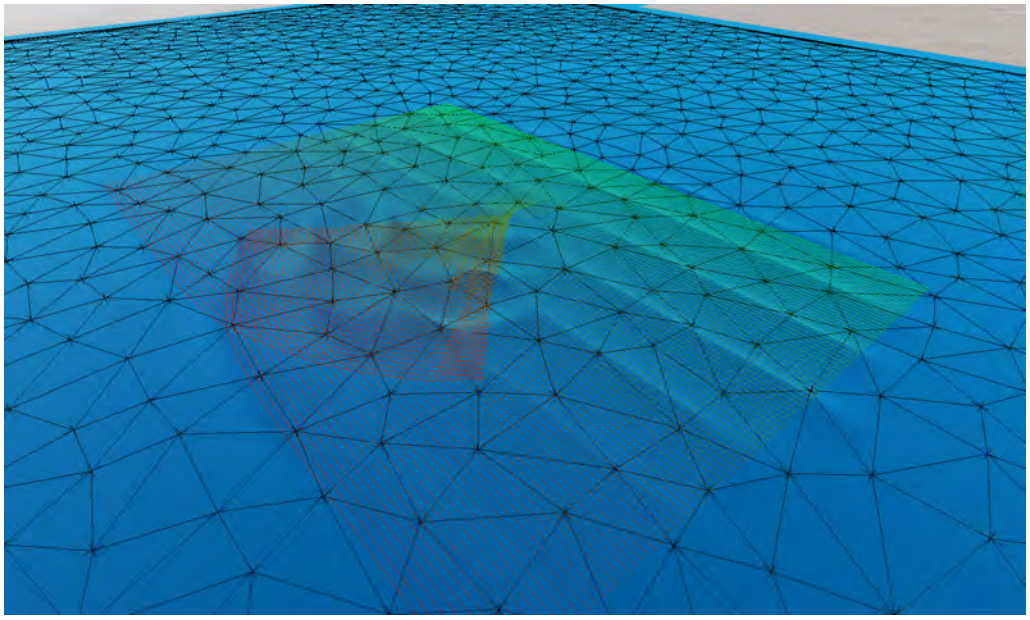


Figure 2: A wavefront forms a swallowtail pattern as it passes over a local low speed area. Multiple time steps of the wavefront are shown transitioning from green to red with increasing phase.

The observation that a wavefront can pass over a world space point multiple times in a heterogeneous depth field provides intuition behind the multi-valued nature of the phase function. Each value of the phase function at a world space point describes a time the wavefront moved passed the point.

4 Method

The algorithm for animating and rendering the water surface was developed using research in the field of computer graphics and oceanography as reference. The thesis covers the subset of the accomplished work that is relevant according to the problem statement.

The thesis project was carried out at the video game development company DICE [2017] in Stockholm, Sweden, and the algorithm was integrated into their in-house game engine, Frostbite [2017].

4.1 Overview

The proposed algorithm for generating and rendering the ocean consists of two steps: the simulation step and the rendering step. The simulation step is based on the novel approach to wavefront construction developed by Jeschke and Wojtan [2015] in which wavefronts are propagated across a virtual scene and wave parameters are recorded as they pass over the triangles of a coarse 2-dimensional triangle mesh (coarse mesh) covering the ocean surface. The rendering step generates the geometrical displacement and normals necessary to accurately render the ocean surface by sampling the recorded wave parameters stored on the coarse mesh. In order to render the ocean at any spatial scale and provide a seamless transition between different levels of detail, individual waves are filtered by wavelength as shown by Bruneton et al. [2010].

As propagating a large number of wavefronts across a virtual scene is computationally expensive, the simulation is a pre-computation step typically triggered by an artist during the content authoring process. This makes it possible to use advanced models in the simulation without taking into account the quality/performance balance otherwise constantly present in real-time graphics. In contrast, performance is crucial in the rendering step since it is triggered each frame of the animation and interactive frame rates are required. In practice, the simulation step is several orders of magnitude slower than the rendering step.

4.2 Simulation

The next section will describe in detail how the simulation is designed to generate, propagate and record wavefronts across any virtual scene. As there are many complex steps involved, the focus will be on the robustness of the presented solutions.

4.2.1 World Representation

As the purpose of this thesis is to develop a water system that reacts to the environment in a believable manner, the simulation step must have the ability to query the virtual scene in which the water is part of. Since the equations of motion for a wave depends on the water depth, the simulation must be able to query the water depth at any world space location. Additionally, the ability to determine where a line segment intersects any solid object in the scene, a so called line-cast, is required to handle wavefront reflection.

In order to limit the scope of the implementation, only the height map of the terrain is considered when a query is performed. This is a reasonable limitation as the terrain is what is used to represent the majority of solid mass in a scene. When a query is performed, the height map is sampled at a specified world space resolution using bi-cubic interpolation. The depth is computed by subtracting the terrain height sample from the y-coordinate of the water surface at the requested location. The line-cast query uses binary search along the line segment to pinpoint the exact point of intersection, if any.

The independent world space sampling resolution makes it possible to ignore high frequency details in the terrain and the bi-cubic interpolation scheme guarantees a continuous depth function value and gradient. These properties are crucial in order to achieve a good looking simulation as the stability of the simulation greatly depends on the smoothness of the depth field with respect to the the integration scheme used to propagate the wavefronts.

4.2.2 Coarse Mesh Generation

A two-dimensional coarse triangle mesh was chosen to be the shared medium between the simulation step and the rendering step as proposed by Jeschke and Wojtan [2015].

When linear interpolation is used to interpolate the wave data from each coarse mesh triangle vertex, only wavefronts that travel in a straight line can be accurately captured. Higher order interpolation schemes are able to capture more detail but since the wave data is only recorded at the vertices of the triangle, there is a limit to what wavefront behavior can be captured inside of the triangle boundary. In order to capture as much as possible of the wavefront behavior, several triangles are combined into a coarse mesh that span the surface of the water volume.

The coarse mesh is adaptively tessellated to conform to the terrain of a scene to take advantage of the fact that different regions give rise to different wavefront behavior. As wavefronts tend to refract and reflect around shores, the coarse mesh must be well tessellated around the shoreline. Similarly, as initially straight wavefronts tend to remain straight as they travel in deep water, the coarse mesh can be

loosely tessellated in these regions without loss of quality.

See Figure 3 for a visual representation of the coarse mesh of an example scene. Note that the visual representation is only used for debugging purposes and is not rendered in practice.

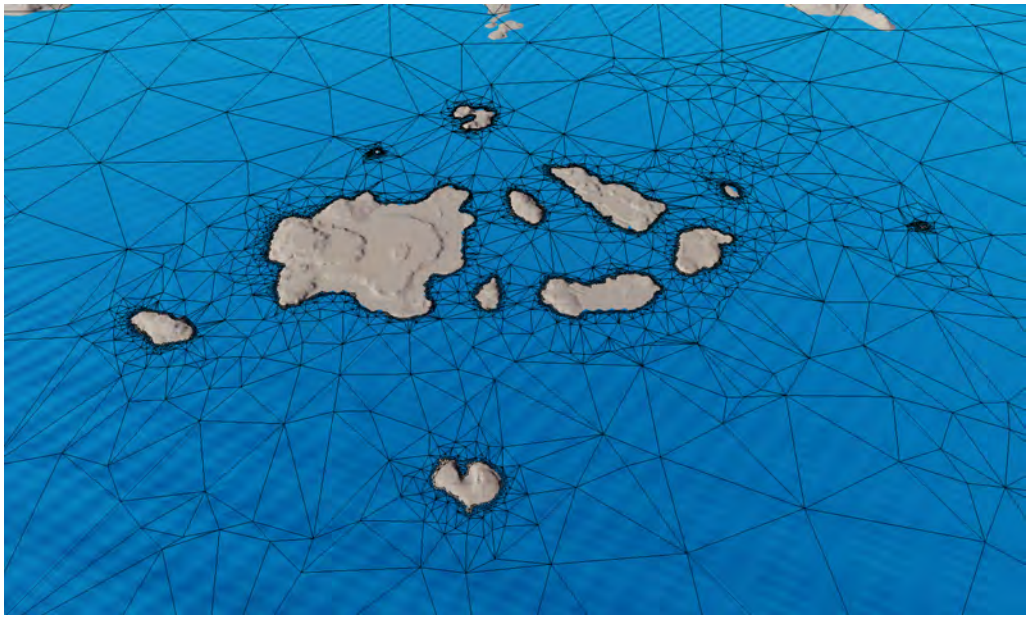


Figure 3: A visual representation of the coarse mesh (black triangles) for an example scene (white terrain) generated using the method described in section 4.2.2. Note how the sampling approach maintains a high density of vertices along the shoreline while not over-sampling the region of flat bottomed shallow water in between the islands.

To generate the coarse mesh for a given scene, a set of vertices are first generated using Poisson disk sampling [Cook, 1986]. Secondly, the Delaunay triangulation [Aurenhammer, 1991] of the set of vertices is found using the concept of convex hulls [Ericson, 2004]. Finally, unwanted triangles are removed and the remaining vertices and triangles are taken as the coarse mesh.

Poisson Disk Sampling In theory, any sampling scheme could be used to generate the vertices of the coarse mesh that spans across the scene. However, since high vertex density is preferred close to shores and low vertex density is preferred in deep water, Poisson disk sampling was chosen to generate the vertices.

In a Poisson disk distribution, any sample is located at least a minimum distance r apart from any other sample. By varying the minimum radius r using a depth-dependent function, it is possible to have different vertex densities in different parts of a scene.

To generate the Poisson disk sampling, the two-dimensional version of the algorithm proposed by Bridson [2007] was implemented and extended to support spatially varying radii. The Smaller-disks sizing function was used in order to achieve full coverage of the water surface [Mitchell et al., 2012]. In order for the algorithm to produce triangles of reasonable size given any radius function, the radius from

the function is clamped so that it lies within a fixed range $[r_{\min}, r_{\max}]$. Clamping the radius in this manner also lets the algorithm terminate in a reasonable amount of time.

Bridson's algorithm takes the extents of the sample domain and the minimum radius r as parameters and produces a Poisson disk sampling in linear time. First, a background grid with cell size equal to $r/\sqrt{2}$ is initialized to accelerate neighborhood searches. An initial sample is randomly picked from the sample domain and inserted into the background grid and into an *active list*. While the active list is not empty, a random point p is picked from it and k new candidate points are generated in a circular annulus between r and $2r$ around p. For each candidate point, the 9 neighboring background grid cells are checked for a potential overlap with an existing sample. If a candidate point does not violate the circular area of radius r around any existing sample, the candidate becomes a new sample and is added to the background grid and active list. In practice, $k = 30$ worked well.

The algorithm was extended to support spatially varying radii by introducing a radius function $r = f(\vec{x})$. The background grid must now have a cell size of $r_{\min}/\sqrt{2}$ to avoid conflicts. The Smaller-disks sizing function accepts a candidate point if the distance between it and any existing sample is less than the minimum of f(candidate point) and f(sample point)).

The obvious problem with this extension is that the worst case number of neighboring background grid cells to check increases dramatically. However, it did not prove to be a problem in practice.

Next, the radius function must be chosen. If the radius function is made proportional to the water depth h, the density of the generated points will increase as the depth decreases:

$$r = k\,|h|\,,$$

where k is a user defined constant. Negative depth values occur where the terrain of the scene is above the water surface and taking the absolute value makes the sampling symmetric with respect to the shoreline. Coarse mesh vertices are needed on both sides of the shoreline in order for coarse mesh triangles to cover the space up to the very start of the terrain. Choosing $k = 1$ is reasonable and will produce points approximately 1 unit apart at a depth of 1 unit. A problem with this sampling approach is that it generates too many points in large areas of shallow water where the seabed is nearly flat. Since a low number of coarse mesh triangles is critical for performance in the rendering step, the sampling approach must be improved.

The observation that an initially straight wavefront will remain straight when it travels in an area of uniform depth can be used to construct a sampling in a similar way: The radius function can be made proportional to the inverse length of the gradient at the candidate point:

$$r = \frac{k}{\max(\epsilon, |\nabla h|)},$$

where ϵ is a small number that determines the size of the largest triangle and prevents division by zero. With this approach, it is important that the underlying

depth field is smooth and does not contain too high-frequency features. In practice, this is achieved by the relatively sparse terrain representation described in 4.2.1. This sampling approach eliminates the high point density in shallow waters where the seabed is nearly flat but it also disregards the shoreline and nearly flat bottomed shores are sampled sparsely. Shores need to be sampled densely no matter the gradient in order for the simulation to accurately capture wave reflections. Thus, the sampling approach needs to be improved further.

A good radius function was achieved by combining the two sampling approaches into one:

$$r = \frac{k\,|h|}{\max(\epsilon, |\nabla h|)}$$

With this sampling approach, shores are sampled densely and flat bottomed shallow waters are sampled sparsely, as desired (See Figure 3). In practice, $k = 1$ and $\epsilon = 0.1$ was found to be suitable for scenes where 1 world unit corresponds to 1 meter.

Delaunay Triangulation In order to create the coarse mesh triangles, the Delaunay triangulation of the coarse mesh vertices is computed. The Delaunay triangulation maximizes the minimum interior angle of all triangles in the triangulation.

The method chosen to compute the Delaunay triangulation takes advantage of the fact that there is a connection between the Delaunay triangulation of a set n-dimensional points and the convex hull of the set of points lifted to a n+1-dimensional parabola [Edelsbrunner and Seidel, 1985]. First, the set of 2-dimensional coarse mesh vertices are lifted to a 3-dimensional parabola where the third coordinate is taken as the squared distance to the origin:

$$\vec{p} = \begin{pmatrix} v_x & v_y & v_x^2 + v_y^2 \end{pmatrix}^T.$$

Intuitively, one can think of the points as being placed on the bottom surface of a bowl that extends upwards in the positive z-direction and that is placed on a table made up of the x and y axis. Next, the convex hull of the set of 3-dimensional points is computed. Finally, only the triangles of the convex hull that can be seen from an observer at $z = -\infty$ looking in the positive z-direction are kept, i.e., triangles whose normals $\vec{n}$ fulfill $n_z < 0$. The 3-dimensional triangles are then projected onto the original two-dimensional plane by discarding the z-coordinate. This procedure yields the sought after closest-point Delaunay triangulation of the original point set [Edelsbrunner and Seidel, 1985].

By transforming the problem in this way, the focus is shifted from implementing a robust two-dimensional Delaunay triangulation algorithm to implementing a robust three-dimensional convex hull generation algorithm.

Convex Hull Generation To construct the convex hull of the lifted set of points in three dimensions, the algorithm proposed by Gustafsson [2013] was implemented.

The algorithm was chosen over other more widespread algorithms because of its simplicity and promising inherent robustness. As the algorithm is topology-based, it will always produce a two-manifold mesh where every edge belongs to exactly two triangles as long as it terminates. However, the algorithm is not guaranteed to terminate due to the limited precision of floating point numbers. Since termination is crucial in a production environment, the algorithm was extended and made robust at the cost of performance.

Gustafsson's algorithm is inspired by support mapping methods such as the Gilbert-Johnson-Keerthi (GJK) algorithm [Gilbert et al., 1988] in that it uses a *support function* to incrementally build the convex hull. Given any direction $\vec{d}$, the support function returns the point in the set that lies furthest in that direction, i.e., the point $\vec{p}_i$ for which $\vec{p}_i \cdot \vec{d} = \max(\vec{p}_1 \cdot \vec{d}, \vec{p}_2 \cdot \vec{d}, \ldots, \vec{p}_n \cdot \vec{d})$ holds. From this follows that any point returned by the support function lies on the surface of the convex hull of the point-set.

The algorithm starts by using the support function to construct an initial convex hull consisting of two oppositely oriented triangles from 3 points. The rest of the points in the input set are added to a *remaining point set*. In each iteration of the algorithm, a triangle is picked from the convex hull and the support function is queried for the point $\vec{p}_{\text{new}}$ in the remaining point set that lies furthest along the triangle normal. If $\vec{p}_{\text{new}}$ lies in front of the triangle, the triangle is replaced by 3 new triangles that connect the edges of the original triangle and $\vec{p}_{\text{new}}$. The point $\vec{p}_{\text{new}}$ is then removed from the remaining point set. This expansion step may cause the hull to become concave and the expansion step is followed by an unfolding step that enforces convexity. In the unfolding step, any two triangles that share a *concave edge* rotate the 4 point indices so that the shared edge is *flipped* and rendered convex. See Figure 4 for an illustration. A concave edge is an edge where the angle between its two triangles is less than π radians. Since an edge flip might render neighboring edges concave, the procedure is repeated until all edges are convex. Flipping a concave edge increases the volume of the hull and since the volume can not increase indefinitely, the unfolding step must terminate in theory. Finally, the algorithm terminates when no triangle on the convex hull can be expanded further.

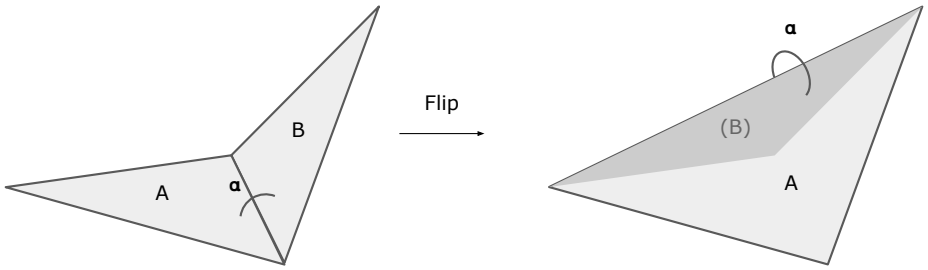


Figure 4: A concave edge is *flipped* by rotating the 4 point indices of the two connected triangles in the unfolding step of the convex hull generation algorithm. The edge is considered concave when the angle α between the triangles that share the edge is less than π radians. As seen in the figure, a flip renders a concave edge convex.

