LiU-ITN-TEK-A--09/057--SE

# A VFX ocean toolkit with real time preview

Björn Rydahl

2009-10-02

LiU-ITN-TEK-A--09/057--SE

# A VFX ocean toolkit with real time preview

Examensarbete utfört i Medieteknik
vid Tekniska Högskolan vid
Linköpings universitet

Björn Rydahl

Handledare Fredrik Limsäter
Examinator Björn Gudmundsson

Norrköping 2009-10-02

# A VFX ocean toolkit with real time preview

Björn Rydahl

September 2009

# Abstract

When working with oceans in the visual effects industry, it is not always very practical or even possible to use real live footage, especially if extreme weather conditions are required. A common scenario is computer generated objects crashing into an ocean generating splashes and foam that should stick to and integrate well with the real ocean surface. Making a shot like that look realistic is very difficult, this is where a fully computer generated ocean surface comes in handy.

Creating high resolution computer generated sequences of an ocean surface with interacting objects is difficult using today's available commercial 3D packages. I have therefore implemented a VFX ocean toolkit, which is a system built for generating the ocean surface, Kelvin wakes and interaction with objects. The ocean toolkit was built with the artist in mind and the need for real time preview to produce results quick and easy in order for the system to remain cost effective.

The ocean toolkit is tightly integrated directly into the procedural 3D animation package Houdini[1] as several plug-ins and shaders that can be combined to create numerous ocean surface effects.

[1]http://www.sidefx.com

# Contents

# List of Figures

# Chapter 1

# Introduction

An ocean is a common nature phenomenon and therefore an important element in motion pictures. Capturing live footage with that perfect ocean look can be very difficult if not impossible due to weather conditions or simply because it does not exist. Integrating splashes and foam with a real ocean surface is also a very difficult, but common task, since it is almost impossible to track the surface and make the foam stick to it. When creating a computer generated surface it is important that it looks very similar to the captured reference footage or story board since that is what the director is aiming for.

The need for a digital ocean system to create visual effects is obvious. Several articles and papers have been published describing how to create ocean like effects but none of them are implemented in the commercial 3D animation packages available today. The most common approach when generating ocean surface waves is described in [1] and has been used successfully in various projects for several years producing very realistic results.

This thesis describes how to implement a VFX ocean toolkit, using previously published literature combined with our own techniques to create a very flexible and controllable ocean generation tool that can be used in a production environment. To achieve our goals large efforts have been put towards enabling real time previews in all the system components.

The reader is expected to have strong knowledge of computer graphics. A good starting point is to read [5].

## 1.1 The ocean toolkit

The obvious approach to generate waves would be to simulate the whole ocean in 3D as a full scale fluid [6]. Because ocean surfaces can stretch for several hundred kilometers a fluid simulation would not be very practical and also a waste of computational resources when we are only interested in the large scale surface motion. Fluid simulations produce very realistic results and could perhaps be used for local object interaction on the surface. In this thesis I have chosen not to use

3D fluid simulations at all since they are to slow and also several very functional implementations already exist in commercial 3D animation packages.

The ocean toolkit is completely 2D grid based where a flat surface grid is displaced in three dimensions and animated with time to create realistic wave motion and allow for waves with sharp peaks to appear.

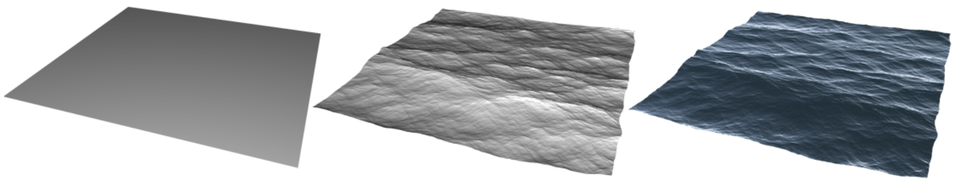

Figure 1.1: The steps to create an ocean surface. Flat grid (left), displaced grid(middle) and final displaced and shaded surface.

The ocean toolkit is built from several different methods to be able to simulate the most important features of an ocean. Three key components exist which are referred to as "Ambient Waves", "Kelvin Wakes and Turbulent Wake" and finally "Object-Water interaction". The reason for splitting up the system into three components is simply because it is not a full scale fluid simulation which would handle all types of effects at once. The "Ambient Waves" component generates the characteristic ocean waves but suffer from the limitation of being very difficult to interact with. The "Object-Water Interaction" component was therefore introduced to enables surface interaction with floating or stationary objects. Generating wakes from boats proved to be difficult and unnecessarily slow using the previous described components because wakes stretch for several kilometers when the wind is calm. Therefore a third component "Kelvin Wake and Turbulent Wake" was introduced.

## 1.2 Houdini

The ocean toolkit is tightly integrated directly into the procedural high-end 3D animation package Houdini[1] as several plug-ins and shaders that can be combined to create numerous ocean surface effects. Houdini was chosen because of its node based structure and purely procedural nature. Houdini is also a very flexible system designed with the more technical artist in mind which suits a system like this perfectly.

### 1.2.1 Operators

The procedural nature of Houdini is found in its operators which can be connected into networks and create highly detailed geometry in relatively few steps compared

[1]http://www.sidefx.com

to other software packages. It is possible to build custom operators using C++ and the HDK, Houdini development kit. The ocean toolkit consists of custom made surface operators (SOPs) and VEX operators (VOPs). VEX (Vector Expression) is one of the internal languages of Houdini and is similar to the RenderMan shading language. The SOPs are used for displacing the low resolution geometry used when doing previews while VOPs are used to achieve the same results but with much higher detail at render time. The real power of Houdini is the possibility to use our custom made nodes together with the native Houdini nodes and push the system beyond its own limits.

### 1.2.2 Mantra and shader networks

Mantra is the name of the internal Houdini renderer which is a production class renderer. It has many similarities with RenderMan[2] and supports both micro polygon renderings as well as ray tracing. Shaders are scripted using the VEX language or by using VOPs in the node based interface and are converted into VEX code at render time. Building VOP networks instead of writing code is extremely beneficial when it comes to overview of writing very complex shaders. Thinking in a node based manner is actually very similar to programming but with a much better overview and scalability. There are also simple scripting nodes that can be used since some operations are just easier to do by writing code. Programming using nodes is not only applicable to shaders but can also be used directly on geometry by transforming the vertices in VOPSOP networks.

### 1.2.3 Point clouds

Houdini supports point clouds which are similar to RenderMan brick maps. Point clouds are points in space with assigned attributes. They allow more complex light interactions such as subsurface scattering and ambient occlusion. It is also a great storage format for point data that can be evaluated at render time. We often store foam in point clouds since they provide resolution independence and are attached easily to the surface using uv coordinates. Making lookups and finding the closest points during rendering is relatively fast and therefore very usable.

[2] http://renderman.pixar.com

# Chapter 2

# Ambient Waves

## 2.1 Overview

An almost infinite amount of sin-waves traveling in all directions with different amplitudes and wave lengths is what gives a characteristic ocean surface look. Simulating a large ocean using fluid dynamics equations is not only impractical but impossible due to calculation costs. We have to cheat and only generate the actual surface of the ocean without underwater activity. The goal is to be able to displace a flat 2D surface into something that looks like an ocean.