In practice, the limited precision of floating point numbers might cause the unfolding step to cycle through a set of flips indefinitely. Indefinite cycling might occur whenever the concavity tests for a series of edges do not agree about the convexity of the hull and the same edges are repeatedly flipped in some cycle. However, the exact same edge that is part of the exact same two triangles should not be flipped twice as this will undo the work of the first flip. To make the algorithm robust, the first flip is assumed to be correct and a *flip hash set* is used to store the 4 vertices associated with each performed flip. At the start of the unfolding step, the flip hash set is initialized to the zero set. Then, whenever a flip is about to be performed, it is first checked against the flip hash set. If there is a match and the flip is about to undo the work of a previous flip, the flip is skipped. If not, the concavity test is assumed to be correct and the flip is performed and added to the flip hash set. As there is a finite number of possible flips, the extension will allow termination of the unfolding step and make the algorithm robust against precision errors. In practice, the high number of possible flips was not a problem and the algorithm successfully terminated in a reasonable amount of time even for very large scenes with large variations in triangle size.

Removal of Hidden Triangles In most scenes, some of the generated triangles of the coarse mesh will lie completely below the terrain of the world. As no water will be visible in these areas, it is unnecessary to store wave data on the triangles.

In order to save memory, the unnecessary triangles are removed from the coarse mesh in a post-processing step. First, all vertices of the coarse mesh that lie below the terrain are marked as invisible. Then, all coarse mesh triangles that are

connected to only invisible vertices are removed. This simple procedure might remove partly visible triangles in theory. However, as long as the shoreline was well tessellated it did not prove to be a problem in practice.

4.2.3 Wavefront Generation

In order for the simulation of the water to be realistic and at the same time allow for artistic control, a flexible system for generating wavefronts must be in place. A single virtual scene may contain many different settings in which water should be present. There should be large ocean waves at open sea, less violent waves in bays and calm water in canals further inland. There might be waterfalls as a result of drops in elevation or man-made construction such as dams. To capture these design goals, the system provides a set of *wavefront generators* that can be placed in a scene and allow artists to tailor the scene to their vision.

All wavefront generators let the user specify a minimum and maximum angular frequency range that the wavefronts created from the generator will be limited to. A generator typically creates several wavefronts and the angular frequency value used for an individual wavefront is restricted to this range. In addition to the angular frequency range, a maximum amplitude to wavelength ratio is specified to control the maximum steepness of all waves that result from the generator.

Sampling the Ocean Wave Spectrum In order to avoid having an artist specify the parameters for each wavefront manually in an open sea setting, wavefronts can be generated by sampling an ocean wave spectrum. The Pierson-Moskowitz spectrum [Pierson and Moskowitz, 1964] is used and extended to two dimensions using a simple falloff function that eliminates waves perpendicular to the wind direction. Two integer values denote the number of angular frequency samples and the number of directional samples respectively.

In contrast to Fréchot [2006] that use an adaptive sampling scheme to finely sample areas of high wave energy, a good range of angular frequencies is preferred when generating wavefronts. Thus, a simple uniform sampling scheme is used. A set of densities are computed by sampling the spectrum using a jittered regular grid in the angular frequency/wind angle plane. When sampling the wave spectrum, the density is discretized and converted to wave amplitude as shown by Fréchot [2006]. Then for each sample, a wavefront is created (with the corresponding angular frequency, propagation direction and amplitude) and spawned just outside the bounds of the water simulation area.

Manual Placement of Wavefronts To allow fine grain control over waves in a particular location, the system provides a planar wavefront generator and a circular wavefront generator. The planar wavefront generator creates a number of wavefronts that lie parallel to the x-axis and at the position of the generator entity. The circular wavefront generator creates a number of circular wavefronts around the position of the generator entity.

Both of these generators let the user specify a minimum and maximum energy range and the number of wavefronts to create. The energy range must be within

$[0, 1]$ and denotes the amplitude from zero to maximum steepness as dictated by the maximum amplitude to wavelength ratio. For each wavefront, an angular frequency value is uniformly sampled from the angular frequency range and an energy value is uniformly sampled from the energy range. Then, the amplitude of the wavefront is determined by multiplying the initial wavelength of the wavefront with the maximum amplitude to wavelength ratio and the sampled energy value.

4.2.4 The Wavefront

In the simulation step, a wavefront is represented by a list of n vertices and a list of $n - 1$ line segments. The vertices and line segments form a curve on the water surface and describe the current shape and location of the wavefront in world space. The resolution of the wavefront changes as the wavefront is propagated across the water surface [Vinje et al., 1993, Gonzato and Le Saëc, 2000, Jeschke and Wojtan, 2015] in order for the piecewise linear representation to approximate the true shape of the wavefront even as it deforms considerably. See Table 1 for what is stored in memory for each wavefront.

Wavefront		
Property	Symbol	Type
Angular Frequency	ω	Float
Phase	ϕ	Float
Previous Phase	ϕ'	Float
Vertices	N/A	Vertex[ ]
Segments	N/A	Segment[ ]
Covered Mesh Edges	N/A	MeshEdge{ }
Initial Amplitude/Wavelength Ratio	N/A	Float
Min Amplitude/Wavelength Ratio	N/A	Float
Max Amplitude/Wavelength Ratio	N/A	Float
Max Amplitude/Depth Ratio	N/A	Float

Table 1: The Wavefront Data Structure

The *Angular Frequency* and *Phase* are the parameters of the wave that are shared across all vertices and segments of the wavefront. Waves with an amplitude/wavelength ratio below *Min Amplitude/Wavelength Ratio* are not visible and should be removed. As the simulation does not capture non-linear effects of water behavior, such as breaking waves, the amplitudes of the segments are simply clamped to lie just below the breaking point of the wave using the *Max Amplitude/Wavelength Ratio* value. In a similar fashion, *Max Amplitude/Depth Ratio* is used to clamp the amplitudes of segments that exceed the current water depth. The *Initial Amplitude/Wavelength Ratio* is used to compute the initial amplitude of the segments of a wavefront when the wavefront is manually placed within a scene as random uniform sampling of this value gives much more believable wave profiles than random uniform sampling of the amplitude directly. This parameter is not used when the wavefront is generated from an ocean wave spectrum.

The integration scheme proposed by Jeschke and Wojtan [2015] was adopted and used to deduce the vertex and segment data structures. In addition to its current state, each vertex and segment also stores some of its previous state so that certain values can be interpolated between time steps when the wavefront is recorded onto the coarse mesh. See Table 2 and Table 3 for what is stored in memory for each vertex and each segment respectively.

Vertex		
Property	Symbol	Type
Position	$\vec{p}$	(Float, Float)
Normalized Travel Direction	$\vec{d}$	(Float, Float)
Depth Below Surface	h	Float
Wavenumber	k	Float
Phase Speed	c	Float
Group Speed	c_g	Float
Previous Position	$\vec{p'}$	(Float, Float)
Previous Normalized Travel Direction	$\vec{d'}$	(Float, Float)
Previous Wavenumber	k'	Float
Previous Phase Speed	c'	Float
Previous Group Speed	c'_g	Float
Within Bounds Flag	N/A	Boolean
Moving Away Flag	N/A	Boolean
Reflection Depth	N/A	Integer

Table 2: The Vertex Data Structure

Segment		
Property	Symbol	Type
Amplitude	a	Float
Energy Respecting Amplitude	a_D	Float
Previous Amplitude	a'	Float
Previous Energy Density	D'	Float
Previous Length	L'	Float
Degenerate Flag	N/A	Boolean
Inverted Amplitude Flag	N/A	Boolean

Table 3: The Segment Data Structure

The division of wave parameters between Vertex and Segment where made to facilitate the integration scheme described in section 4.2.6.

The *Within Bounds Flag* and *Moving Away Flag* are used to keep track of where a vertex is in relation to the extents of the water surface. If a vertex is not within bounds and it is moving away, it is removed from the simulation. The *Reflection Depth* value counts the number of times a vertex has been reflected off the shore and is analogous to the recursion depth value used in conventional ray tracing of light rays for image generation. The *Degenerate Flag* is used to denote segments that are

no longer a good fit for the simulation and are about to be removed. The *Inverted Amplitude Flag* is used to keep track of the sign of the amplitude of a segment. An explicit flag makes computations involving amplitude and energy density simpler and results in more readable code.

4.2.5 The Simulation Step

The simulation step propagates the wavefronts across the water surface and records wave data onto the coarse mesh whenever a wavefront segment passes over a coarse mesh vertex. The wave data samples are connected by *chains* stored on the coarse mesh edges. When all wavefronts have been propagated, the chains are combined into *wave overlaps* stored on the coarse mesh triangles. Finally, the coarse mesh triangles form the coarse mesh that is used as the input to the rendering step.

The simulation follows a simple procedure:

1. Propagate wavefronts forward in time
2. Record wavefronts onto coarse mesh
3. If no wavefronts are left, end the simulation; otherwise return to step 1

The propagation step is described in Section 4.2.6 and the recording step is described in Section 4.2.8.

4.2.6 Wavefront Propagation

The wavefront propagation step consists of the following 12 sub-steps. Each wavefront in the simulation is handled, in order, within each sub-step.

1. Clean up covered edges
2. Remove degenerate segments
3. Maintain wavefront resolution
4. Compute wavenumbers
5. Conserve energy
6. Compute phase speeds
7. Handle refraction
8. Integrate position
9. Check within bounds
10. Compute depths
11. Handle reflection
12. Identify degenerate segments

The remainder of this section describe the details necessary to robustly implement the wavefront propagation.

Clean up covered edges Each wavefront stores the set of coarse mesh edges that it covers at any given movement. A wavefront covers a coarse mesh edge if any of its segments intersect the edge.

In this step, the set of covered edges is iterated and any edge the wavefront no longer covers is removed from the set.

Remove degenerate segments This step iterates the list of segments and removes all segments that are marked as degenerate. A segment is marked as degenerate in the *Identify degenerate segments* step and may become degenerate for a number of different reasons. For example, any segment that moves away from the water simulation area is marked as degenerate.

If a wavefront does not contain any segment after this step, it is removed from the simulation.

Maintain wavefront resolution In order to reliably simulate a wavefront, a constant spatial resolution must be enforced as it is propagated across the scene. If the resolution is insufficient, the piecewise linear representation will poorly represent the true shape of the wavefront.

Previous simulation methods keep the resolution constant in world space [Gonzato and Le Saëc, 1997]. This works well for slowly expanding or contracting wavefronts where neighboring vertices are moving in roughly the same direction. However, as the method does not take the curvature of the wavefront into account, it performs poorly when there are sharp changes in travel direction or kinks in the wavefront. In such cases, the wavefront will be under-sampled and its motion unpredictable.

A common problematic case is when an initially straight wavefront develops a swallowtail pattern as it passes through a local low speed region (See Figure 2). In this case, two kinks will develop on either side of the slow moving center-section of the wavefront as the outer edges fold over it. There will be large differences in the travel direction of neighboring vertices that lie on either side of a kink, and the sharp corner will be poorly represented if the resolution is held constant in world space.

To solve the problem, the observation that a wavefront with kinks in world space becomes a smooth bicharacteristic strip when it is transformed into reduced phase space [Osher et al., 2002, Hauser et al., 2006] is used and the wavefront resolution is held constant in reduced phase space. See Figure 5 for an illustration of a wavefront in world space and its corresponding shape in reduced phase space.

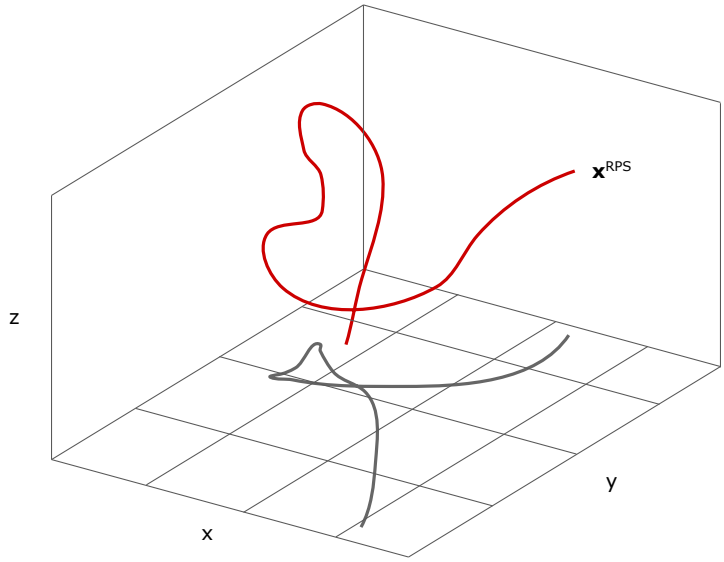


Figure 5: An illustration of a wavefront in world space (dark gray curve) and its corresponding shape in reduced phase space (red curve). Note that the wavefront does not self-intersect nor display kinks in reduced phase space.

First, each vertex position $\vec{x}$ is lifted to reduced phase space:

$$\vec{x}^{RPS} = \begin{pmatrix} x_x & x_y & \alpha \arccos(d_x) \operatorname{sign}(d_y)) \end{pmatrix}^T,$$

where α is a normalizing scaling factor and

$$\operatorname{sign}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}.$$

Then, for each segment, the minimum distance between its 2 vertices in three-dimensional reduced phase space is determined while taking the periodicity of the z-coordinate into account.

Let δ denote the ideal world space distance between wavefront vertices. If the length of a segment in reduced phase space is larger than $1.5\,\delta$, the segment is subdivided to two segments by inserting a mid-point vertex. If the segment length in reduced phase space is below $0.5\,\delta$, it is collapsed and the two vertices making up the segment are replaced with a mid-point vertex. The collapse removes the original segment and connects its two adjacent segments together. In both cases, linear interpolation is used to generate the state of the new vertex. See Figure 6 for an illustration of segment subdivision and collapse.

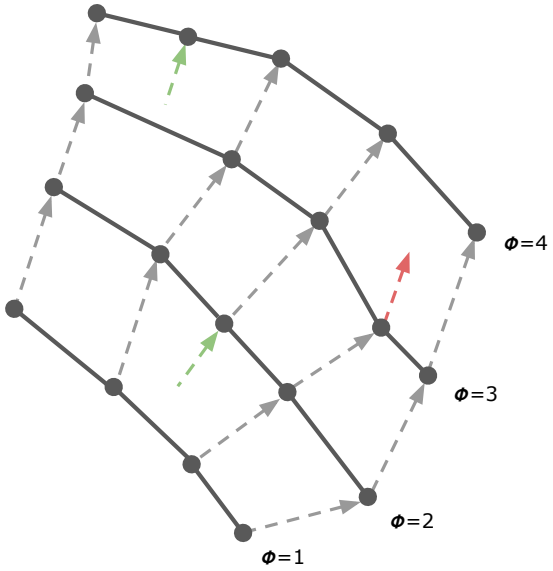


Figure 6: An illustration of segment subdivision and collapse during wavefront propagation. A single wavefront is shown over four consecutive time steps. Solid dark gray circles and lines represent wavefront vertices and segments respectively. Dashed gray arrows indicate travel paths for the vertices. Green and red arrows indicate the addition and removal of a mid-point vertex during segment subdivision and collapse respectively.

The α scalar in the transform to reduced phase space determines the importance of wavefront curvature in the resolution measurement. When $\alpha = 1$, a difference of 1 radian in the propagation direction of two neighboring vertices will weigh equal to 1 unit of spatial separation. Increasing α will increase the sampling resolution of curves in the wavefront. Setting $\alpha = 0$ reduces the measurement to world space and the tessellation behavior to that of previous methods. In practice, a value of $\alpha = 2$ was found to be keep wavefronts well-formed.

Compute wavenumbers As the wavefront is propagated across the water surface, the water depth, h, at each vertex changes to reflect the distance to the seabed at the vertex position, $\vec{p}$. When the depth changes, the wavenumber of a wave must change according to the dispersion relation (Equation 3) and since the depth is not uniform across the wavefront, each vertex has its own wavenumber.

This step computes the wavenumber, k, of each vertex given its current water depth, h, using the fixed-point iteration scheme proposed by Jeschke and Wojtan [2015]. As the wavenumber, k, is related to the wavelength, λ, by $\lambda = 2\pi/k$, changing the wavenumber is analogous to changing the spatial rate of repetition of a wave.

First, the phase speed $c(k, h)$ given the current wavenumber, k, and water depth, h, is computed using Equation 4. Next, the wavenumber, k, is updated according to $k := \omega/c(k, h)$ from Equation 2. The two steps are iterated until k convergences. In practice, 20 iterations were sufficient.

Conserve energy This step conserves the energy stored in the wavefront as it stretches and contracts over time.

First, the group speed, c_g, of each vertex is approximated by finite differences of Equation 3:

$$c_g := \frac{\omega(k + \Delta k, |h|) - \omega(k, |h|)}{\Delta k},$$

with $\Delta k = 10^{-4}$.