Several papers and articles have been presented in this field over the last couple of years and even though results are very promising the main method used [1] has some major limitations and drawbacks. Despite that I use the very same method as the foundation for the ambient wave system. Major efforts have been put toward improving and getting around these limitations making the method even more suitable for production.

## 2.2 Mathematical background

When simulating fluids [6] of any kind the incompressible Navier-Stokes equations often come in handy, but as stated previously it has to be simplified when simulating a large ocean surface.

My intention is not to fully explain the reduction of the incompressible Navies-Stokes 2.1 equation but rather give the reader an idea of the mathematical background used for generating the ambient waves. A full explanation can be found in [11] and in [1].

$$\frac{\partial u}{\partial t} + u \bullet \nabla u = -\frac{\nabla p}{\rho} + \nu \nabla^2 u + F \tag{2.1}$$

$$\nabla \bullet u(x,t) = 0 \tag{2.2}$$

The Bernoullie's equation can be derived as a nonlinear reduction of the Navier-Stokes equations where the degrees of freedom of the flow is reduced from three to a single one by introducing a velocity potential instead of the velocity $u(x,t)$.

$$\frac{\partial \Phi}{\partial t} + \frac{1}{2}(\nabla \phi)^2 = -p - U \tag{2.3}$$

$$\nabla^2 \phi(x,t) = 0 \tag{2.4}$$

The Bernoullie's equation is still quite complex since it is capable of simulating breaking waves and has to be reduced even further. This reduction is described more in depth by [1] where linearization is done by removing the quadric term $\frac{1}{2}(\nabla \phi)^2$ and therefore removing the most complicated wave motion. The equation is also restricted to evaluating only the quantities on the actual surface being a height field function that only depends on horizontal position. The energy term $U$ is

$$U = gh \tag{2.5}$$

where $g$ is the gravity constant $9.8 m/sec^2$ and $h$ is surface height. Because we are restricting ourselves to the surface, pressure can be seen as a constant which we choose to be 0. This means that Bernoullie's equation has been linearized to

$$\frac{\partial \Phi(x_\perp,t)}{\partial t} = -gh(x_\perp,t) \tag{2.6}$$

where $x_\perp$ is a label of horizontal position. The mass conservation equation 2.4 has also been restricted to the surface by converting it into two components, one that labels horizontal position and one that is pointing down into the water volume.

$$\left\{ \nabla_\perp^2 + \frac{\partial^2}{\partial y^2} \right\} \phi(x_\perp,t) = 0 \tag{2.7}$$

To fulfill 2.7

$$\frac{\partial}{\partial y} = \pm\sqrt{-\nabla_\perp^2} \tag{2.8}$$

must hold. Combining these equations using the time derivative of the velocity potential we get a single equation describing the evolution of the surface height.

$$\frac{\partial^2 h(x_\perp,t)}{\partial t^2} = -g\sqrt{-\nabla_\perp^2}\, h(x_\perp,t) \tag{2.9}$$

This equation transforms into a two dimensional Laplacian equation after taking two more time derivatives.

## 2.3 Dispersion relation

The previously described and quite complex math ends up in a small and very simple equation, describing how wave propagation varies with the wavelength or frequency of a wave, called the dispersion relation. More details on how to find the solution of 2.9 can be found in [1] where research to verify the dispersion relation using motion from real ocean waves is discussed.

$$\omega^2(k) = gk \tag{2.10}$$

where $k$ is the wave number, $g$ is gravity and $\omega$ is the frequency. Equation 2.10 holds for deep water where the water depth can be ignored. If the bottom is shallow compared to the wave length the depth is accounted for by using the dispersion equation

$$\omega^2(k) = gk \tanh(kD) \tag{2.11}$$

where $D$ is the water depth.

## 2.4 Ocean wave spectrum

If we look at an open ocean, we notice that the surface consist of a large number of waves with various length and period travelling in all directions. Describing this surface using sinusoids is simply not an option due to its complexity. Instead we look at the concept of a spectrum that is based on work by Joseph Fourier. Using Fourier transform, any surface can be represented as an infinite series of cosine and sine functions oriented in all possible directions. Finding the series representing an ocean surface is, if not impossible, then extremely difficult. The frequency spectrum can, on the other hand, be found quite easily. It is simply a matter of measuring ocean height at a sample point for a certain amount of time, calculate the Fourier transform of the time series and generate a periodogram. The periodogram is usually very noisy and therefore an average of several periodograms should be used. The average periodogram is called the spectrum of the wave-height and gives the distribution of the variance of surface height at the sample point as a function of frequency. More about the ocean wave spectrum can be found in [7]. There are several analytical semi-empirical models for calculating the wave spectrum. The most commonly used is the Phillips spectrum [9] which produces waves driven by the wind. Other models include the Pierson-Moskowitz Spectrum [10] and the JONSWAP spectrum [7].

## 2.5 Creating waves using fourier transform

Generating waves using Fourier transform was introduced to the computer graphics society by Mastin et al.[8] who transformed white noise from the spatial domain to the Fourier domain where it was filtered using the Pierson-Moskowitz spectrum and then inverse transformed, resulting in a realistic looking ocean height field. The waves where animated by shifting the phase each frame. A few years later Tessendorf [1] described a similar approach to the problem where he started in the frequency domain, using the Phillips [9] spectrum together with Gaussian distributed noise, and transformed it to the spatial domain using an inverse FFT. The approach described by Tessendorf has been used in production several times and is still the main preference when it comes to creating renderings for films and commercials. I have therefore used it as the foundation when creating ambient waves.

As stated in [1], a wave height field $h(x,t)$ can be represented as the sum of sinusoids with complex and time-dependent amplitudes

$$h(x,t) = \sum_{\vec{k}} \tilde{h}(\vec{k},t) exp(i\vec{k} \cdot x) \tag{2.12}$$

where $\tilde{h}(\vec{k},t)$ is the height fourier amplitudes that define the surface structure, $t$ is time and $\vec{k}$ is the wave vector which is a horizontal vector that indicates the direction of wave propagation.

The wave vector is calculated as $\vec{k} = (k_x, k_z)$ where $k_x = 2\pi n/L_x$ and $k_z = 2\pi m/L_z$, $n$ and $m$ are integers with bounds $-N/2 \leq n < N/2$ and $-M/2 \leq m < M/2$. $N$ and $M$ are the resolution of the FFT while $L_x$ and $L_z$ are used for scaling. From the wave vector $\vec{k}$ we get the wave number $k$

$$k = \left|\vec{k}\right| \tag{2.13}$$

and the wave length $\lambda$

$$\lambda = \frac{2\pi}{k} \tag{2.14}$$

The wave number $k$ also gives the frequency $\omega$ using the dispersion relation from equation 2.10. The fourier amplitudes are generated to be consistent with oceanographic phenomenology and I generate the coefficients using Gaussian distributed random numbers together with the Phillips wave spectrum as suggested by [1].

$$\tilde{h_0}(k) = \frac{1}{\sqrt{2}}(\varepsilon_r + i\varepsilon_i) sqrt P_h(k) \tag{2.15}$$

where $\varepsilon_r$ and $\varepsilon_i$ are random numbers from a Gaussian random number generator with mean 0 and standard deviation 1. The Phillips wave spectrum is used to make it look like waves on a real ocean.