Then, the segments are updated in the following manner: First, the current and previous group speed of the segment, $\hat{c}_g$ and $\hat{c}'_g$, is computed by averaging the group speeds of the two vertices. Second, the world-space length L of the segment is determined. Third, the energy density is computed:

$$D := \frac{\hat{c}'_g}{\hat{c}_g} \frac{L'}{L} D'.$$

Fourth, the wavenumber of the segment, $\hat{k}$, is computed by averaging the wavenumber of the two vertices. Fifth, the amplitude is calculated:

$$a := \sqrt{\frac{2D}{\rho g + \sigma \hat{k}^2}}.$$

Finally, the wavelength of the segment, $\hat{\lambda} = 2\pi/\hat{k}$, is multiplied with the Max Amplitude/Wavelength Ratio and the average depth of the two vertices, $\hat{h}$, is multiplied with the Max Amplitude/Depth Ratio in order to yield $a_{\text{max steepness}}$ and $a_{\text{max depth}}$ respectively. If $a > a_{\text{max steepness}}$, the waves are steep enough to break and should tumble over themselves in a non-linear fashion in order to dissipate energy. However, as the simulation is based on linear wave theory that is not able to capture this phenomenon, energy is simply removed from the simulation by setting $a := a_{\text{max steepness}}$ and recalculating the energy density using Equation 6. At this point, the Energy Respecting Amplitude is set to the current amplitude $a_D := a$. Next, if $a > a_{\text{max depth}}$, the wave will with high probability contribute to bring the water surface down below the seabed when the water is rendered. As this is likely a temporary problem caused by a sudden spike in the underlying terrain height, the energy density is left unchanged and only the amplitude is clamped by setting $a := a_{\text{max depth}}$. This behavior is not realistic but it increases the lifetime of the wavefronts and lets them reach further inland while preventing the resulting waves from intersecting the terrain, which increases the overall perceived quality of the simulation.

Note that the amplitude is calculated from the energy density each time step and only used when the wavefront is recorded onto the coarse mesh. Thus, clamping the amplitude does not influence the future of the simulation.

Compute phase speeds In this step, the phase speed c of each vertex is updated according to Equation 2, $c = \omega/k$, where ω is the angular frequency of the wavefront and k the most recent wavenumber of the vertex.

Note that this is the analytical phase speed $c(k, h)$ computed in the last iteration of the fixed-point iteration scheme in the *Compute wavenumbers* step.

Handle refraction In order to capture wave refraction, wavefront vertices refract as they pass contour lines in the underlying water depth field [Fournier and Reeves, 1986, Ts'o and Barsky, 1987, Gonzato and Le Saëc, 1997, Jeschke and Wojtan, 2015].

The gradient of the seabed, ∇h, is evaluated at the position of each vertex. If $|\nabla h(\vec{p})| > 10^{-2}$, Snell's law is used with the previous phase speed, c', and the new phase speed, c, to determine the new propagation direction of the vertex:

$$\vec{d} := \begin{cases} \vec{n}\sqrt{1-s^2} + \vec{t}\,s & |s| \leq 1 \\ \vec{d} - 2(\vec{d}\cdot\vec{n})\vec{n} & |s| > 1 \end{cases},$$

where

$$\vec{n} = \begin{cases} \nabla h(\vec{p})/|\nabla h(\vec{p})| & \vec{d}\cdot\nabla h(\vec{p}) \geq 0 \\ -\nabla h(\vec{p})/|\nabla h(\vec{p})| & \vec{d}\cdot\nabla h(\vec{p}) < 0 \end{cases},$$
$$\vec{t} = \begin{pmatrix} -n_y & n_x \end{pmatrix}^T,$$
$$s = \frac{c}{c'}\vec{d}\cdot\vec{t}.$$

Note that the normal $\vec{n}$ always point in the direction of $\vec{d}$ and that $\vec{d}$ is reflected along $\vec{n}$ if total internal reflection occurs ($|s| > 1$).

Integrate position In this step, the vertex positions are integrated forward in time using simple Euler integration:

$$\vec{p} := \vec{p'} + \vec{d}\,c\,\Delta t.$$

More advanced integration schemes could be used but Euler integration proved to work well in practice.

Check within bounds Since the vertices have been moved to new locations in the previous step, their state flags must now be updated.

Each vertex position is checked against the two-dimensional oriented bounding box of the *focus area*. The focus area is a part of the water surface in which one would like the wavefronts to start refract due to changes in depth. Outside the focus area, waves are assumed to be deep water waves. If no focus area is desired, the oriented bounding box of the focus area is set to that of the whole water surface.

The Within Bounds Flag is set to true if the vertex is inside the oriented bounding box. The Moving Away Flag is set to true if the vertex is not within bounds and its direction is pointing away from the oriented bounding box.

Compute depths In this step, the water depth, h, of each vertex is updated. In order for newly created wavefronts to hold their shape until they have entered the simulation area, the evaluation of water depth depends on the state flags of the vertices. The update is performed in two passes.

In the first pass, the water depth, h, is updated for all vertices that are within bounds, i.e., whose Within Bounds Flag is set to true. For a given vertex, the world is queried for the water depth, h, at $\vec{p}$.

In the second pass, vertices that are not within bounds are considered. For each such vertex, the water depth, h, is set to that of the nearest vertex, by index, along the wavefront that is within bounds. If no such vertex exists (no vertex on the wavefront is within bounds), the water depth, h, is set to a maximum value that corresponds to the depth in deep water, $h_{\max} = 10^2$.

Handle reflection In order to capture waves reflecting off different obstacles in the world, wavefronts are reflected when they pass the shoreline, defined by $h = 0$. See Figure 7 for an image of a wavefront reflecting off the shoreline in the simulation.

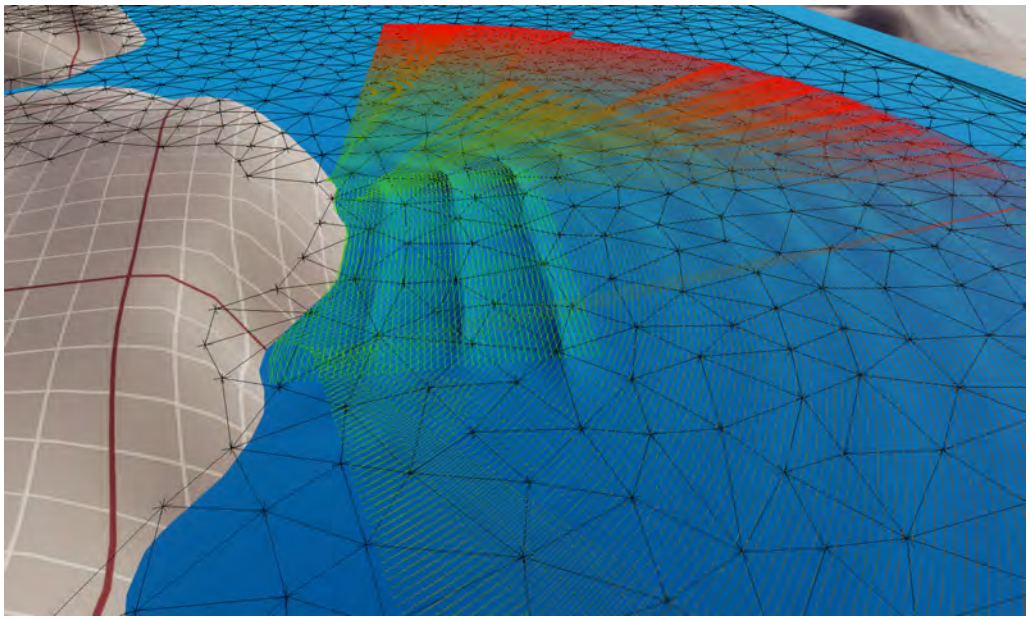


Figure 7: A wavefront reflecting off the shoreline within the simulation. Multiple time steps of the wavefront are shown transitioning from green to red with increasing phase. The coarse triangle mesh is shown in black. Underneath, the resulting water surface produced by the rendering step is shown in blue.

In order to limit the number of wavefronts and improve performance of the simulation and rendering step, each vertex keeps an internal counter of how many times it has reflected and loses its ability to reflect once the counter exceeds a set limit. In practice, the limit was set to 6 reflection bounces.

The reflection handling is performed in two passes. In the first pass, all vertices of the wavefront are iterated. The position $\vec{p}$ and the position in the previous time step $\vec{p'}$ are used as arguments in a line-cast query of the world. If the line intersects the shoreline, the vertex is temporarily marked as reflected and updated with:

$$
\begin{aligned}
\vec{p}_{\text{refl}} &:= \vec{p} - 2((\vec{p} - \vec{i}) \cdot \vec{n})\vec{n}, \\
\vec{d}_{\text{refl}} &:= \vec{d} - 2(\vec{d} \cdot \vec{n})\vec{n}, \\
\vec{p'}_{\text{refl}} &:= \vec{i}, \\
\vec{d'}_{\text{refl}} &:= \vec{d}_{\text{refl}}, \\
h_{\text{refl}} &:= h(\vec{i}), \\
\text{Reflection Depth}_{\text{refl}} &:= \text{Reflection Depth} + 1,
\end{aligned}
$$

where $\vec{i}$ and $\vec{n}$ is the intersection point and normal of the line-cast. Internally, the normal is set to the normalized gradient of the terrain at the point of intersection, $\vec{n} = \nabla h(\vec{i})/|\nabla h(\vec{i})|$.

In the second pass, mid-point vertices are inserted between segment vertex pairs in which one of the vertices reflects while its neighbor does not in order to better represent the kink in the wavefront that appears in such cases. See Figure 8 for an illustration. The mid-point vertices created during one time step should separate the wavefront into partitions in which the Reflection Depths of the vertices are equal and distinct from the vertices of neighboring partitions. If a single mid-point vertex is created during one time step, all vertices to the right of this vertex will have reflected one more time than the vertices to the left of it, or vice versa depending on from which direction the wavefront hit the shoreline. As the amplitudes of reflecting waves are inverted and the mid-point vertices split each segment that intersects the shoreline into two new segments (that lie on either side of the intersection), they also serve the purpose of clearly defining which segments should have their amplitude inverted.

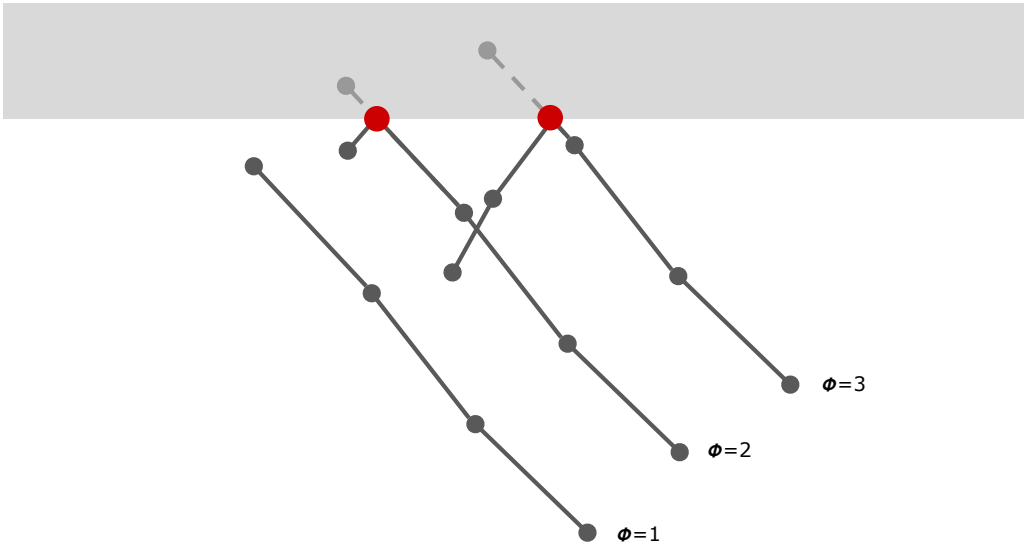


Figure 8: An illustration of reflection during wavefront propagation. A single wavefront is shown over three consecutive time steps. Solid dark gray circles and lines represent wavefront vertices and segments respectively. Vertex and segment positions before reflection handling are shown in dashed light gray. Red circles denote mid-point vertices created when a vertex is reflected while its neighbor is not.

Thus, a mid-point vertex is created and inserted in between vertex A and vertex B of segment S of a given segment if A was marked as reflected in the first pass while B was not and Reflection Depth$_A$ > Reflection Depth$_B$. The inequality check is required as a mid-point vertex between A and B should not be created if B reflected before A in some previous time step. The mid-point vertex is initialized as a clone of A with the modifications:

$$\begin{aligned} \vec{p} &:= \vec{j}, \\ \vec{p'} &:= \vec{p}, \\ h &:= h_B, \end{aligned}$$

where $\vec{j}$ is the intersection point of a line-cast from $\vec{p}_B$ to $\vec{p}_A$. The water depth h is set to that of vertex B in order for the mid-point vertex to travel the same distance as B in the next time step. When the mid-point vertex is created the original segment S is split into two segments, P and Q, that are connected to vertex A and B respectively, in addition to the mid-point vertex. P and Q are initialized as clones of S with the modifications:

$$\begin{aligned} L'_P &:= L'_S\,(1-s), \\ L'_Q &:= L'_S\,s, \\ D'_P &:= D'_S\,e, \\ \text{Inverted Amplitude}_P &:= \neg\text{Inverted Amplitude}_S, \end{aligned}$$

where

$$s = \begin{cases} |(\vec{j} - \vec{p}_B)|/|(\vec{p}_A - \vec{p}_B)| & \text{index of A is less than index of B} \\ |(\vec{j} - \vec{p}_A)|/|(\vec{p}_A - \vec{p}_B)| & \text{otherwise} \end{cases}$$

and e is a constant scalar in the range $[0, 1]$ denoting the amount of energy not lost in the collision. In practice, $e = 0.8$ was used.

If both A and B were marked as reflected in the first pass, the geometry of the wavefront is unchanged and S is updated with:

$$D'_S := D'_S\, e,$$
$$\text{Inverted Amplitude}_S := \neg\text{Inverted Amplitude}_S.$$

Identify degenerate segments In this step, the segments are iterated and degenerate segments are identified. There are three reasons for which a segment may be flagged as degenerate. If any of them are true, the Degenerate Flag of the segment is set to true.

First, a segment is marked as degenerate if the wave generated by the segment is not steep enough, i.e., if $a_D/\lambda_A <$ Min Amplitude/Wavelength Ratio $\vee\ a_D/\lambda_B <$ Min Amplitude/Wavelength Ratio is true where $\lambda_A = 2\pi/k_A$ and $\lambda_B = 2\pi/k_B$ are the wavelengths of the two vertices of the segment. Note that this is the same steepness measurement as used in the *Conserve energy* step. Waves that are deemed not steep enough have an unperceivable effect on the final water surface and the removal of these segments is crucial for good performance of both the simulation and rendering step. In practice, Min Amplitude/Wavelength Ratio $= 5 \cdot 10^{-3}$ was used.

Secondly, a segment is marked as degenerate if both of its vertices have left the water simulation area and are moving away, i.e., if Moving Away Flag$_A \wedge$ Moving Away Flag$_B$ is true.

Thirdly, a segment is marked as degenerate if any of its vertices are below the terrain of the world, i.e., if $h_A < -\epsilon \vee h_B < -\epsilon$ is true and ϵ is a small positive value to account for floating-point precision errors in the *Handle reflection* step. In practice, $\epsilon = 10^{-2}$ was used.

4.2.7 The Chain

In the recording step, wave data *samples* are recorded onto the coarse mesh in *chains* stored on coarse mesh edges. Whenever a wavefront segment passes a coarse mesh vertex, a wave data sample is constructed and a chain referencing this sample is created and added to each edge connected to the vertex. A sample stores a snapshot of the phase, amplitude, phase speed and normalized travel direction of the wavefront as it moved past the vertex. A chain stores the extent of a wavefront that has moved past an edge and is used to connect samples that may be safely interpolated along edges. A chain can have up to two samples associated with it.

See Table 4 and Table 5 for what is stored in memory for each sample and each chain respectively.

Sample	
Property	Type
Vertex	Pointer
Phase	Float
Amplitude	Float
Phase Speed	Float
Normalized Travel Direction	(Float, Float)

Table 4: The Sample Structure

Chain	
Property	Type
Wavefront	Pointer
Left Vertex Index	Integer
Right Vertex Index	Integer
Sample Count	Integer
Sample 0	Sample
Sample 1	Sample
Side Of Edge Determined Flag	Boolean
Side Of Edge Positive Flag	Boolean

Table 5: The Chain Structure

Let a vertex v_i on a wavefront be *covered* by a chain if the chain's Wavefront pointer references the wavefront and Left Vertex Index $\leq i \leq$ Right Vertex Index. A *determined chain* is a chain in which all wavefront vertices covered by the chain are located on one side of the line spanned by the coarse mesh edge the chain belongs to. Determined chains are usually created if the wavefront is parallel to the coarse mesh edge as the chain is created. An *undetermined chain* is a chain in which at least two of the wavefront vertices lie on opposite sides of the line spanned by the coarse mesh edge the chain belongs to. Undetermined chains may be created when a wavefront is perpendicular to a coarse mesh edge and a wavefront segment intersects the coarse mesh edge as it passes a coarse mesh vertex. As soon as a chain becomes determined, the Side Of Edge Determined Flag is set to true and the Side Of Edge Positive Flag is set to true if the covered wavefront vertices lie above the coarse mesh edge, otherwise it is set to false. A point $\vec{p}$ lies above a coarse mesh edge if $(\vec{p} - \vec{o}) \cdot \vec{n} > 0$, where $\vec{n}$ is the counter-clockwise normal of the edge and $\vec{o}$ is a point on the edge. A point lies on the *correct* side of a coarse mesh edge with respect to a determined chain if the point lies above the edge and the Side Of Edge Positive Flag is set to true or if the point does not lie above the edge and the Side Of Edge Positive is set to false. Once a chain has been determined its flags are fixed and the chain cannot become undetermined or change side unless it is merged with another chain.