$$P_h(k) = A \frac{exp(\frac{-1}{(kL)^2})}{k^4} exp(-k^2 l^2) \left|(\hat{k} \cdot \hat{\omega}) d\right|^a \tag{2.16}$$

where $A$ is the Phillips equilibrium constant defined as 0.0081 [9] or it can be a user defined amplitude constant which is the case in my implementation. $k$ is the wave number, $L$ is the largest possible wave due to wind speed and gravity $L = V^2/g$ where $V$ is wind speed and $g$ is gravity. $\hat{k}$ is the normalized wave vector,

$\hat{\omega}$ is the normalized wind direction and $l$ is the smallest possible wave length. $a$ is the wind-wave alignment term and $d$ is an artificial damping term used to damp reflections caused by the Fourier Transform. Now I can calculate the Fourier amplitudes for the wave height field at any given time $t$ using the dispersion relation and the previously defined coefficients as

$$\tilde{h}(\vec{k},t) = \tilde{h}_0(\vec{k})exp(i\omega(\vec{k})t) + \tilde{h}_0^*(-\vec{k})exp(-i\omega(\vec{k})t) \tag{2.17}$$

**Algorithm 1** Generating waves using fourier transform

```
1: for all grid cells do
2:     Calculate $\vec{k}$
3:     Calculate coefficients $\tilde{h}_0$ and $\tilde{h}_0^*$ using gaussRand(), $\vec{k}$ and $P_h(k)$
4:     Calculate fourier amplitudes $\tilde{h}$ using $\tilde{h}_0$ and $\tilde{h}_0^*$
5: end for
6: displacementmap = IFFT($\tilde{h}$);
```

The waves generated using Fourier transform are very smooth with rounded peaks that work well in fair-weather conditions or as building blocks for larger waves. When the wind strength increases, natural waves are often sharply peaked at the tops or as referred to in [1], choppy. The waves are made choppy by introducing a horizontal displacement together with the already defined vertical displacement. The horizontal displacement is generated using an IFFT and the previously calculated fourier amplitudes as described in [1].

Displacement in the horizontal plane generates waves with sharp peaks and also introduces the circular effects often seen when an object is floating on the surface and moving in circles as described by Gerstner [12]. Choppy waves are added to the height field and controlled by a constant $\lambda$ that can be user defined to set the amount of choppiness. When the constant is high, waves tend to self intersect which not only ruins the illusion but should also generate spray and foam. Tessendorf [1] introduced the jacobian of the horizontal transformation which is a measurement of its uniqueness. Zero displacement means the jacobian is 1 and a value less than zero indicates self intersection.

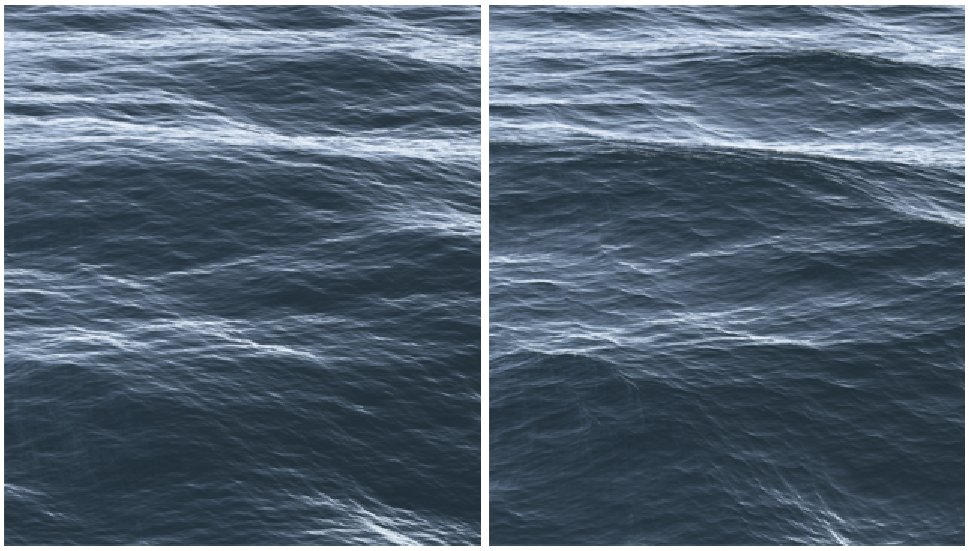

Figure 2.1: Demonstration of choppy waves (right image) compared to smooth waves (left image).

## 2.6 Repetition removal system

The resulting wave height field is periodic and will therefore tile perfectly, making it possible to create an infinite ocean with just one small tile generated by a simple Fourier transform. Repetition is hardly noticeable if the size of the tile is kept large and the camera angle selected with care. For production renderings, high detailed oceans are required and make even the largest tile quite small. Giving artists full control and freedom to move the camera around as they choose is also important. Therefore I have introduced a tiling removal system which transforms (warps) and blends the surface making every part of an almost infinite ocean look unique. The tiling removal is based on transforming the world coordinates using noise before the displacement lookup in the ocean tile. Doing this will result in every tile looking slightly different but still fit together perfectly. The noise used is somewhat arbitrary, but a simple trick to make it behave nice is to use the x-coordinate of the shader sample to warp in the z-direction and vice versa. It is also important to be able to control the scaling of the warp because it can very easily create strange artifacts where noise patterns become visible instead of the wave characteristics. Depending on the wave shapes and camera angle, one warp is usually not enough and therefore a double sampling was introduced which is the mean value of two tile samples where one is shifted in space. Simplified a double sample can be described as two unique ocean surfaces blended together. To remove tiling artifacts even further in extreme conditions one can mix several double samples using noise. I use Catmull-Rom spline interpolation for all lookups which is not for free and it is therefore a good idea to keep the number of samples as low as possible.

World coordinates are warped using fractal noise and a scaled based on the FFT resolution

$$P_{xwarp} = P_x * scale * fnoise(P_z, 0, P_x) \tag{2.18}$$

$$P_{zwarp} = P_z * scale * fnoise(P_x, 0, P_z) \tag{2.19}$$

before a lookup is made in the previously generated ocean tile. $P_x$ is the x-coordinate of the current shader sample and $P_{xwarp}$ is the new x-coordinate which has been warped/transformed and will be used for finding out the displacement of the current shader sample. $fnoise()$ is a function used to generate fractal noise. Several lookups warped individually are combined

$$\vec{D} = snoise\frac{(S_1 + S_2)}{2} + (1 - snoise)\frac{(S_3 + S_4)}{2} \tag{2.20}$$

where $\vec{D}$ is the final displacement, $S_{1\_4}$ is a uniquely warped lookup and $snoise()$ is a function used to generate simplex noise.