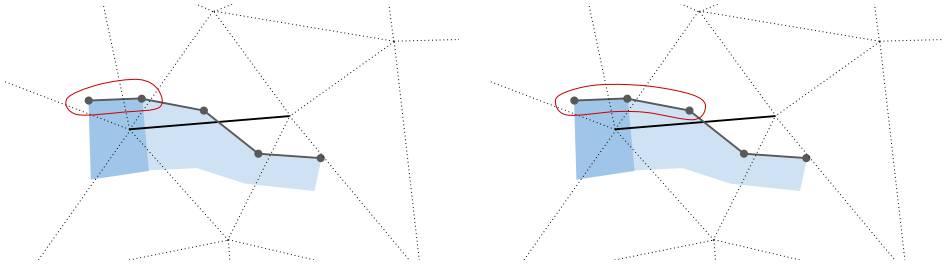
When a chain with at least one sample associated with it is removed from a coarse mesh edge, it is saved in a list of *finished chains* that are stored on the edge. It is crucial to save all chains that contain samples, no matter the situation in which they are removed, since there are always several chains being created (on multiple coarse mesh edges) referencing the same sample and if any single one of them is saved while another is not, there will be a discontinuity in the water surface. Finished chains are immutable and are only used as input to the final step of the coarse mesh construction procedure where they are combined into wave overlaps, as described in Section 4.2.9.

4.2.8 Wavefront Recording

The wavefront recording step consists of the following 6 sub-steps. Each wavefront in the simulation is handled, in order, within each sub-step.

1. Update map of all covered edges
2. Validate existing chains
3. Create new chains
4. Expand determined chains
5. Merge overlapping chains
6. Remove finished chains

See Figure 9 for an overview of the whole procedure. The remainder of this section describe the details necessary to robustly record wavefronts.



(a) Create new chain (b) Expand determined chain

(c) Create new chain (d) Expand determined chains

Figure 9: An overview of how chains are used to track a wavefront across a coarse mesh edge over two consequtive time steps.

Update map of all covered edges As the later steps are based on manipulating the chains of the coarse mesh edges, a map of all edges currently covered by any wavefront is constructed by taking the union of the Covered Mesh Edges sets of all wavefronts.

Validate existing chains In this step, the chains of all covered edges are updated so that they respect the current state of the wavefronts in the scene.

First, the Left Vertex Index and Right Vertex Index of all chains are clamped to lie within the range of the vertices of the respective wavefront. Secondly, all undetermined chains are tested to see if they qualify as determined chains. Any chain that qualifies as determined is made determined as described in Section 4.2.7. Thirdly, the covered vertex range of each determined chain is shrunk, if possible and by one vertex at a time, until all covered vertices are on the correct side of the coarse mesh edge. At last, all determined chains that do not lie on the correct side of the coarse mesh edge are removed.

Create new chains As the positional update of the wavefronts is discrete, the wavefront recording step must determine if a wavefront passes over any coarse mesh vertex as it is swept from its configuration in the previous time step to its configuration in the current time step.

Inverse bilinear interpolation [Quilez, 2010] is used to test if a segment overlaps a coarse mesh vertex. If an overlap is found, a sample is created and the coefficients s and t of the inverse bilinear interpolation are used to interpolate the values for the sample from the two time steps of the segment. Then, for each coarse mesh edge connected to the overlapped vertex, a chain referencing the new sample is created. See Figure 10 for an illustration of the creation of a chain for a single coarse mesh edge.

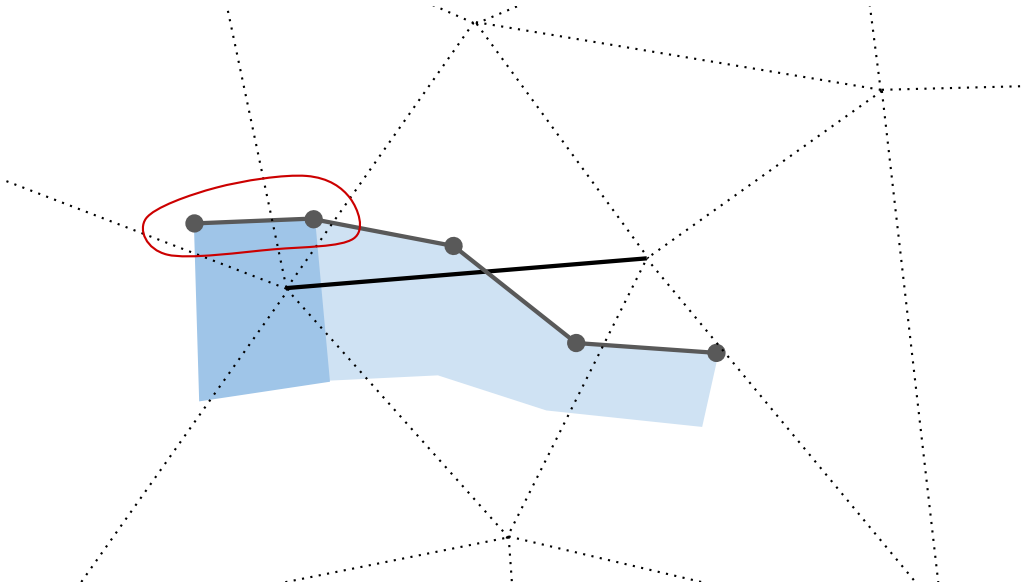


Figure 10: An illustration of the creation of a determined chain (red curve) for a single coarse mesh edge as a wavefront segment path overlaps a coarse mesh vertex.

In addition, determined chains containing no samples are created whenever wavefront vertices intersect coarse mesh edges as they are swept from their configuration in the previous time step to their configuration in the current time step. These chains are only created because they may bridge gaps between undetermined chains (that are unable to expand) in the *Merge overlapping chains* step. See Figure 11 for an illustration of a configuration where this is useful.

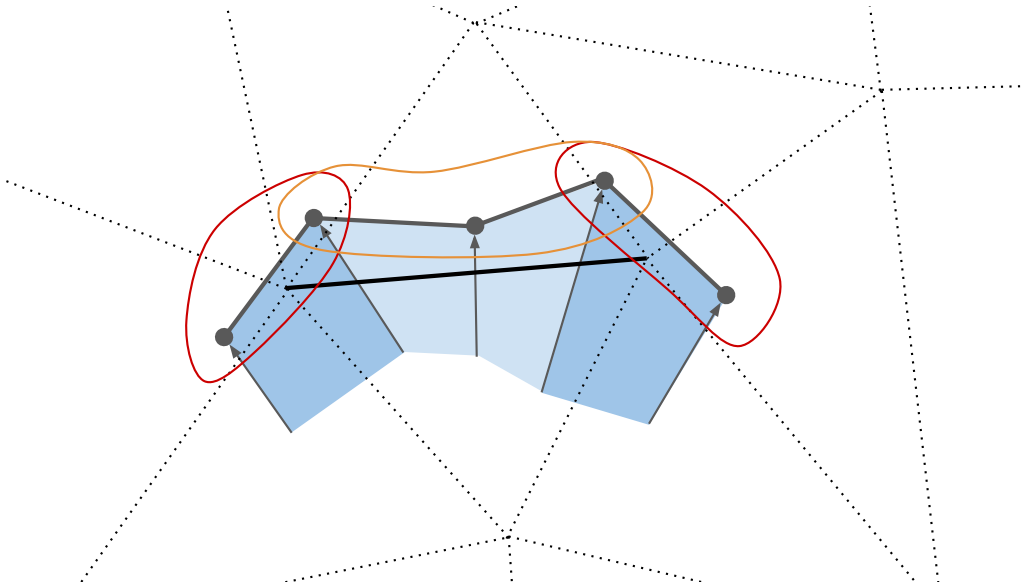


Figure 11: An illustration showing a configuration for which intersection tests between wavefront vertex paths and coarse mesh edges are useful. Two undetermined chains (red curves) are created as the two outermost wavefront segment paths overlap the two coarse mesh vertices of the edge. The undetermined chains are unable to expand and merge before being removed. In order to bridge the gap between the two chains, another chain (orange curve) is created when the paths of the 3 innermost vertices intersect the edge.

Expand determined chains In this step, the vertex coverage of determined chains are expanded to the left by decreasing the Left Vertex Index and to the right by increasing the Right Vertex Index. If a vertex that is not on the correct side of the coarse mesh edge is encountered in the direction of expansion, the expansion stops in that direction. Once the expansion in both directions have stopped, the procedure is done.

Merge overlapping chains Once a wavefront has travelled across two vertices that are joined by an edge in the coarse mesh, the edge will contain two chains, each referencing one sample. If the two samples originated from different vertices, the vertex coverage of the two chains touch (Right Vertex Index$_A$ + 1 $<$ Left Vertex Index$_B$ $\vee$ Left Vertex Index$_A$ $-$ 1 $>$ Right Vertex Index$_B$) and the wavefront does not intersect the edge along the union of the vertex coverage, the two sample values may be interpolated because the wavefront cleanly moved across the whole edge. In this case, the two the chains are merged into one chain containing the two samples. Likewise, if a chain containing no sample is merged with a chain containing one sample, the resulting chain chain will contain one sample. However, two chains with two samples each are not allowed to merge because they already encode a wavefront moving from one vertex to another and this is the limit of the linear approximation used. The merging procedure is performed for each coarse mesh edge, one wavefront at a time, and continues until it is no longer possible to

merge any two chains.

Note that two chains with different notions of "correct side" may be merged if the wavefront moves perpendicularly to the coarse mesh edge and zig-zags back and forth before leaving the edge. The notion of correct side is only used to remove chains for which the wavefront moves back the same way it came before passing another coarse mesh vertex.

When two chains A and B are merged, they are replaced with a new chain where

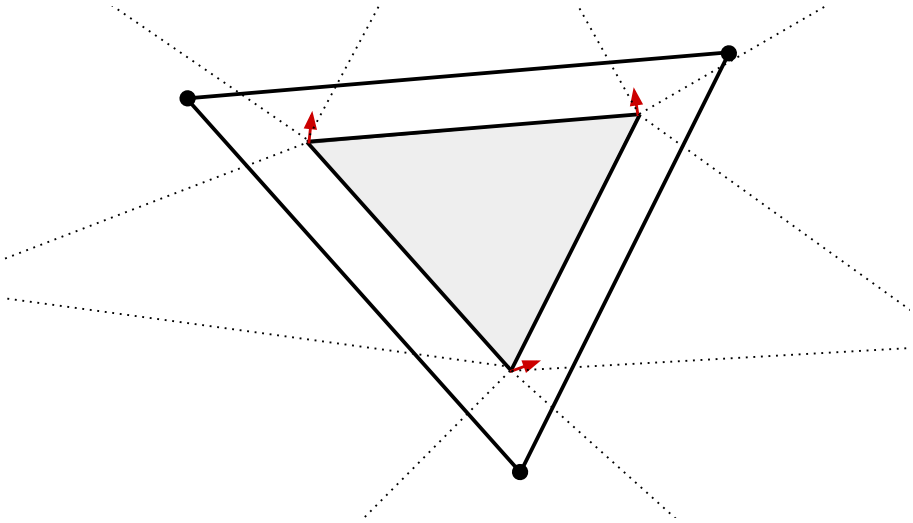
$$
\begin{aligned}
\text{Wavefront} &:= \text{Wavefront}_A \\
\text{Left Vertex Index} &:= \min(\text{Left Vertex Index}_A, \text{Left Vertex Index}_B) \\
\text{Right Vertex Index} &:= \max(\text{Right Vertex Index}_A, \text{Right Vertex Index}_B) \\
\text{Sample Count} &:= \text{Sample Count}_A + \text{Sample Count}_B \\
\text{Sample 0} &:= \begin{cases} \text{Sample 0}_A & \text{Sample Count}_A = 1 \\ \text{Sample 0}_B & \text{Sample Count}_A = 0 \wedge \\ & \text{Sample Count}_B = 1 \end{cases} \\
\text{Sample 1} &:= \begin{cases} \text{Sample 0}_B & \text{Sample Count}_A = 1 \wedge \\ & \text{Sample Count}_B = 1 \\ \text{nothing} & \text{otherwise} \end{cases} \\
\text{Side Of Edge Determined Flag} &:= \begin{cases} \text{true} & (\text{Determined}_A \wedge \text{Determined}_B \wedge \\ & \text{Side}_A = \text{Side}_B) \vee \\ & (\text{Determined}_A \wedge \neg\text{Determined}_B) \vee \\ & (\neg\text{Determined}_A \wedge \text{Determined}_B) \\ \text{false} & \text{otherwise} \end{cases} \\
\text{Side Of Edge Positive Flag} &:= \begin{cases} \text{Side}_A & \text{Determined}_A \\ \text{Side}_B & \neg\text{Determined}_A \wedge \text{Determined}_B \\ \text{false} & \text{otherwise} \end{cases}
\end{aligned}
$$

Remove finished chains In this step, any chain whose wavefront does not intersect the respective coarse mesh edge is removed.

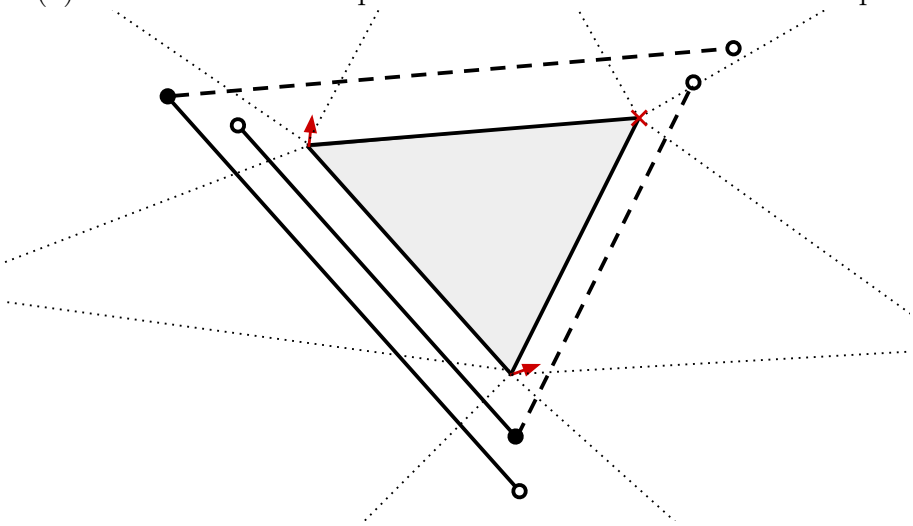
4.2.9 Combining Chains on Edges Into Wave Overlaps on Triangles

In order to be able to interpolate sample values across coarse mesh triangles, finished chains from the edges of a triangle are combined into *wave overlaps* for that triangle. This is done as single-valued functions along coarse mesh edges are combined into single-valued functions within coarse mesh triangles by Jeschke and Wojtan [2015]. The finished chains with 1 and 2 samples presented in Section 4.2.8 are equivalent to the incomplete and complete edges described by Jeschke and Wojtan [2015].

See Figure 12 for a graphical explanation of the procedure and Table 6 for what is stored in memory for each resulting wave overlap.



(a) 3 chains with 2 samples each combine into 1 wave overlap



(b) 1 chain with 2 samples and 2 chains with 1 sample each combine into 2 wave overlaps

Figure 12: Two cases of how 3 chains from 3 coarse mesh edges can be combined into wave overlaps for the coarse mesh triangle. Red arrows denote the normalized travel direction of the samples that are shared between the chains and indicate how the wavefront moved passed the vertices when the samples were created. The red x denotes a missing sample. A filled black circle indicates that the amplitude from the sample is used for that corner in the wave overlap. A hollow black circle indicates that an amplitude of zero is used for that corner in the wave overlap. A dashed line indicates that the missing sample is replaced with extrapolated values from the existing sample in the wave overlap. a) All 3 chains contain 2 samples each and reference the same 3 samples, indicating that the wavefront moved fully past the triangle. A single wave overlap is recorded. b) 1 chain with 2 samples and 2 chains with 1 sample each reference the same 2 samples, indicating that the wavefront moved passed only one edge before being removed or reflected. Two wave overlaps are recorded.

Wave Overlap	
Property	Type
Angular Frequency	Float
Phase	Float[3]
Amplitude	Float[3]
Phase Speed	Float[3]
Normalized Travel Direction	(Float, Float)[3]

Table 6: The Wave Overlap Structure

First, wave overlaps that represent wavefronts that moved fully past coarse mesh triangles are created (Figure 12a). For each coarse mesh triangle, all combinations of 3 chains with 2 samples each from the 3 edges of the triangle are found. A wave overlap is then created for each combination where the 3 chains share the same 3 samples.

Secondly, wave overlaps that represent wavefronts that partially crossed coarse mesh triangles are created (Figure 12b). For each coarse mesh triangle, all combinations of 2 chains that were not part of a combination enumerated in the first step are found. A wave overlap is then created for each combination where a sample is shared by the chains. The amplitude recorded at the shared sample gets transferred to the wave overlap whereas the amplitudes of the other corners are set to 0. Missing samples are approximated by extrapolating the existing sample.

As coarse mesh triangles share the same coarse mesh edges (and consequently chains), the generated wave overlaps form a continuous surface function with no discontinuities across the entire coarse mesh.

Sorting wave overlaps by wavelength Once all wave overlaps of a coarse mesh triangle have been created, they are sorted by wavelength in descending order. For each wave overlap, the wavelength of the wave is determined at each triangle vertex i, $\lambda_i = 2\pi c_i/\omega$, and the minimum of the three values is used for the sorting comparison. Since the wavelength of the wave can not be more extreme inside the triangle than at any of its vertices when linear interpolation is employed, the minimum value gives a good measurement of the shortest wavelength across the entire wave overlap.

4.3 Rendering

The next section will describe the rendering step and how the simulation data stored in the coarse mesh is used to displace and render the water surface. The focus will be on achieving the best possible performance on current generation graphics hardware.

In order to render the water surface using the graphics card (GPU), the surface must be decomposed into a set of triangles. The number of triangles pushed to the GPU will partly determine the performance of the rendering step and excessive use of triangles will leave little computational resources to the rest of the virtual world. In a production environment, the water rendering subsystem cannot use a majority of the typical 16 ms time budget designated to rendering a single frame (with a 60 Hz update frequency target). Thus, care must be taken when constructing the water surface geometry so that only visible triangles are pushed to the GPU. In addition, the screen space size of the triangles should be approximately constant across the water surface so that the image quality is uniform across the screen and so that no single area of the screen is a computational bottleneck.

The coarse mesh constructed in the simulation step might seem to be a natural choice for rendering the water surface. For simplicity and compatibility with the previous water rendering algorithm used by Frostbite [2017], a world space aligned quadtree was used instead. See Section 7.1 for a discussion on the subject.

4.3.1 Water Surface Geometry

In order to construct the vertices and triangles of the water surface geometry, a world space aligned quadtree is adaptively subdivided with respect to the view frustum every frame. Each leaf node in the tree contains a grid of 16^2 cells ($17^2 = 289$ vertices, $2 \cdot 16^2 = 512$ triangles) that are rendered using the unique world-space transform of the leaf node. As the tree is adaptively subdivided, different leaf nodes will have different scales and the resolution of the water surface geometry will vary spatially in world space. See Figure 13 for an abstract illustration of the tree generation.

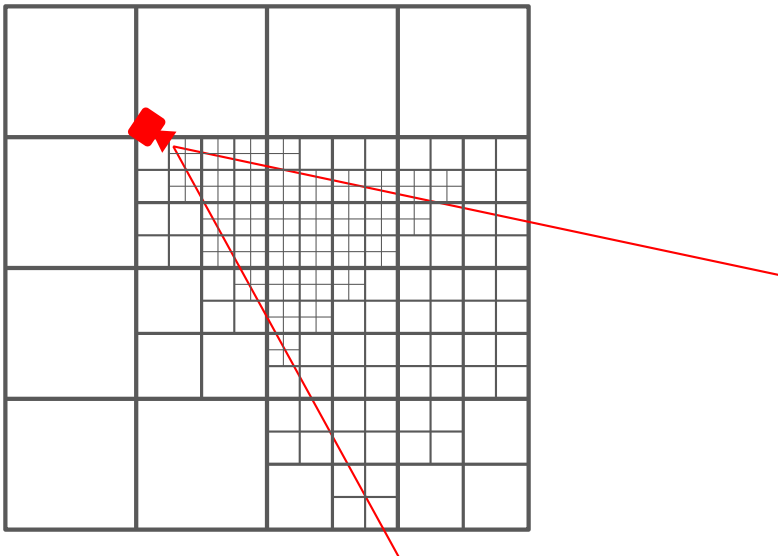


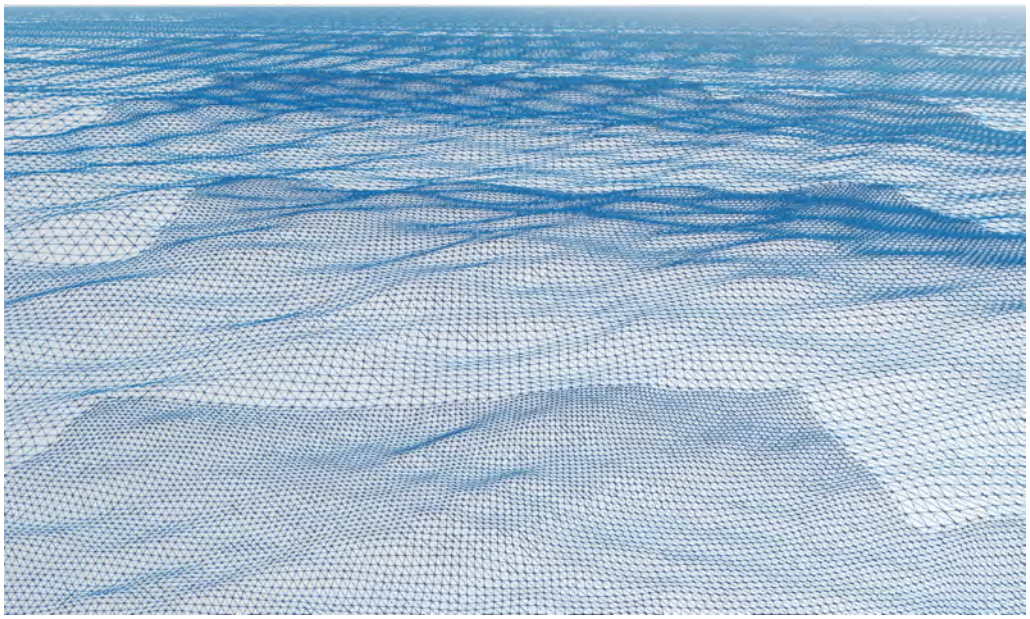
Figure 13: An illustration of the world space aligned quadtree, as seen from above, that is used to generate the water surface geometry. The view frustum is shown in red and the leaf nodes of the quadtree as gray boxes. Note that there are leaf nodes at different spatial scales which gives rise to spatially varying resolution of the water surface.

A node in the tree is subdivided if the world space size of a triangle is larger than the approximate world space size, in the horizontal or vertical direction, of r_{min} pixels at the same distance from the camera. In practice, $r_{\text{min}} = 4$ pixels was used which yields surface triangles no larger than 4 pixels in screen space. When no node of the quadtree can be subdivided further, the resulting vertices and triangles of the leaf nodes form a flat, adaptively tessellated, water surface across the virtual scene.

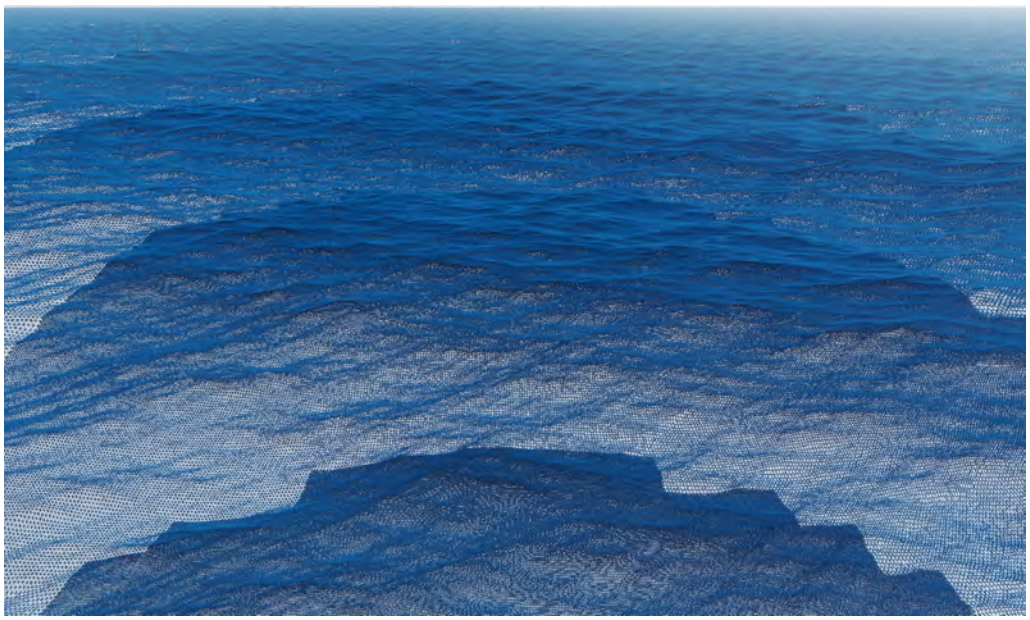
The world space aligned quadtree was chosen over a projected grid [Johanson and Lejdfors, 2004] approach because of its inherent spatial persistency. In a world space quadtree, it is probable that a vertex on the water surface exists in multiple subsequent frames of an animation, even if the view frustum is moving, and this coherency can be used to optimize the height-lookup of the water surface.

In contrast, when using a projected grid, the positions of vertices projected onto the water surface are very sensitive to movement of the view frustum and there is no coherency in world space vertex positions between frames. Furthermore, if the projected grid is not very finely tessellated, movement of the view frustum can cause rolling motions on the water surface as the sampling points move in world space, even if the Nyquist limit is respected.

See Figure 14 for a wireframe view of the resulting water surface.



(a) $r_{\min} = 16$



(b) $r_{\min} = 4$

Figure 14: A wireframe view of the generated water surface. The structure of the world space aligned quadtree can be seen where there is a resolution difference between adjacent leaf nodes, i.e., where leaf nodes at different tree depths are adjacent. Note that the vertical component of the screen space triangle size is very small in the distance due to the square node-based subdivision scheme employed. See Section 6 for a discussion on the subject.

4.3.2 Evaluating The Simulation

The next step is to evaluate the simulation by offsetting the vertices on the water surface according to the wave overlaps stored on the coarse mesh triangles. Computing the vertex offset of a single vertex according to the simulation is denoted a *height-lookup*. Once the entire water surface has been evaluated for a given animation frame, the water surface can be rendered.

The simplest way of evaluating the simulation for a given animation frame is to perform height-lookups for all vertices in all leaf nodes of the quadtree at once. However, because of the high number of total vertices on the water surface and the high number of wave overlaps required to reach the desired level of quality, this update scheme is too performance intensive to be used in practice on current graphics hardware.

Staggered update scheme To improve performance, the vertices of the water surface are evaluated using a staggered update scheme at a low update frequency. A staggered update scheme reduces the number of vertices that need to be updated every frame by evaluating different vertices every frame according to some predefined pattern.

As the simulation data stored in the coarse mesh may be evaluated at any point in time for any world space position independently of previous state, a height-lookup can be performed for any vertex independently of any other vertex on the water surface. This makes it possible to evaluate vertices in a staggered fashion. Furthermore, if the update frequency of the water is lower than the update frequency of other aspects of the world, for example the physics engine, it is possible evaluate each vertex at two points in time and then linearly interpolate between the two height values in order to fill in the missing frames and give the impression that the entire virtual world is updated at the same rate.

Vertex displacement buffer In order to implement a staggered update scheme, the offset values computed for a set of vertices during a frame must be accessible in subsequent frames when other vertices are evaluated. A persistent vertex displacement buffer is used to keep track of the offsets of all vertices on the water surface. A leaf node that exists for several frames occupies a fixed set of 289 elements, denoted a slot, in the vertex displacement buffer.

After the quadtree is constructed for a given frame, the leaf nodes of the quadtree are iterated and compared against the leaf nodes of the quadtree of the previous frame. If a new leaf node that did not exist in the previous frame is encountered, it is assigned a free slot in the vertex displacement buffer. If a leaf node that existed in the previous frame no longer exists in the current frame, its slot in the vertex displacement buffer is freed and the data stored in the slot remains unused and stagnant until the slot is assigned to a new leaf node.

In addition, the dynamic buffers required to evaluate and render the water surface are filled as the list of leaf nodes are iterated.

Render buffer pool Since the number of leaf nodes in the quadtree varies on a frame-by-frame basis, a render buffer pool is used to better store the data needed for the visible leaf nodes on any given frame. The render buffer pool consists of a list of groups, each with its own set of render buffers capable of representing a number of leaf nodes. In practice, each group is able to accommodate 256 leaf nodes. When a new leaf node is encountered and there are no free slots available in any existing group, a new group is created and added to the render buffer pool and the new leaf node is assigned to the first slot in the new group. In a similar fashion, the list of groups may be defragmented and trimmed if the slot capacity has been significantly higher than the average number of leaf nodes for a number of subsequent frames. Using this render buffer pool, the maximum graphics memory wasted is bounded to the space occupied by 256 leaf nodes.

Render buffer group Each render buffer group contains one vertex displacement buffer, one slot index buffer, one position buffer, one stride node index buffer and one full node index buffer. The vertex displacement buffer is persistent across frames whereas the others are dynamically constructed every frame. See Figure 15 for a schematic view of a single render buffer group and an illustration of how the indices of one buffer reference elements of another.

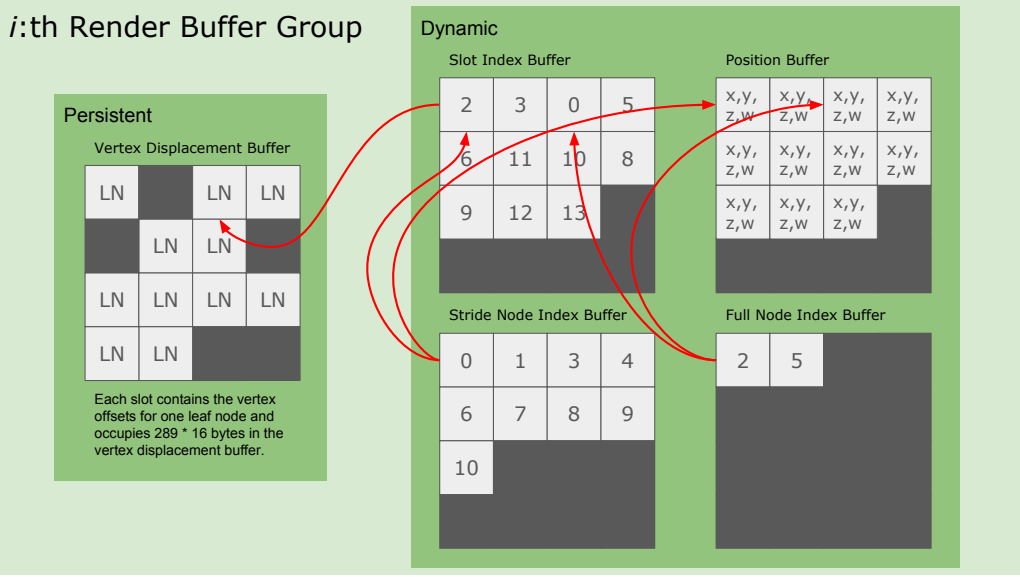


Figure 15: A schematic view of the buffers in one group of the render buffer pool and the internal references present.

The slot index buffer and position buffer encodes the leaf nodes assigned to the group. Each element of the slot index buffer references which slot in the vertex displacement buffer to use for the given leaf node and the corresponding element of the position buffer encodes the transform of the given leaf node. In addition to the vertex displacement buffer, these are the only buffers needed to render the water surface.

The stride node index buffer and full node index buffer are used when the water surface is evaluated. Since not all vertices of a slot are evaluated during a frame in

the staggered update scheme, the slots of new leaf nodes must be fully evaluated when they first appear in order for them not to contain any stagnant data from any previously assigned and now no longer visible leaf node. This is not a problem for a leaf node that exists over several frames since they will, by design, keep the same slot in the vertex displacement buffer throughout all frames. The full node index buffer is constructed in such a way that each element references an index in the slot index buffer and position buffer corresponding to a new leaf node and the stride node index buffer is constructed in such a way that each element references an index corresponding to a leaf node that have existed for at least one frame.

Compute shader jobs Once the quadtree has been constructed and the render buffer pool has been updated, a compute shader job is dispatched for each render buffer group to evaluate its vertex displacement buffer. Once all compute shader jobs are complete, all visible parts of the water surface have been evaluated and the water surface as a whole is ready to be rendered.

In addition to all render buffers of the group, the compute shader job takes two integers as input: the stride count and the stride index. The stride count denotes the number of stride groups that the vertices of a slot should be divided into. A stride group is a set of vertices that are all evaluated at the same time. The stride index denotes which such stride group should be evaluated in the given compute shader job dispatch.

In order to reduce aliasing artifacts on the water surface, stride groups are chosen in such a way that the vertices are evaluated in wave-like patterns. Let i be the local vertex index, $0 \leq i < 289$, of a slot in the vertex displacement buffer. Then, vertex i belongs to stride group j if and only if the remainder after dividing i by the stride count is j. Let n denote the number of vertices along one side of a leaf node. When n is odd, vertices of the same stagger group, that belong to different leaf nodes at the same scale, will be aligned in world space as long as the stagger count is even and less than n. See Figure 16 for an illustration of the visual patterns the stride groups give rise to in world space.

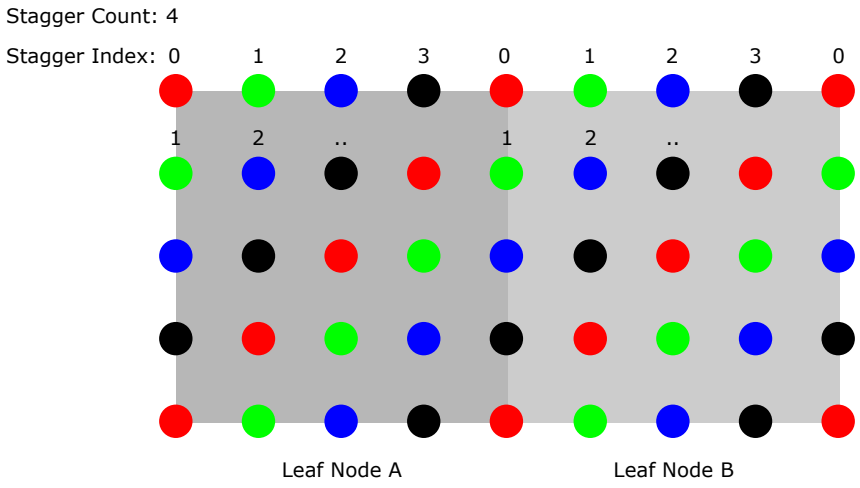


Figure 16: An illustration of two adjacent leaf nodes and the visual patterns their stride groups give rise to in world space.

The time between compute shader job dispatches for a given stride index, denoted the update interval, decides the times at which to evaluate the water surface at any given point. The surface is evaluated once at the nearest multiple of the update interval and the current time, offset by the stride index, backward in time and once at the nearest multiple of the update interval and the current time, offset by the stride index, forward in time. See Figure 17 for an schematic view of the timing schedule.