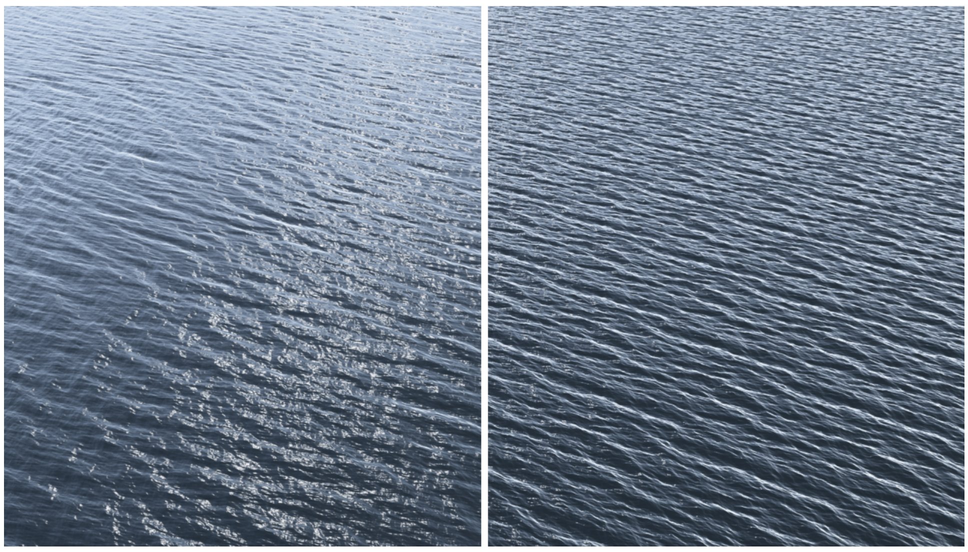

Figure 2.2: Demonstration of the repetition removal system. Left image has repetition removal applied and right image is the original.

## 2.7 Realtime preview

The system is built to be used in production and therefore the ability to preview the surface while searching for that perfect ocean look is important. On modern workstations a multi threaded FFT-library runs very fast even at extreme resolutions such as $4096^2$, but still not in real time. Therefore I have introduced the possibility to preview the surface using anything from $32^2$ to $4096^2$. The lower resolutions are simply low pass versions of the high resolution and therefore result in a smoother surface which looks the same but can be animated in real time. Making this work is simply a matter of scaling and of course making sure the Fourier coefficients represent the same frequencies even if the resolution is changed. This is done by generating small frequencies first and to ensure the same random number is always used to generate a certain frequency.

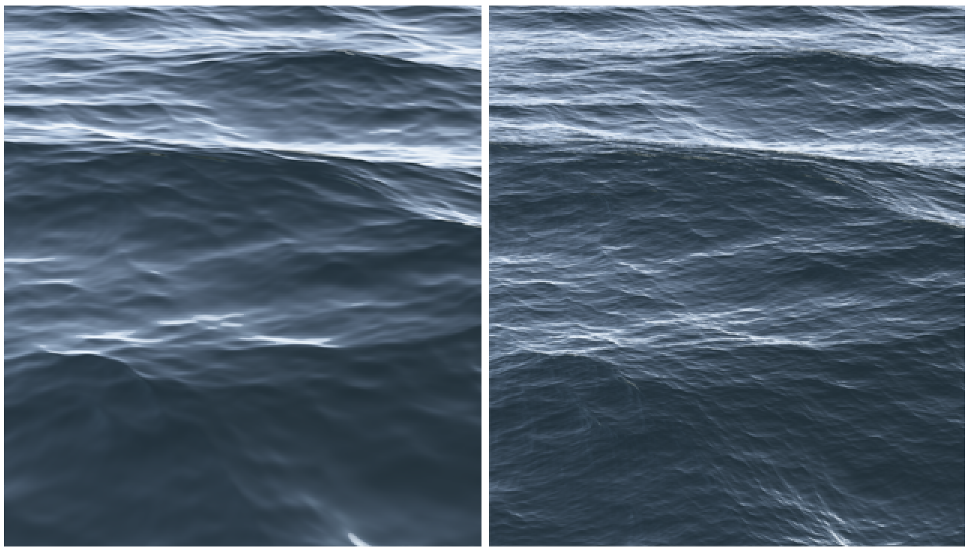

Figure 2.3: Demonstration of different resolutions producing the same results.

## 2.8 Huge Ambient waves

The system was developed with a special project in mind and we needed to be able to simulate extremely rough weather conditions with giant waves. Generating really large waves while also achieving high surface detail needed for closeups, is simply not possible because I would like to keep the FFT resolution at a maximum of $4096^2$ to not loose precision or use to much memory. Wavelengths have to be smaller than the resulting tile for realistic wave motion. Larger waves require larger tiles which is possible at a cost of surface detail. This can be thought of as having a 2D texture with a certain resolution. It is not possible to increase the scale without loosing the fine detail. We need to be able to do this and I have therefore introduced the possibility of placing several independent tiles and ocean wave spectrums on top of each other which I refer to as stacking. Stitching the surfaces together is possible by using a UV-space which also prevents them from sliding on top of each other. The UV-space is simply uv-coordinates attached to the points of the surface. When the flat grid is beeing displaced I use the uv-coordinates instead of world coordinates for finding out what displacement to do. The reason for using uv-coordinates instead of world coordinates is that they stick to the surface and do not change even if the surface is transformed (displaced). The same uv-space is then used when displacing the surface even further using another wave spectrum. The latest displacement will therefore stretch to and follow any motion of the first displacement.

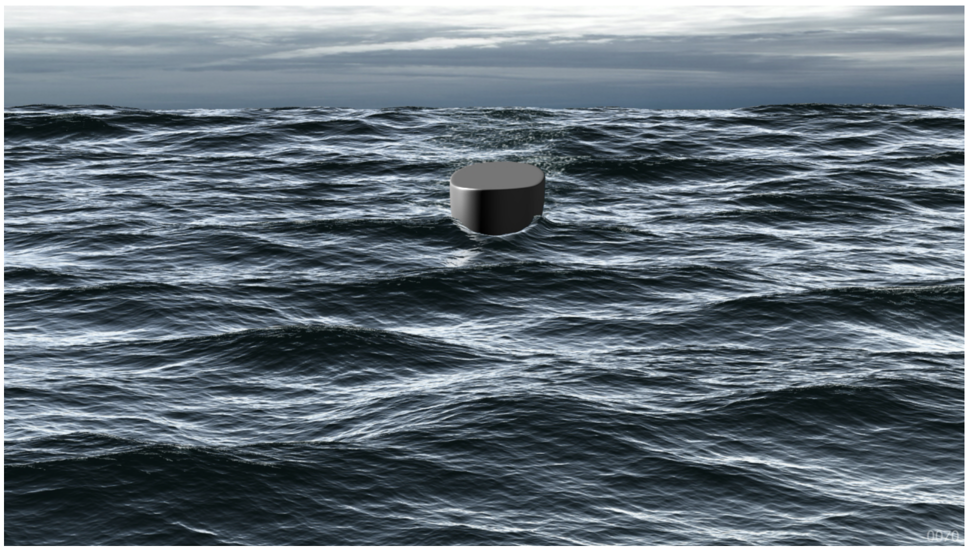
Figure 2.4: Small waves combined with a larger wave using stacking.

## 2.9 Mastering the waves

Absolute control is essential when generating digital images for motion pictures. Simulations are great to a certain extent but will always need tweaking and fixing in the end. I have therefore added several scaling and shifting parameters for the time and the spatial domain. Scaling and shifting is far from enough because they modify the overall wave shapes and not the individuals. I therefore also added the possibility to tweak certain waves using the built in Houdini paint tools and gain even more control. Modifying a single wave shape is simply a matter of hand painting directly onto the ocean surface mesh. Also, as a result of the previously described UV-space which enables stacking it is possible to use a hand animated mesh as the base building block for very specific motion or wave phenomenas. The ability to stack surfaces together proved very useful when creating a specific look using real footage as reference. Not only did it improve results but also decreased the time needed for the initial setup.

## 2.10 Whitecaps and foam

Foam is one of the most important components when making a digital ocean look realistic. For ambient waves, foam appears when waves fall over and break or simply when they become choppy enough. The wave spectrum approach is not suitable for real breaking waves but when wave peaks become sharp, whitecaps and foam should appear. One way to determine when this happens is by using the jacobian of the horizontal transformation and this works fairly well but becomes very unreliable if used together with the repetition removal system. This is because we sample the surface several times which can make self intersecting waves very smooth if merged with a wave similar to its inverse. Therefore the jacobian might tell us to generate foam where the surface is perfectly flat and ruin the illusion. Another drawback with this approach is how to make foam stay on the surface for a while before fading away. Each frame an ocean patch is generated and the jacobian tells us where the patch is self intersecting. Keeping track of the foam in the time domain to make it fade away smoothly is simply a matter of keeping the foam from the previous frame. Unfortunately our repetition removal system makes every point on the surface unique and keeping foam data from one frame to the next is therefore not as trivial anymore. Foam generated using the jacobian is referred to as JacobiBreaks. The solution to the problem of generating foam at the right place is a matter of finding out where waves are self intersecting after the repetition removal system has been applied. This can be done using standard collision detection and could be extremely slow. I use an indexed grid as the initial surface and intersections can therefore be found very quickly since I always know the indexes of the neighboring points. Self intersections are found by comparing a point to its neighbors and verify if it has been moved past any of them.

A point cloud is generated from the intersections and is basically points in space with certain attributes attached to them such as birth time, amount of self intersection, UV-coordinate and so on. The point cloud is used at render time in a foam shader. I refer to this method as GeoBreaks. Point clouds are very practical compared to attaching attributes directly to a surface since they provide resolution independence but still stick to the surface using its uv coordinates. I uses both methods when generating foam. The JacobiBreaks method is quite useful for adding very small detail and breaking up homogeneous surface patterns. I deal with the time domain problem by loading several old patches into memory each frame. This works well, is quite fast for less than 50 frames and enables a two second life span. Two seconds is plenty for small detailed waves and I therefore rarely use more than 25 frames. The GeoBreaks method is the preferred approach when generating whitecaps and foam from larger waves.

# Chapter 3

# Kelvin Wake and Turbulent Wake

## 3.1 Overview

The idea with a boat wake system is to be able to generate the typical patterns from boats traveling on the open sea. The optimal solution to this problem would be a system that can handle any type of object interaction with the water surface. The objects can be anything from a floating bottle generating ripples to a full size battleship traveling at high speed across the surface. The characteristic v-shaped ship wake would be generated automatically by simply moving the ship around.

## 3.2 Interaction system

A full fluid simulation in 3D is out of the question since the surface covered by a ship wake can be enormous, stretching several kilometers before it fades away. There are two methods previously described in the computer graphics literature which both rely on deforming a 2D surface. The IWave by Jerry Tessendorf [2] and the Wave Particles by Cem Yuksel [3]. Both methods aim for real time performance such as in computer games. I have implemented them both and will give a short introduction to each one and also why i decided not to use either of them when generating wakes.

### 3.2.1 IWave

IWave has been used in several motion pictures with very realistic looking results. The method was suggested by Jerry Tessendorf back in 2004 [2] and is aimed at real time applications. The idea is based on a 2D convolution and some masking operations. A vertical derivative is calculated using the surface height grid and then used to propagate the surface over time. I found the IWave method quite attractive when dealing with objects moving randomly across the water surface. Unfortunately artifacts start showing up looking like waves traveling in the opposite direction to what is expected from certain object movements. The artifacts are hardly noticeable if the object is moving around quite randomly or if the resolution of the simulation grid is increased. Unfortunately, ships often travel along a relatively straight path and wakes could stretch for several kilometers making this method unusable.

### 3.2.2 Wave Particles

The Wave Particles method by Cem Yuksel [3] is a fairly new idea and it has therefore not been used in production yet as far as I know. It aims for real time applications and uses the graphics processing unit to increase the performace. The idea is based on individual particles unaware of each other but together form wave fronts. They are subdivided into more particles as distance between them increases during wave propagation. The results I achieved with the Wave Particles implementation were similar to when using the IWave method except for the artifacts.

Even though I can handle several hundred thousand particles without trouble, several millions are needed to cover the area of a ship wave. Therefore the Wave Particles method is not suitable for generating wakes either but it is still a very attractive method and will be described more in depth later.

## 3.3 Ship wakes

One reason why none of the previously mentioned methods are suitable for generating ship wakes is because of their low detail. Ship wakes are extremely complex even though they all have a very similar look. Wakes produced by ships or any similar object consist of two parts, the turbulent wake and the characteristic Kelvin wave. The turbulent wake trails the ship and is generated by the propellers of the ship. The surface of the turbulent wake is often almost flat since the turbulence caused by the propellers damps the ambient waves. The v-shaped Kelvin wake named after Lord Kelvin (William Thomson) trails the ship as long arms (Kelvin arms). The most significant parts of the Kelvin wake are the divergent waves and the transverse waves.

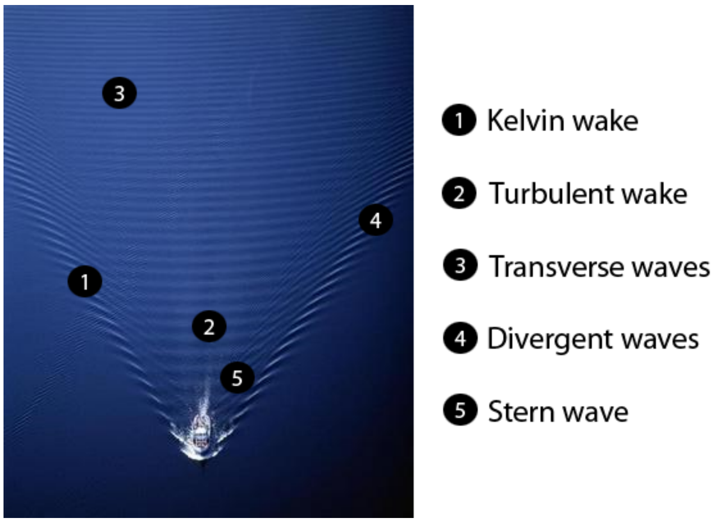

Figure 3.1: The major components in wakes generated by boats.

## 3.4 Generating the Kelvin Wake

Finding an equation that describes the Kelvin wake is a quite complicated task that involves solving a boundary value problem for the moving pressure point [13]. The result on the other hand is simple and consists of two parametric equations

$$x = -Y(2\cos(t) - \cos^3(t)) = -(1/4)Y(5\cos(t) - \cos(3t)) \tag{3.1}$$

$$y = -Y(\cos^2(t)\sin(t)) = -(1/4)Y(\sin(t) + \sin(3t)) \tag{3.2}$$

plotted for $-\frac{\pi}{2} < t < \frac{\pi}{2}$ generates the following patterns

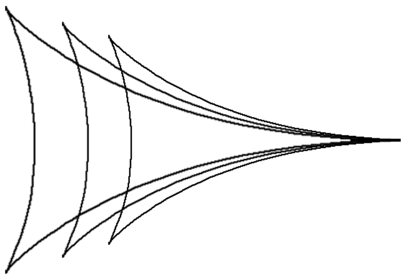

Figure 3.2: Mathematically generated wake.

where $Y$ is the current phase.