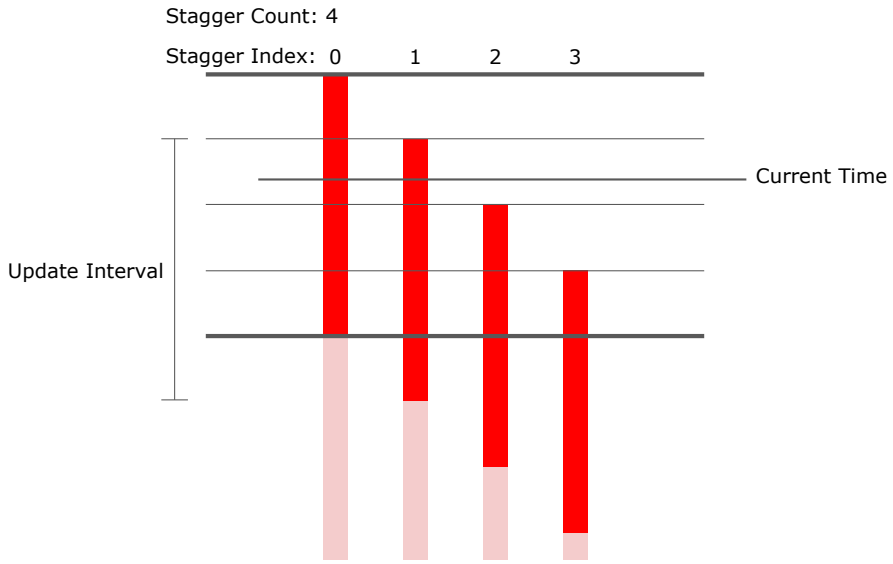


Figure 17: A schematic view of the timing schedule employed in the staggered update scheme. The red bars denote the time interval dedicated to a given stride index. At the start of a time interval, height-lookups are performed for all vertices in the given stride group determining the vertex offsets at the beginning of the time interval and at the end of the time interval. During the time interval, the two offset values for each vertex in the stride group are linearly interpolated.

Update frequency When using the staggered update scheme, one may trigger compute shader job dispatches every frame, enumerating the possible stride index values one by one. However, this does not lend itself to a variable frame rate since previous evaluations assume that the next dispatch happens at a specific point in time. In practice, the frame rate will vary and the update scheme must be decoupled from the normal rendering loop.

Thus, the water simulation is run at an update frequency f that is lower than that of the rest of the virtual world. The update interval is then fixed to $1/f$ seconds.

In practice, the update frequency of the water simulation is set to $f = 8\,\mathrm{Hz}$ using a stagger count of 8. This was the perceived lower limit of the staggered update scheme as a lower update frequency caused the inherent visual artifacts to become apparent.

4.3.3 The Height-Lookup

The compute shader job evaluates the water surface at all vertices in a given stagger group using Equation 1. For each vertex, the geometric offset and surface normal of the water surface are computed for the two time points given by the staggered update scheme.

First, the coarse mesh triangle that contains the vertex is found. Then, the wave overlaps of the found coarse mesh triangle are iterated. For each wave overlap, the wave parameter values are interpolated to the vertex and the contribution from the wave overlap to the geometric offset and surface normal are computed for the two

time points using the wave profile. Finally, the contributions from all wave overlaps are accumulated and the final geometric offset and surface normal for the vertex is written to the vertex displacement buffer.

In order to avoid aliasing due to the Nyquist limit, each contribution is attenuated according to the wavelength of the wave overlap. Since the wave overlaps are sorted by wavelength in descending order, the iteration terminates whenever the contribution of a wave overlap is zero due to attenuation. This scheme improves performance as it limits the compute shader execution time for coarse mesh triangles that are far away from the camera.

Coarse mesh triangle lookup A top-down 2-dimensional grid implemented as an index buffer is used to accelerate the lookup of coarse mesh triangles in the compute shader. The grid covers the water surface and each grid cell contains a reference to the first element of a linked list of coarse mesh triangles intersecting that cell. All linked lists are packed into a single 2-value index buffer where the first value references a coarse mesh triangle and the second value the next element in the linked list.

Thus, in order to find the coarse mesh triangle containing a 2-dimensional point $\vec{p}$ on the water surface, the grid cell i containing $\vec{p}$ is first found. Then, the grid index buffer is queried at i to get the index j referencing an element in the linked list index buffer. The linked list index buffer is subsequently queried at j for the coarse mesh triangle t_j and the next index n_j. If t_j contains $\vec{p}$ in the plane, the procedure returns t_j. Otherwise, the next element in the linked list is examined iteratively by setting $j := n_j$. Finally, if an n_j that does not point to another element is encountered, the end of the linked list has been reached and $\vec{p}$ must lie outside the bounds of the water surface and the procedure returns nothing.

In practice, a constant grid size of 1024^2 was used. This results in a grid cell side length of approximately 4 m in a 16 km^2 square-shaped simulation.

Wave filtering In order to eliminate aliasing on the water surface, the Nyquist limit must be respected when waves are sampled. Thus, the contribution of a wave overlap to the geometric offset is attenuated and filtered as described by Bruneton et al. [2010].

The sample size is the maximum world space distance between two vertices on the water surface and the exact value can be determined from the leaf node scale in the position buffer. However, if the exact value is used there will be visible discontinuities on the water surface where leaf nodes at different depths in the quadtree are adjacent. In order to provide a seamless transition between leaf nodes, the approximate world space sample size, s, used when filtering is determined by

$$s = \sqrt{2}\min(s_x, s_y),$$

where

$$
\begin{aligned}
s_x &= \frac{c_x|\vec{v}|}{|\max(\vec{v}_x/|\vec{v}|, \vec{v}_z/|\vec{v}|)|} \\
s_y &= \frac{c_y|\vec{v}|}{|(\vec{v}_y/|\vec{v}|)|} \\
c_x &= 2\tan(\text{fov}_x/2)\frac{r_{\min}}{w} \\
c_x &= 2\tan(\text{fov}_y/2)\frac{r_{\min}}{h}
\end{aligned}
$$

and $\vec{v}$ is the vector from the vertex to the camera in world space, fov_x the horizontal field of view, fov_y the vertical field of view, w the screen width in pixels, h the screen height in pixels and $r_{\min}$ the desired screen space extent of a triangle (in pixels) on the water surface as described in Section 4.3.1.

The wavelength used when filtering is the minimum wavelength value stored on the wave overlap as described in Section 4.2.9.

Wave parameter interpolation In order to evaluate the wave function (Equation 1) at any point within a coarse mesh triangle, the wave parameter values stored in each wave overlap are interpolated as shown by Jeschke and Wojtan [2015].

First, the barycentric coordinates, $\vec{b}$, of a point $\vec{p}$ are computed with respect to the corner points of the triangle as shown by Ericson [2004]. Then, when the wave overlaps of the triangle are iterated, the three corner values of each wave parameter are linearly interpolated yielding a value for each wave parameter at $\vec{p}$:

$$
\begin{aligned}
\hat{\phi} &= \begin{pmatrix}\phi_0 & \phi_1 & \phi_2\end{pmatrix}^T \cdot \vec{b} \\
\hat{a} &= \begin{pmatrix}a_0 & a_1 & a_2\end{pmatrix}^T \cdot \vec{b} \\
\hat{c} &= \begin{pmatrix}c_0 & c_1 & c_2\end{pmatrix}^T \cdot \vec{b} \\
\hat{\vec{d}} &= \begin{bmatrix}\vec{d}_0 & \vec{d}_1 & \vec{d}_2\end{bmatrix}\vec{b} \Big/ \left|\begin{bmatrix}\vec{d}_0 & \vec{d}_1 & \vec{d}_2\end{bmatrix}\vec{b}\right|
\end{aligned}
$$

where $\cdot$ denotes the dot product.

Side-vertex interpolation [Jeschke and Wojtan, 2015] was also implemented but not used due to the increased instruction count compared to linear interpolation.

Wave profile A trochoidal wave profile is used to offset the water surface in both the vertical and horizontal directions as shown by Finch [2004] and Bruneton et al. [2010]. In contrast to their work, however, waves do not travel at constant phase speeds and the phase function must be utilized. Given a world space point, $(x\,y\,z)^T$, on the water surface, the offset point according to the wave profile, $\vec{p}$, is then:

$$
\vec{p} = \begin{pmatrix}x\\y\\z\end{pmatrix} + \sum_{i=1}^{N} \begin{pmatrix}\hat{d}_{ix}\hat{a}_i\cos(\omega_i\hat{\phi}_i - \omega_i t)\\ \hat{a}_i\sin(\omega_i\hat{\phi}_i - \omega_i t)\\ \hat{d}_{iy}\hat{a}_i\cos(\omega_i\hat{\phi}_i - \omega_i t)\end{pmatrix},
$$

where ω_i the angular frequency of wave overlap i and $\hat{\vec{d}}_i$ the interpolated direction, $\hat{a}_i$ the interpolated amplitude and $\hat{\phi}_i$ the interpolated phase of wave overlap i at $(x\,y\,z)^T$. In order to calculate the surface normal at $\vec{p}$, the partial derivatives of $\vec{p}$ is first calculated in the x and z directions, resulting in the surface tangent, $\vec{t}$, and bitangent, $\vec{b}$, respectively:

$$\begin{aligned}
\vec{t} &= \frac{\partial}{\partial x}\vec{p} \\
&= \begin{pmatrix}1\\0\\0\end{pmatrix} + \sum_{i=1}^{N}\left[\begin{pmatrix}-\hat{d}_{ix}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\\ \hat{a}_i\cos(\omega_i\hat{\phi}_i-\omega_i t)\\ -\hat{d}_{iy}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\end{pmatrix}\omega_i\frac{\partial}{\partial x}\hat{\phi}_i\right] \\
&= \begin{pmatrix}1\\0\\0\end{pmatrix} + \sum_{i=1}^{N}\left[\begin{pmatrix}-\hat{d}_{ix}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\\ \hat{a}_i\cos(\omega_i\hat{\phi}_i-\omega_i t)\\ -\hat{d}_{iy}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\end{pmatrix}\omega_i\frac{\hat{d}_{ix}}{\hat{c}_i}\right]
\end{aligned}$$

$$\begin{aligned}
\vec{b} &= \frac{\partial}{\partial z}\vec{p} \\
&= \begin{pmatrix}0\\0\\1\end{pmatrix} + \sum_{i=1}^{N}\left[\begin{pmatrix}-\hat{d}_{ix}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\\ \hat{a}_i\cos(\omega_i\hat{\phi}_i-\omega_i t)\\ -\hat{d}_{iy}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\end{pmatrix}\omega_i\frac{\partial}{\partial z}\hat{\phi}_i\right] \\
&= \begin{pmatrix}0\\0\\1\end{pmatrix} + \sum_{i=1}^{N}\left[\begin{pmatrix}-\hat{d}_{ix}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\\ \hat{a}_i\cos(\omega_i\hat{\phi}_i-\omega_i t)\\ -\hat{d}_{iy}\hat{a}_i\sin(\omega_i\hat{\phi}_i-\omega_i t)\end{pmatrix}\omega_i\frac{\hat{d}_{iy}}{\hat{c}_i}\right],
\end{aligned}$$

where $\hat{c}_i$ is the interpolated phase speed of wave overlap i at $(x\,y\,z)^T$.
The surface normal, $\vec{n}$, can then be calculated using the cross product:

$$\vec{n} = \frac{\vec{b}\times\vec{t}}{|\vec{b}\times\vec{t}|}.$$

5 Results

The algorithm was tested in two different virtual scenes: an archipelago was used for testing performance and a depiction of the Mediterranean coastline was used to assess visual quality in a typical production environment.

The archipelago scene was used to test performance because it provided the largest possible area for water simulation and only contains terrain in addition to the water. The Mediterranean coastline scene was used to assess visual quality because it closely represents a production quality scene, containing photo-realistic terrain, textures and foliage in addition to the water.

5.1 Performance

The performance of the algorithm was assessed using a virtual scene consisting of a group of islands [DICE, 2017] and water covering an area of $16\,\mathrm{km}^2$. The focus area of the simulation was set to $1.2\,\mathrm{km}^2$ and centered around a set of islands in the center of the scene. Two wavefront generators where used to simulate a $5\,\mathrm{m/s}$ wind from the North and a $10\,\mathrm{m/s}$ wind from the South. The resulting seascapes collide at open sea to the East and West of the focus area and produce violent sea conditions in these areas. The islands in the focus area shelter the immediate surroundings to the North and South from the waves coming from the opposite direction and moderate sea conditions representative of the wind speed of the respective wavefront generator can be observed. In between the islands, the water is sheltered from all sides and the surface is nearly undisturbed. See Figure 18 for an overhead view of the scene and Table 7 for a summary of the simulation parameters used and output of the algorithm.

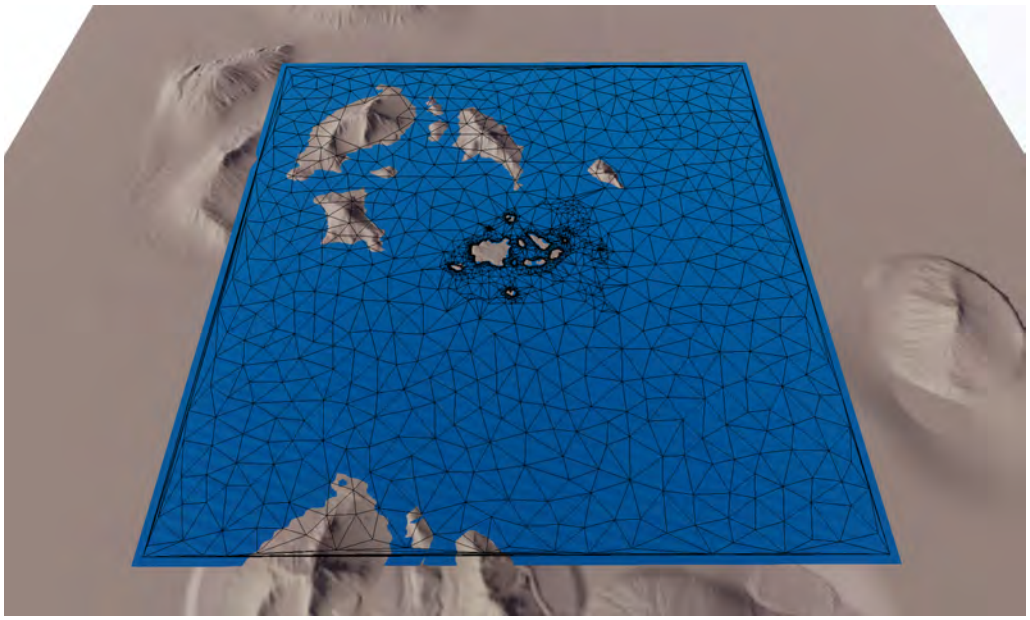
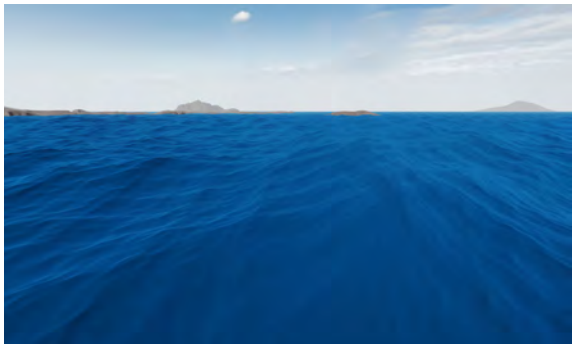


Figure 18: The virtual scene used for performance measurements. The water plane covers an area of $16\,\mathrm{km}^2$ (coarse black triangles) with a focus area of $1.2\,\mathrm{km}^2$ (dense black triangles).

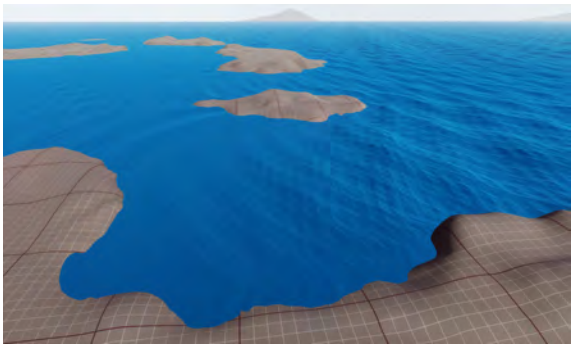
Simulation specifications	
Total area	$16\,\mathrm{km}^2$
Focus area	$1.2\,\mathrm{km}^2$
Wavefront generators	2
Wavefronts simulated	40
Min Amplitude/Wavelength Ratio	0.005
Max Amplitude/Wavelength Ratio	0.07
Simulation time	164 minutes
Coarse mesh triangles	8408
Max wave overlaps per triangle	146
Average wave overlaps per triangle	34.2
GPU memory occupied by simulation	23 megabytes

Table 7: A summary of the simulation parameters used and the output of the algorithm.

Two hardware configurations were used to test the algorithm: a personal computer with an Intel Xeon E5-1650 processor and a Nvidia GeForce GTX 970 graphics card and a Sony PlayStation 4. Performance measurements were recorded from 6 different viewpoints in the scene, with and without the staggered update scheme enabled. The frame buffer resolution was 1920×1147 on the personal computer and 1600×900 on the PlayStation 4. See Figure 19 for screenshots of the 6 different viewpoints.