Using equations 3.1 and 3.2 with several phases and some other modifications it is possible to generate the divergent and the transverse waves that together compose the heart of the Kelvin wake pattern. When looking at reference footage it appears as if the wake is stationary to the moving ship and we can therefore get away with making the ship drag a texture along its path. Generating such a wake displacement map is easy and just a matter of creating a large enough texture using a modified version of the Kelvin wake equations 3.1 and 3.2. The texture will be kept the same during the simulation due to long calculation times because of the trigonometric functions. Usually the size and wave length differs depending on speed which is something we can fake by fading the texture instead of regenerating it between frames. As always when dealing with rendered images for motion pictures it is all about the looks and cheating/speed is often preferred before physical accuracy.

## 3.5 UV-space

Making the displacement texture follow a boat is quite simple and doing shader lookups is just a matter of shifting the coordinates correctly. Unfortunately we would like the system to support boats traveling along a spline which in many cases will need to be curved. To support splines the texture will need to follow the curve and bend properly. Such operations are not trivial and bending a texture introduce lots of operations that are almost impossible to store without aliasing effects or need for higher resolutions. The solution to this problem is as suggested by Mårten Larsson [4] to use a uv-space. Along the spline we will generate polygon segments until we have generated a polygon surface that covers the area where our wake should appear. The resolution of the surface is based on curvature or user defined settings.

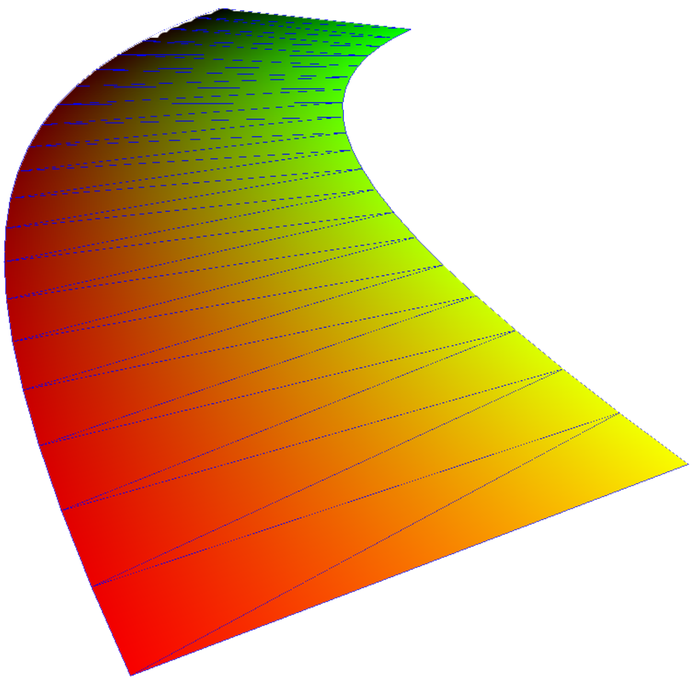

Figure 3.3: The generated geometry rendered with uv coordinates as colors.

UV-coordinates are assigned to each vertex while generating the polygons and rendered above as colors.

During the rendering we use simple collision detection to find out if the shader sample point is situated inside one of our polygons. In case of a collision the exact uv-coordinates can be interpolated from the vertices using barycentric coordinates. Finally the uv-coordinates are used to do a lookup in the previously generated wake displacement texture. The resulting displacement is a wake that looks as if it bends along the spline. To make this work in reality a few tests are made on the polygons to make sure coordinates are valid and overlapping or self intersection does not occur for extreme conditions. Also the segments may have to be tesselated further to remove artifacts caused by stretching in the direction perpendicular to the spline. Tesselating will result in triangles with similar size and also reduce the number of segments needed along the spline even when curvature is relatively high. The polygon surface is generated every time step at render time but is a relatively quick procedure unless the number of boats is very high. The displacement lookups on the other hand can generate quite a render hit since collision detection is involved and highly depends on the number of boats. The collision detection can be improved considerably by only taking into account boats close enough to the shading sample or introducing level of detail.

**Algorithm 2** Generate UV-space

```
1: Get boat animation path
2: Define number of segments using curvature, LoD or user settings
3: Divide path into segments with start and end points
4: for all segments do
5:     Calculate perpendicular vector at start and end point
6:     Use vectors to generate a polygon
7:     Assign UV-coordinates to polygon vertices
8:     Triangulate polygon using curvature, LoD or user settings
9: end for
```

## 3.6 Turbulent wake

The turbulent wake generated by the propellers and the boat hull will follow the path of the boat until broken apart. Since I have already generated the uv-space which is stationary to the boat, the foam covering the turbulent wake can be calculated at render time independently of previous frames. The uv-space tells us how far behind the boat a sample point is and the rest is only a matter of user controlled parameters. Combining the uv-space coordinates with real world coordinates makes it possible to warp the wake and give it a characteristic turbulent look. I use turbulent noise based on time, real world coordinates to make foam appear stationary, and the u coordinate (width) of the uv-space. Then a cosine function is modulated with the noise and scaled using the v-coordinate (length) to give a turbulent marble looking pattern that increases as we move away from the vessel.

$$u_{new} = cos(2\pi * TURB(u,s,t)) * (\frac{v}{v_{max}}) \tag{3.3}$$

where $u_{new}$ is the warped u coordinate used to generate foam, $s$ and $t$ are the uv coordinates of the ocean surface used instead of world position to prevent foam from sliding on top of the ocean surface.

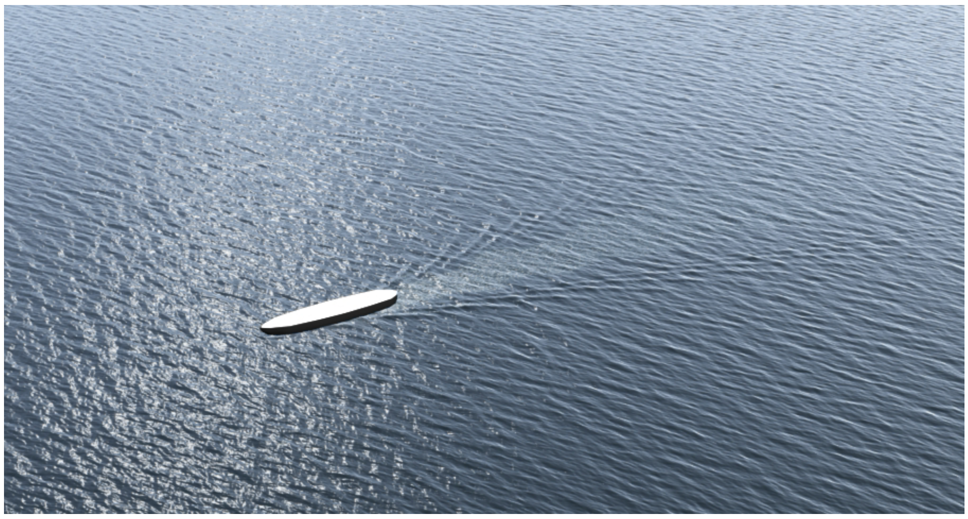

Figure 3.4: Kelvin wake and turbulent wake generated with the ocean toolkit.

# Chapter 4

# Object-Water Interaction

## 4.1 Overview

Ambient waves and boat wakes are both very static systems that can be combined easily but lack the possibility to affect each other and be affected by objects. Therefore an interaction system generating waves using the other systems as input is necessary. When dealing with Boat Wakes we discussed the possibility to use the IWave system presented by Tessendorf [2] or the Wave Particles approach by Cem Yuksel [3]. Both of them had limitations and where not suitable for generating large wakes but worked well in restricted regions and perform much faster than a full 3D fluid simulation. The IWave method suffers from artifacts unless the resolution is increased quite a lot and is also restricted to a height map while the new Wave Particles aimed for real time performance is very smooth even at low resolutions. After several empirical tests and performance considerations I decided to use the wave particles.

## 4.2 Wave Particles concept

The wave particles method is based on a height field representation where the height at a certain point is the sum of a set of local deviation functions. Each local deviation function is associated with what is called a wave particle. Particles are individuals and have nothing to do with other neighboring particles. Waves are formed by several wave particles traveling next to each other forming what is called a wave front.

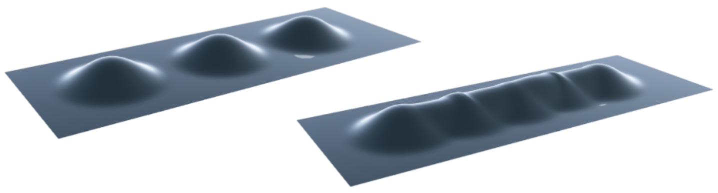

Figure 4.1: Single wave particles compared to a wavefront formed by several waveparticles.

The deviation functions are described in [3] as a wave length based waveform function

$$W_i(u) = \frac{1}{2}(cos(\frac{\pi u}{l_i}) + 1)\prod(\frac{u}{l_i}) \tag{4.1}$$

multiplied with a blending function

$$D_i(x,t) = a_i W_i(u) B_i(v) \tag{4.2}$$

which makes blending neighboring wave particles into wave fronts easier. $l_i$ is the wave length,$u$ is a vector pointing from the current sample point toward the current particle and $\prod$ is a rectangle function. The Cem Yuksel decided to use a radius based deviation function instead where the wave form function is used as a blending function on itself

$$D_i(x,t) = \frac{a_i}{2}(cos(\frac{\pi\,|x - x_i(t)|}{r_i}) + 1)\prod(\frac{|x - x_i(t)|}{2r_i}) \tag{4.3}$$

making each wave particle represent a small circular bump on the height map instead of being shaped as a rectangle. $a_i$ is the particle amplitude and $r_i$ is the particle radius. The reason for using the later approach is the real time aspect, and if wave particles are represented as round bumps they can be rendered efficiently as smoothed points using modern graphics hardware. Because I am not using graphics hardware the use of rectangular wave particles is certainly appealing and would require less particles.

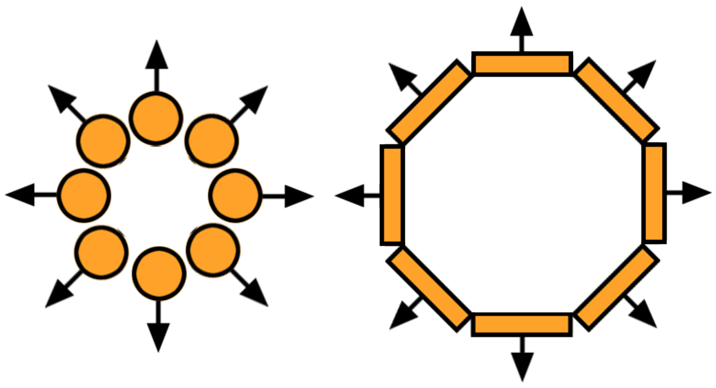

Figure 4.2: Ripple composed of round particles compared to rectangular ones.

Unfortunately rendering rectangular wave particles proved much slower and not as smooth looking as using the circular ones. Wave particles carry birth time and birth position making it possible to find their current position without having to update them each and every frame. They can also carry additional information such as an angle making it possible to figure out when the distance to particle neighbors is getting to high. Keeping these distances within certain limits is important to keep wave fronts intact. When the distance become larger than a predefined threshold a subdivision is performed introducing 3 new particles for each subdivided old particle. Amplitudes and angles become one third of the old particle but the radius, birth time and birth position stay the same.

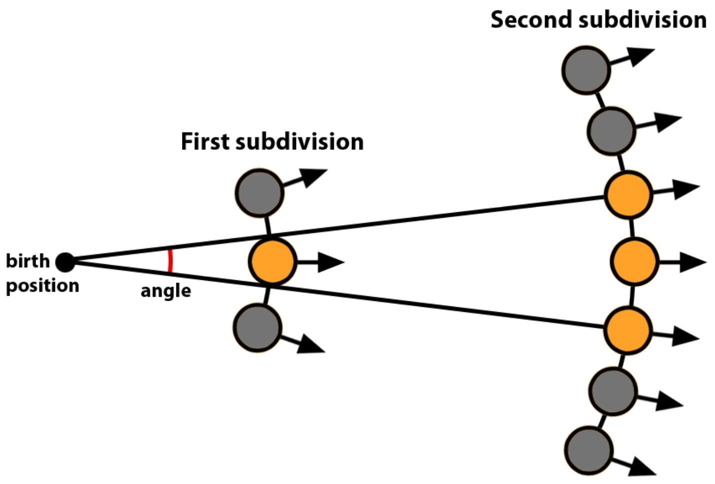

Figure 4.3: Particles are subdivided once they are unable to cover their angle.

The only time particles need to be updated is when a subdivision is required, the particle reaches a boundary or it has been alive long enough to face its death.

Updating particles is a task that should be avoided if possible because our simulation might contain millions of particles. Therefore as suggested by [3] the particles could be associated with buckets that are associated with frames. When a wave particle is generated I calculate subdivision time as

$$p_{subdiv} = p_{birthtime} + \frac{\frac{p_{radius}}{2}}{p_{speed} * sin(\frac{2}{3}p_{angle})} \tag{4.4}$$

and also time of death and time when it hits a boundary. The shortest time gives us what task needs to be performed next and the particle can be ignored until that time. When that time comes I perform the task and do the same thing all over again which is repeated until the simulation stops or the particle is killed. This optimization removes the direct connection between number of particles and updating them, making it possible to use an even larger total number of particles. A major performance issue when using an enormous amount of particles is obviously writing them to disk each frame and disk storage. Disk storage might not sound like a big issue and with today's hard drives it is not, but network traffic can be and disks are usually network disks.

## 4.3 Choppy Wave Particles

When generating ambient waves using the linear Fourier transform, waves ended up very smooth and not as choppy as one would expect. Waves are made choppy by adding an extension to make them behave more like waves should, as stated by [12], where a floating object will move in circles as waves are passing by. An extension generating a similar result when using wave particles was suggested by [3] where the previously described deviation function is divided into two parts, one vertical deviation (the old deviation function eq. 4.2) and one horizontal deviation

$$D_i^L(x,t) = L_i(\hat{u}_i \cdot (x - x_i))D_i(x,t) \tag{4.5}$$

where

$$L_i(u) = -sin(\frac{2\pi u}{l_i})\prod(\frac{u}{l_i})\hat{u}_i \tag{4.6}$$

which is a displacement in the horizontal plane just as when making ambient waves choppy.