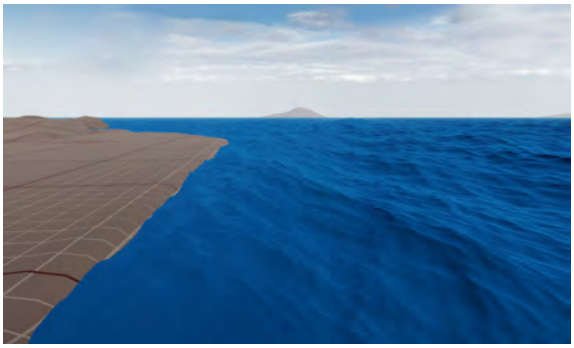


(a) Open sea South of island cluster

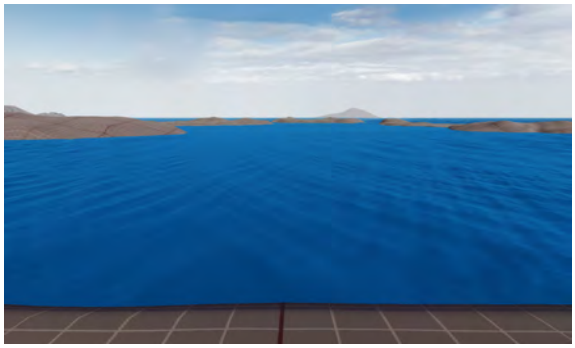


(b) Southern inlet of island cluster

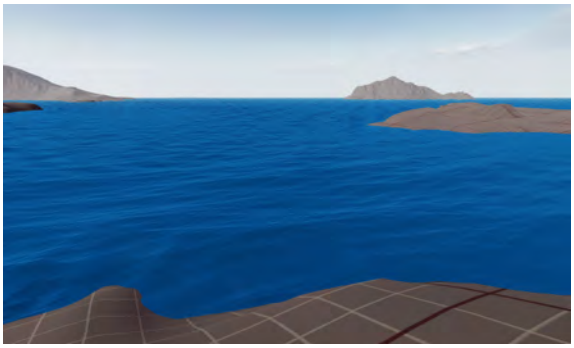
(c) Violent sea East of island cluster



(d) Southern shoreline of South island



(e) Calm water in between islands



(f) Northern shoreline of North island

Figure 19: Screenshots of the six viewpoints at which performance measurements were recorded.

The total water surface evaluation time for a given viewpoint can be seen in Table 8. This corresponds to the total time consumed by the compute shader job dispatches on the GPU. The coarse mesh triangle lookup time can be seen in Table 9 and corresponds to the time it takes to find the coarse mesh triangle of interest for all vertices of a given stagger group in the vertex displacement buffer on the GPU, when using the staggered update scheme. Both of these measurements correspond to the part of the algorithm that must be executed independently of the number of screen pixels the water occupies in a rendered frame.

It is clear from these measurements that most time is typically spent accessing the wave data of the coarse mesh triangle as opposed to looking up the triangle itself.

However, the coarse mesh triangle lookup time is not an insignificant fraction of the total and can sometimes amount to approximately half of the time spent.

Total water surface evaluation time (GPU)				
	GeForce GTX 970		PlayStation 4	
Viewpoint	Full	Staggered (avg)	Full	Staggered (avg)
A	2.482	0.41 (0.162)	2.997	0.76 (0.833)
B	1.252	0.26 (0.113)	1.537	0.61 (0.667)
C	2.629	0.43 (0.182)	2.952	1.25 (1.187)
D	2.152	0.38 (0.164)	2.737	1.26 (1.179)
E	1.813	0.33 (0.135)	2.146	0.65 (0.697)
F	1.855	0.35 (0.147)	2.184	0.73 (0.794)

Table 8: Total water surface evaluation time on the GPU in milliseconds. Parenthesis denote averages over all rendered frames, not only frames where a stagger update occurs. The average evaluation time is considerably higher on the PlayStation 4 because it runs at a lower frame rate and a larger fraction of the frames trigger water surface evaluations. Note that the evaluation time of the staggered update scheme is close to the expected theoretical performance gain of 8 times, with a stagger count of 8.

Coarse mesh triangle lookup time (GPU)		
	GeForce GTX 970	PlayStation 4
Viewpoint	Staggered (avg)	Staggered (avg)
A	0.13 (0.053)	0.25 (0.264)
B	0.13 (0.053)	0.24 (0.256)
C	0.14 (0.059)	0.28 (0.299)
D	0.15 (0.060)	0.27 (0.290)
E	0.15 (0.062)	0.28 (0.301)
F	0.15 (0.060)	0.27 (0.291)

Table 9: Coarse mesh triangle look-up time on the GPU in milliseconds. Parenthesis denote averages over all rendered frames, not only frames where a stagger update occurs. The average evaluation time is considerably higher on the PlayStation 4 because it runs at a lower frame rate and a larger fraction of the frames trigger water surface evaluations.

The GBuffer rendering time for a given viewpoint can be seen in Table 10. This corresponds to the time it takes for the GPU to rasterize all triangles of the water surface into the GBuffers used by the deferred renderer. The time required varies depending on the the previous contents of the ZBuffer. In particular, if the virtual scene contains elements in the foreground that occlude the water surface, the performance impact of rendering the water surface will decrease.

GBuffer rendering time (GPU)		
Viewpoint	GeForce GTX 970	PlayStation 4
A	2.103	3.749
B	1.895	2.774
C	2.111	3.474
D	2.046	3.409
E	1.926	3.615
F	1.777	2.497

Table 10: GBuffer rendering time on the GPU in milliseconds

The Quadtree & buffer construction time for a given viewpoint can be seen in Table 11. This corresponds to the time it takes to build the quadtree and fill the contents of the dynamic render buffers every frame for one core of the CPU. The job is scheduled to be executed alongside other independent jobs in order to maximize processor utilization when multiple cores are available.

Quadtree & render buffer construction time (CPU)		
Viewpoint	Intel Xeon E5-1650	PlayStation 4
A	1.941	3.434
B	1.522	2.558
C	1.883	3.115
D	1.927	3.348
E	1.923	3.413
F	1.869	3.160

Table 11: Quadtree & render buffer construction time on the CPU in milliseconds

5.2 Achieved Visual Quality

The visual quality of frames generated by the algorithm was assessed using a virtual scene depicting the Mediterranean coast [DICE, 2017]. One wavefront generator was used and the wind direction was set to face the coastline.

5.2.1 Overview

See Figure 20 for screenshots of six viewpoints representative of the visual quality that can be achieved when using the algorithm in a typical production environment. In order to see the wave patterns produced on a grand scale, Figure 21 shows a series of screenshots from a top-down perspective as the altitude increases.

(a) Open sea

(b) Violent sea near coastal cliffs

(c) Waves focus in inlet

(d) Waves refract around natural jetty

(e) Waves refract around island

(f) Waves lining up with bay shoreline

Figure 20: Screenshots of the six viewpoints that are representative of the image quality of the algorithm when used in a production environment.

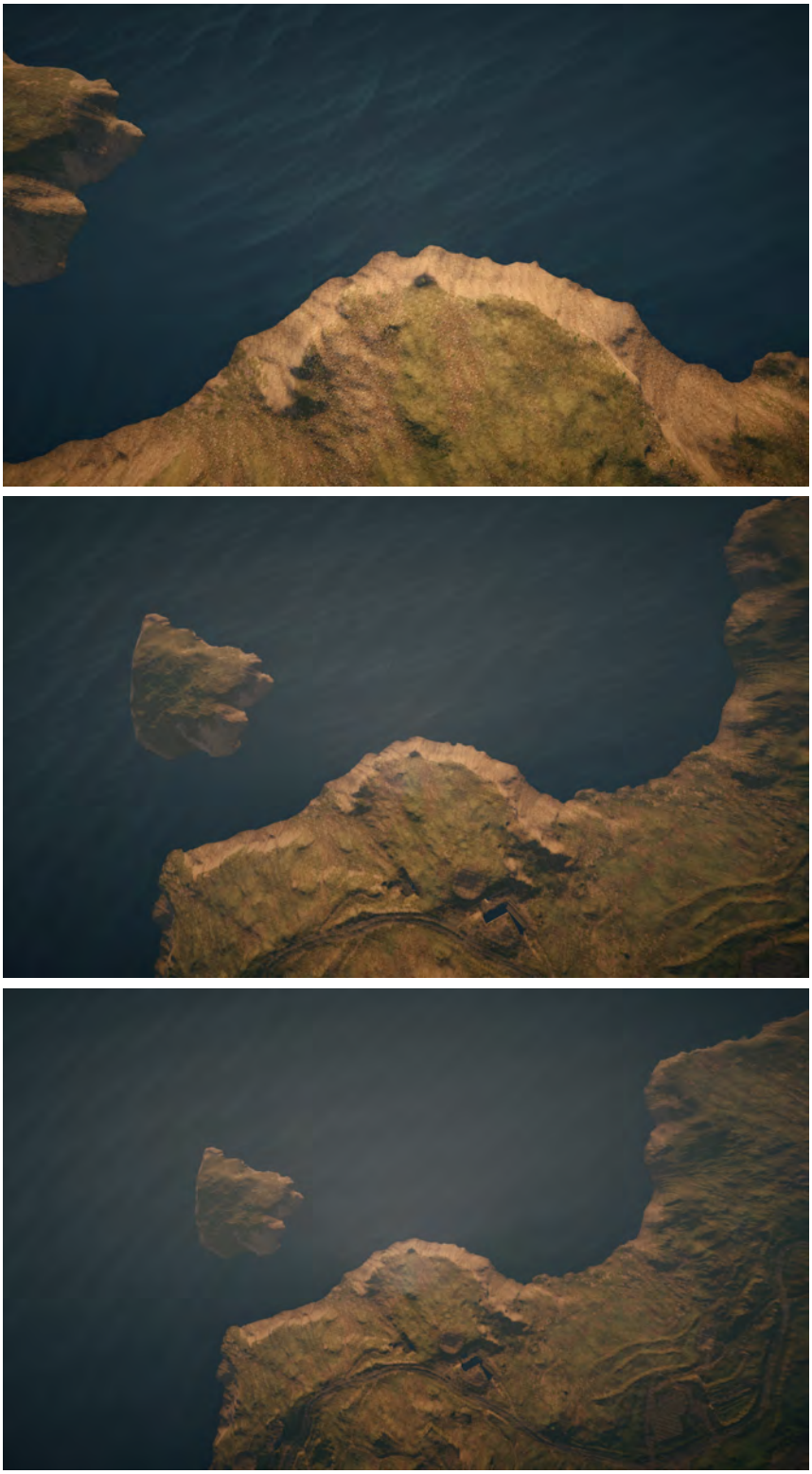


Figure 21: Screenshots from a top-down view as the altitude increases

In practice, there is no visible aliasing. The employed wave filtering effectively eliminates spatial aliasing and even though a low update frequency is used for the staggered update scheme, no temporal aliasing was observed.

Repetition patterns can be observed in areas close to the boundaries of the simulation where wavefronts have not yet started to diverge due to changes in depth. However, note that as each coarse mesh triangle can be visually unique, it is possible to eliminate repetition patterns in theory even close to the boundaries of the simulation.

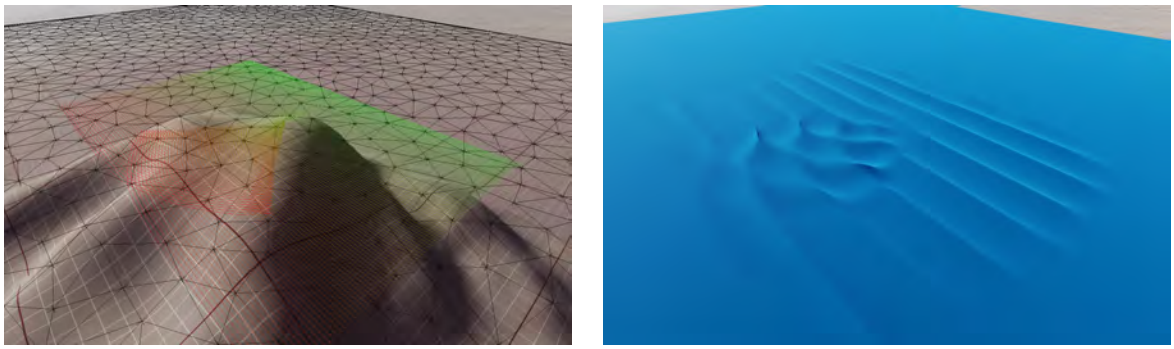
In theory, cracks in the water surface geometry are expected in places where leaf nodes at different depths in the world space quadtree are adjacent since stagger group indices only line up at the borders of adjacent leaf nodes that have the same world space size. In practice, no visible cracks in the water surface can be observed.

Visible seams between coarse mesh triangles can be observed if a high frequency wave is visible in one triangle but gets filtered in an adjacent triangle due to inadequate sampling resolution. The reason for this behavior is that the minimum wavelength value of the three corners of a wave overlap is used when filtering waves as described in Section 4.3.3 and the wave is not attenuated within the coarse mesh triangle in which it is visible.

5.2.2 Captured Water Wave Behaviors

The following section illustrates specific wave behaviors that are captured by the simulation.

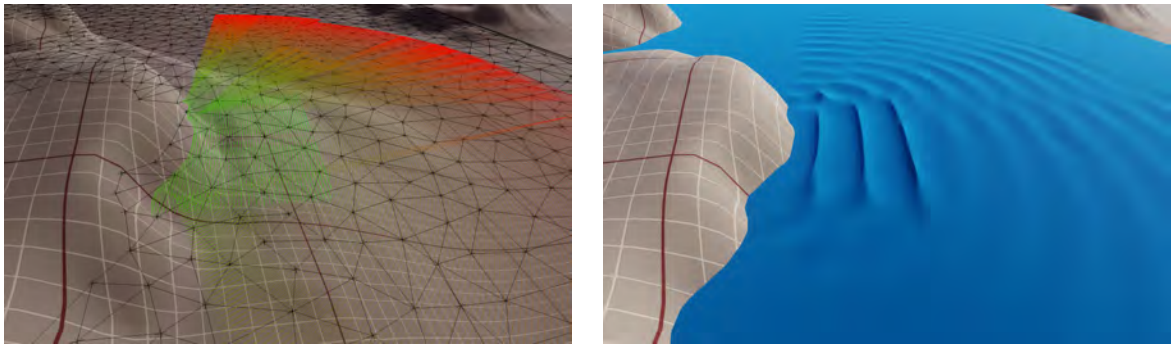
Refraction A water wave refracts as it moves from one depth to another and points on the wave travel at different phase speeds. Within the simulation, a wavefront refracts if the change in the underlying depth field is significant. See Figure 22 for a view of a single wavefront developing a swallowtail pattern as it enters a local area of shallow water.



(a) Simulation step (b) Rendering step

Figure 22: A wavefront refracts to produce a swallowtail pattern as it enters a local shallow area of the seafloor. a) Multiple time steps of the wavefront are shown transitioning from green to red with increasing phase. The coarse triangle mesh is shown in black. The seafloor is shown in gray. b) The rendered water surface.

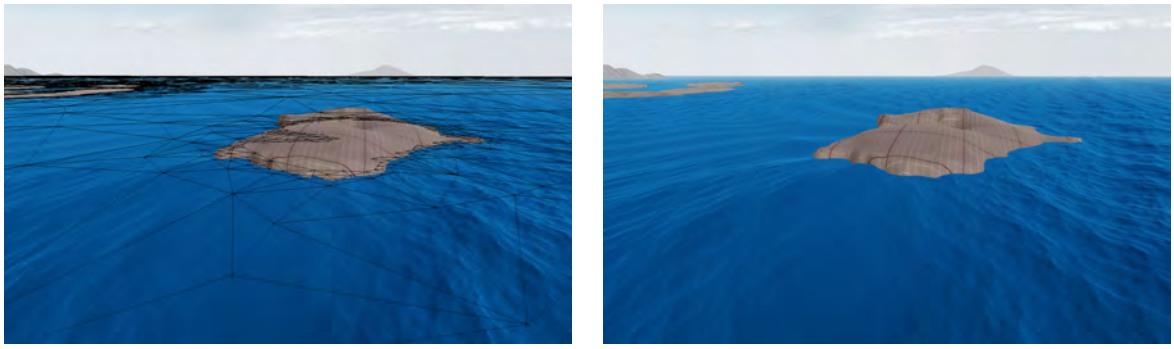
Reflection A water wave reflects when it collides with hard surfaces. Within the simulation, a wavefront reflects when it intersects scene geometry. See Figure 23 for a view of a single wavefront as it reflects off the shoreline.



(a) Simulation step (b) Rendering step

Figure 23: A wavefront reflecting off the shoreline within the simulation. a) Multiple time steps of the wavefront are shown transitioning from green to red with increasing phase. The coarse triangle mesh is shown in black. The seafloor is shown in gray. b) The rendered water surface. Note the null lines that appear where reflected waves cancel each other out.

Dispersion Water waves of different wavenumbers travel at different phase speeds due to dispersion. Within the simulation, this phenomenon is handled naturally by the integration scheme used. See Figure 24 for a view of multiple wavefronts travelling at different phase speeds.



(a) Simulation step (b) Rendering step

Figure 24: Dispersion within the simulation. a) The coarse triangle mesh is shown in black. Due to the amount of information, wavefront time steps are not shown. b) The rendered water surface.

6 Discussion

The proposed algorithm successfully fulfills the aims of the thesis in that it robustly simulates waves interacting with the terrain in a variety of different virtual scenes and renders the animated water surface in real-time at good frame rates (more than 60 Hz). Furthermore, the algorithm has proven to be suitable even for very large scenes as shown in Section 5.

We show how to automatically generate a coarse mesh of good quality, that sample the simulation accurately with few triangles, given any input terrain using a balanced Poisson disk radius function and a robust topology-based convex hull algorithm for constructing the Delaunay triangulation.

Next, we provide a detailed description of how to robustly integrate wavefronts across the water surface and store wave data on the coarse mesh as first showed by Jeschke and Wojtan [2015].

The resulting simulation accurately captures wave behaviors that can be observed for water waves under the high frequency approximation and the lack of diffraction for wavelengths that are comparable to features in the underlying depth field is not noticeable in practice. To improve the simulation further, the wavelength independent diffraction approximation suggested by Jeschke and Wojtan [2015] could be implemented.

The wavefront generators worked well for introducing and propagating a specific sea state from the boundaries of the simulation. However, not all waves are formed by wind far away and more work can be done on automatically generating wavefronts for waves originating from local wind conditions. This would extend the use case of the proposed algorithm to isolated bodies of water such as small lakes or even water puddles.

For rendering the water surface, we show that a staggered update scheme is feasible even at low update frequencies and that it can effectively improve rendering performance by a factor of 8. This contribution makes the algorithm suitable for real-time applications.

The world space aligned quadtree used to construct the geometry of the water surface makes it possible to implement a staggered update scheme. However, the choice of subdividing nodes uniformly along the x and z axis poses some problems. As can be seen from Figure 14, the subdivision scheme over-samples the water surface in the distance. The reason for this behavior is that a node in the quadtree is subdivided until the sampling resolution is below the world space size of a set number of pixels (r_{min}) in the horizontal *and* vertical direction. When viewing the water surface from a low altitude, the vertical screen space sampling resolution is often adequate even at low subdivision levels because of the perspective. However, the horizontal screen space sampling resolution is not good enough at the same subdivision level and the nodes are subdivided further. Because of the uniform subdivision scheme used in the quadtree, this causes the vertical sampling resolution to become extreme and results in very small water surface triangles and wasted computational resources. This is particularly bad as the over-sampling occurs primarily in the distance where the effect of under-sampling might be negligible. In order to improve the system, a non-uniform subdivision scheme could be explored or one could accept a larger

screen space sampling resolution in the horizontal direction and stop subdivision once adequate resolution is reached in any direction.

For performing height-lookups on the water surface, linear interpolation of wave parameters proved to be sufficient. The difference between linear and side-vertex interpolation was not apparent in our test cases because there were often multiple triangles, piecewise approximating wavefront curvature, in areas where wavefronts tend to refract as a result of the tessellation scheme employed to construct the coarse mesh. Thus, the negligible improvement in visual quality provided by the higher order interpolation scheme did not outweigh the increased instruction count required.

To completely avoid visible seams between coarse mesh triangles, one should ideally interpolate the wavelength at each corner of a wave overlap and attenuate waves according to this value when filtering wave overlaps, as opposed to the minimum of the three values. The stopping criterion can use the minimum value for performance considerations, but the attenuation should work with the interpolated value in order to avoid discontinuities in which waves appear on adjacent coarse mesh triangles.

6.1 Cache Coherence

Vertex displacement buffer The persistent vertex displacement buffer is sparse by construction. As slots are allocated and deallocated the buffer will become fragmented with respect to the world space locations of the leaf nodes it references. The displacement stored for vertices within a leaf node will be close in memory but slots of leaf nodes that are adjacent in world space might be far apart in the vertex displacement buffer, or even belong to different render buffer groups entirely.

Ideally, to use the shared memory on the GPU effectively, one would like the slots of leaf nodes that cover a particular coarse mesh triangle to be in the same vertex displacement buffer and to be close in memory within the buffer itself. Then, compute shader threads executing in the same warp and thread group will access the same wave data and use the shared memory of the thread group more effectively. This is the case if the slots of adjacent leaf nodes are adjacent in the vertex displacement buffer.

To solve the issue, the vertex displacement buffer should be defragmented in some way. An idea for a defragmentation routine is to switch the slots of two leaf nodes in the buffer every frame so that slots become gradually sorted by the distance to the viewer.

Static coarse mesh The proposed algorithm as outlined is this thesis and by Jeschke and Wojtan [2015] considers a static coarse mesh independent of the view frustum chosen at runtime. Even though the coarse mesh is adaptively tessellated to handle scale differences, too small coarse mesh triangles are not feasible in practice because they cause memory access coherency issues when viewed from afar.

For example, consider a long stretch of rugged coastline where the coarse mesh is finely tessellated. The evaluation of two neighboring vertices on the water surface (that are executed in the same warp or thread group in the compute shader) might access widely different wave data if the vertices happen to be located in two different

coarse mesh triangles. When the rugged coastline is viewed from a distance, a small screen space area may contain a large number of different coarse mesh triangles and cause an enormous amount of cache misses. Since accessing wave data is typically the most performance intensive task of evaluating the water surface (See Section 5.1), incoherent memory accesses can be particularly bad for the performance of the algorithm.

Note that this is not a problem for small triangles that are close to the viewer since they occupy a large amount of screen space and thus a large number of vertices on the water surface geometry.

See Section 7.2 for a potential solution to this problem.

7 Future Work

7.1 Coarse Mesh As Water Surface Geometry

The coarse mesh tree could potentially be used for rendering, eliminating the need for the coarse mesh triangle lookup procedure when evaluating the water surface. As the lookup time can amount to approximately 50% of the total evaluation time in some cases (See Section 5.1), this would result in increased rendering performance.

Since the coarse mesh is spatially coarse by definition, it would have to be tessellated in a view-dependent manner to be used to directly represent the water surface. This could be done similarly to how the world space quadtree is tessellated in Section 4.3.1. Then, any vertex on the water surface would simply need to traverse the subdivision tree upwards to find the set of wave overlaps to iterate.

7.2 Coarse Mesh Tree

To solve the cache coherence problem described in Section 6.1, we propose recording wave parameters in a *coarse mesh tree* where the tessellation factor can be chosen at runtime so that each coarse mesh triangle occupies approximately the same screen space area. Since wave parameters will need to be stored separately for each subdivision level of a coarse mesh triangle, the approach will be limited by its memory footprint.

In order to construct the coarse mesh tree, the triangles of a coarse mesh may be subdivided until they reach a desired detail level. The simulation can then record wave parameters onto the highest level of detail (LOD) representation in the coarse mesh tree. Once wavefront propagation and recording is done, finished chains can be propagated to a lower LOD in the tree by replacing two chains from two coarse mesh edges that share a sample in the high LOD representation with a single chain on the single coarse mesh edge in the lower LOD representation. See Figure 25 for a graphical explanation of the proposed procedure.

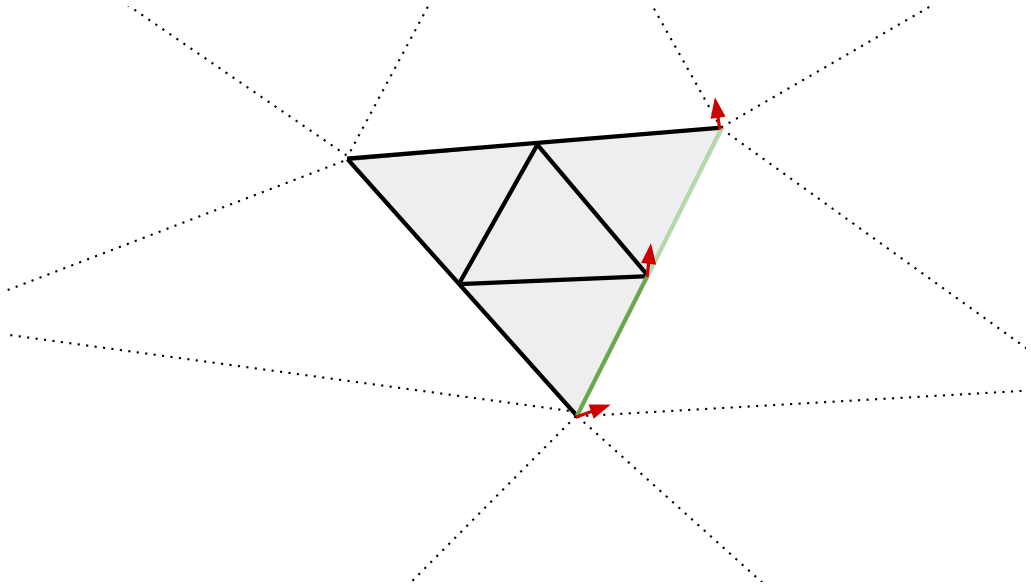


Figure 25: An illustration of how finished chains can be propagated to a lower LOD in a coarse mesh tree. Consider two coarse mesh edges (green and light green lines) on a high LOD representation (small triangles) that line up with a coarse mesh edge on a low LOD representation (large triangle). If a wavefront moved fully passed the low LOD triangle, there must be a chain with 2 samples on the green edge and a chain with 2 samples on the light green edge that share a sample (middle red arrow) in the high LOD representation. In this case, one can approximate the wave in the low LOD representation by creating a chain connecting the samples at the endpoints of the two chains. This procedure can then be performed recursively to lower LOD representations in the coarse mesh tree.

A coarse mesh tree has the potential of capturing everything from the small ripples caused by reflections off individual stones on a close-by beach to the impact of large waves from a roaring sea on a far away shore, without causing massive amounts of cache misses.

8 Conclusion

An algorithm for rendering large expanses of water that interact with the terrain of the virtual scene in real-time was presented. The simulation robustly propagates and records wavefronts across the water surface in a pre-computation step while taking the shape of the terrain into account. As a result, the refraction, reflection and dispersion phenomena of the water waves are captured and the simulation can accurately represent both deep water waves and shallow water waves. During runtime, the simulation is evaluated using a staggered update scheme and the water surface is rendered using an adaptively tessellated world space aligned quadtree.

The staggered update scheme utilizes the observation that the water simulation can be evaluated at a low update frequency without loss of visual quality and effectively improves the performance by a factor of 8 as the computational workload is split across multiple frames.

In future work, a coarse mesh tree could be explored in order to improve cache coherence, which is the perceived bottleneck of the algorithm, and eliminate a lookup procedure to further improve rendering performance.

9 Acknowledgements

I would like to thank my advisors Fredrik Larsson and Jan Schmid (DICE), my coworker Andreas Wennström and my academic supervisor Erik Sintorn (Chalmers) for helping me realize the thesis. In particular, I would like to thank my advisors at DICE for coming up with the idea of the staggered update scheme.

References

George Biddell Airy. Tides and waves. *Encyclopaedia Metropolitana*, pages 241–396, 1841.

Franz Aurenhammer. Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure. *ACM Comput. Surv.*, 23(3):345–405, September 1991. ISSN 0360-0300. doi: 10.1145/116873.116880. URL `http://doi.acm.org/10.1145/116873.116880`.

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull Algorithm for Convex Hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996. ISSN 0098-3500. doi: 10.1145/235815.235821. URL `http://doi.acm.org/10.1145/235815.235821`.

Robert Bridson. Fast Poisson Disk Sampling in Arbitrary Dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM. ISBN 978-1-4503-4726-6. doi: 10.1145/1278780.1278807. URL `http://doi.acm.org/10.1145/1278780.1278807`.

Kevin Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228, December 1979. ISSN 0020-0190. doi: 10.1016/0020-0190(79)90074-7. URL `http://www.sciencedirect.com/science/article/pii/0020019079900747`.

Eric Bruneton, Fabrice Neyret, and Nicolas Holzschuch. Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF. *Computer Graphics Forum*, 29(2):487–496, May 2010. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2009.01618.x. URL `http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2009.01618.x/abstract`.

Robert L. Cook. Stochastic Sampling in Computer Graphics. *ACM Trans. Graph.*, 5(1):51–72, January 1986. ISSN 0730-0301. doi: 10.1145/7529.8927. URL `http://doi.acm.org/10.1145/7529.8927`.

DICE. DICE, 2017. URL `http://www.dice.se/`.

H. Edelsbrunner and N. R. Shah. Incremental Topological Flipping Works for Regular Triangulations. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, SCG '92, pages 43–52, New York, NY, USA, 1992. ACM. ISBN 978-0-89791-517-5. doi: 10.1145/142675.142688. URL `http://doi.acm.org/10.1145/142675.142688`.

Herbert Edelsbrunner and Raimund Seidel. Voronoi Diagrams and Arrangements. In *Proceedings of the First Annual Symposium on Computational Geometry*, SCG '85, pages 251–262, New York, NY, USA, 1985. ACM. ISBN 978-0-89791-163-4. doi: 10.1145/323233.323266. URL `http://doi.acm.org/10.1145/323233.323266`.

Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc., Boca Raton, FL, USA, 2004. ISBN 978-1-55860-732-3.

Mark Finch. Effective water simulation from physical models. *GPU Gems*, 1:5–29, 2004.

Alain Fournier and William T. Reeves. A Simple Model of Ocean Waves. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 75–84, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15894. URL `http://doi.acm.org/10.1145/15922.15894`.

Frostbite. Frostbite - Game Engine, 2017. URL `http://www.frostbite.com/`.

Jocelyn Fréchot. Realistic simulation of ocean surface using wave spectra. In *Proceedings of the First International Conference on Computer Graphics Theory and Applications (GRAPP 2006)*, pages 76–83, Portugal, 2006. URL `https://hal.archives-ouvertes.fr/hal-00307938`.

Franz Gerstner. Theorie der Wellen. *Annalen der Physik*, 32(8):412–445, January 1809. ISSN 1521-3889. doi: 10.1002/andp.18090320808. URL `http://onlinelibrary.wiley.com/doi/10.1002/andp.18090320808/abstract`.

E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, April 1988. ISSN 0882-4967. doi: 10.1109/56.2083.

Jean-Christophe Gonzato and Bertrand Le Saëc. A phenomenological model of coastal scenes based on physical considerations. In Prof Dr Daniel Thalmann and Ass Prof Dr Michiel van de Panne, editors, *Computer Animation and Simulation '97*, Eurographics, pages 137–148. Springer Vienna, 1997. ISBN 978-3-211-83048-2 978-3-7091-6874-5. URL `http://link.springer.com/chapter/10.1007/978-3-7091-6874-5_10`. DOI: 10.1007/978-3-7091-6874-5_10.

Jean-Christophe Gonzato and Bertrand Le Saëc. On modelling and rendering ocean scenes. *The Journal of Visualization and Computer Animation*, 11(1):27–37, February 2000. ISSN 1099-1778. doi: 10.1002/(SICI)1099-1778(200002)11:1⟨27::AID-VIS214⟩3.0.CO;2-5. URL `http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1099-1778(200002)11:1<27::AID-VIS214>3.0.CO;2-5/abstract`.

Dennis Gustafsson. Convex Hulls Revisited, March 2013. URL `http://tuxedolabs.blogspot.com/2013/03/convex-hulls-revisited.html`.

J. Hauser, M. Sambridge, and N. Rawlinson. Phase space methods for multi-arrival wavefronts. *Exploration Geophysics*, 37(4):331–339, December 2006. ISSN 0812-3985. doi: 10.1071/EG06331. URL `http://library.seg.org/doi/abs/10.1071/EG06331`.

Stefan Jeschke and Chris Wojtan. Water Wave Animation via Wavefront Parameter Interpolation. *ACM Trans. Graph.*, 34(3):27:1–27:14, May 2015. ISSN 0730-0301. doi: 10.1145/2714572. URL http://doi.acm.org/10.1145/2714572.

Claes Johanson and Calle Lejdfors. Real-time water rendering. *Lund University*, 2004. URL http://fileadmin.cs.lth.se/graphics/theses/projects/projgrid/projgrid-lq.pdf.

Yuri Kryachko. Using vertex texture displacement for realistic water rendering. *GPU Gems*, 2:283–294, 2005. URL https://kineme.net/files/GPU_Gems2_ch18.pdf.

Scott A. Mitchell, Alexander Rand, Mohamed S. Ebeida, and Chandrajit Bajaj. Variable radii Poisson-disk sampling, extended version. In *Proceedings of the 24th Canadian Conference on Computational Geometry*, volume 5, 2012. URL http://www.academia.edu/download/39785603/Variable_Radii_Poisson-Disk_Sampling20151107-8769-c3vpn8.pdf.

Stanley Osher, Li-tien Cheng, Myungjoo Kang, and Hyeseon Shim Y. Geometric Optics in a Phase Space Based Level Set and Eulerian Framework. *J. Comput. Phys*, 79:622–648, 2002.

Darwyn R. Peachey. Modeling Waves and Surf. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 65–74, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. doi: 10.1145/15922.15893. URL http://doi.acm.org/10.1145/15922.15893.

Willard J. Pierson and Lionel Moskowitz. A proposed spectral form for fully developed wind seas based on the similarity theory of S. A. Kitaigorodskii. *Journal of Geophysical Research*, 69(24):5181–5190, December 1964. ISSN 2156-2202. doi: 10.1029/JZ069i024p05181. URL http://onlinelibrary.wiley.com/doi/10.1029/JZ069i024p05181/abstract.

Inigo Quilez. Inverse Bilinear Interpolation, 2010. URL http://iquilezles.org/www/articles/ibilinear/ibilinear.htm.

N. Rawlinson, J. Hauser, and M. Sambridge. Seismic ray tracing and wavefront tracking in laterally heterogeneous media. In *Advances in Geophysics*, volume 49, pages 203–273. Elsevier, 2008. ISBN 978-0-12-374231-5. URL http://linkinghub.elsevier.com/retrieve/pii/S0065268707490033.

Olof Runborg. Mathematical models and numerical methods for high frequency waves. *Commun. Comput. Phys*, 2(5):827–880, 2007. URL http://www.nada.kth.se/~olofr/Publications/runborg-cicp.pdf.

Jerry Tessendorf. Simulating ocean water. *Simulating Nature: Realistic and Interactive Techniques. SIGGRAPH*, 1(2):5, 2001. URL https://www-evasion.imag.fr/Membres/Fabrice.Neyret/NaturalScenes/fluids/water/waves/fluids-nuages/waves/Jonathan/articlesCG/simulating-ocean-water-01.pdf.

Pauline Y. Ts'o and Brian A. Barsky. Modeling and Rendering Waves: Wave-tracing Using Beta-splines and Reflective and Refractive Texture Mapping. *ACM Trans. Graph.*, 6(3):191–214, July 1987. ISSN 0730-0301. doi: 10.1145/35068.35070. URL `http://doi.acm.org/10.1145/35068.35070`.

V. Vinje, E. Iversen, and H. Gjøystdal. Traveltime and amplitude estimation using wavefront construction. *GEOPHYSICS*, 58(8):1157–1166, August 1993. ISSN 0016-8033. doi: 10.1190/1.1443499. URL `http://library.seg.org/doi/abs/10.1190/1.1443499`.