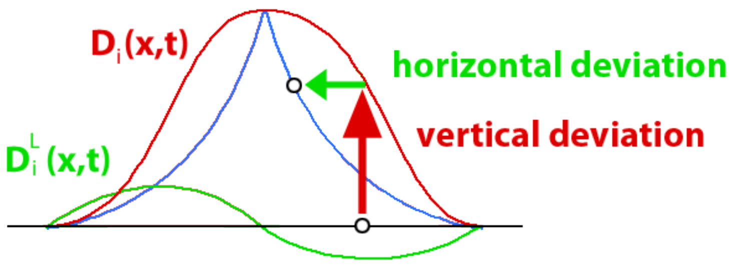

Figure 4.4: Figure describing how a wave is generated from vertical and horizontal displacement.

## 4.4 Generating wave particles

An object which is part of the simulation must consist of triangles since every triangle represents a force pushing or dragging the liquid [3]. The amount of liquid displaced is given by

$$V = A_{face}(U \cdot N)\delta t \tag{4.7}$$

where $A_{face}$ is the face area given by the half the length of the face normal (unnormalized), $U$ is the relative velocity of the face, $N$ is the face normal and $\delta t$ is the time step. If $V$ is positive it means the liquid is pushed and a negative sign means it is dragged. The relative velocity $U$ is given by

$$U = V_o + V_w \tag{4.8}$$

where $V_o$ is the object velocity and $V_w$ is the velocity of the water surface. Object and water velocity can be calculated using time and space derivatives. If the face is placed on the top of the object and still is submerged it will generate a ripple on the surface which is done automatically by creating a wave particle with $2\pi$ dispersion angle. If the face is not on the top side of the object the displaced volume is added to a 2D grid representing the contour of the object seen from above. When all volumes are calculated and added to the grid we redistribute them to their closest contour grid cell. Then volumes are smoothed along the contour by keeping half the volume and redistributing the other half to neighboring contour cells. We also calculate a travel direction at each contour cell given by the direction pointing as far away from the neighboring cells as possible. Directions are also smoothed by summing each of the neighboring directions with the current cell and dividing the results by 2. The angle between the new directions is the angle which the current cell is responsible of as described in 4.5. Both smoothing procedures can be repeated until the result is good enough for its purpose. These operations are extremely quick even with high resolutions since only the contour cells are considered. When the smoothing is done, it is time to generate wave particles from the result. I generate one wave particle from each contour cell and it has one mission, to cover its angle until death.

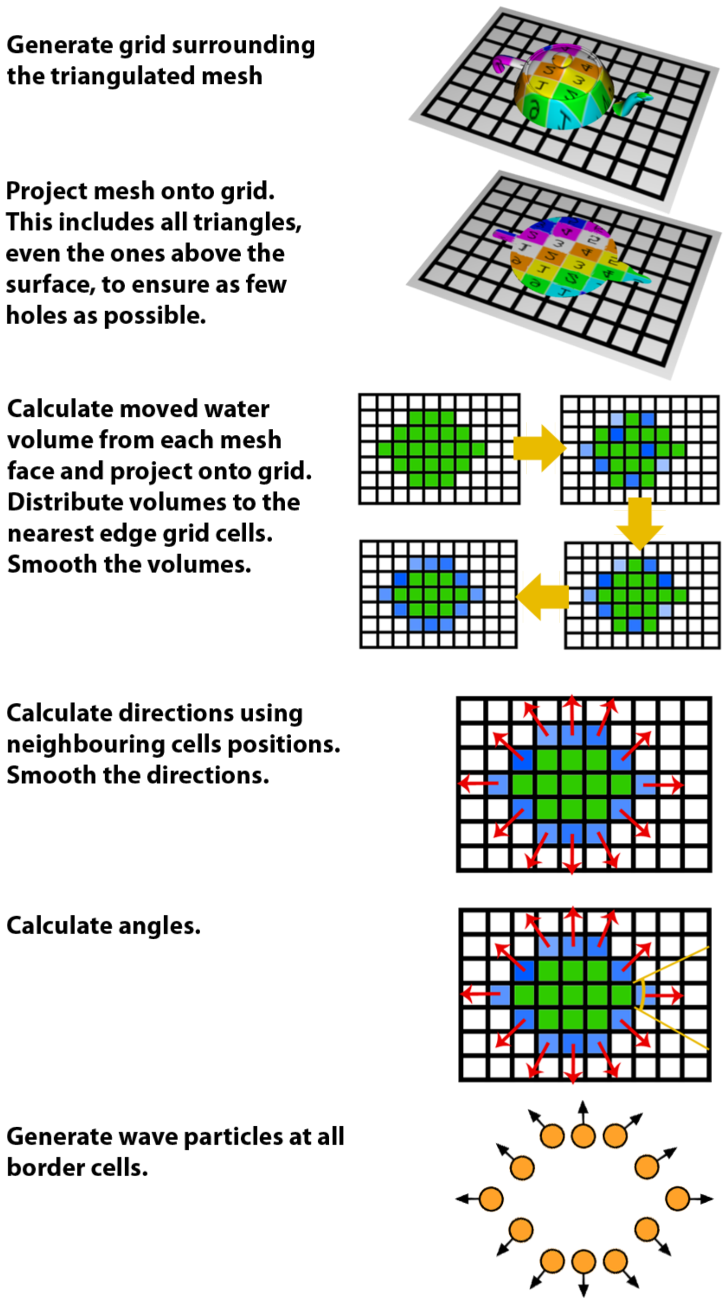
Generate grid surrounding the triangulated mesh
Project mesh onto grid. This includes all triangles, even the ones above the surface, to ensure as few holes as possible.
Calculate moved water volume from each mesh face and project onto grid. Distribute volumes to the nearest edge grid cells. Smooth the volumes.
Calculate directions using neighbouring cells positions. Smooth the directions.
Calculate angles.
Generate wave particles at all border cells.

Figure 4.5: Figure describing how to generate wave particles.

## 4.5 Rendering Wave Particles

The Wave Particles should be rendered as small bumps, based on trigonometric functions, that together form moving wave fronts. The by [3] proposed method is based on using the GPU and the Particles are therefore rendered as points on a grid which is blurred using 4.3 as a kernel. The system we are building is tightly streamlined with Houdini and our first rendering approach was based on using a point cloud to represent the Wave Particles. Point clouds can be loaded into a displacement shader and the closest points accessed quite efficiently for many purposes. When the amount of Wave Particles increased, the point clouds became a huge bottleneck with rendering times of almost a minute for a simple surface displacement. Instead I decided to generate a displacement texture similar to the real time approach. Each particle is added to a texture using 4.3 which involves trigonometric functions. The use of these functions can be eliminated by a lookup table to increase performance even with an enormous amount of particles. The new approach made the same renderings real time. I also render two more textures for the horizontal displacements which will be used along with the height map in the displacement shader.

# Chapter 5

# Shaders

## 5.1 Displacement

One of the most common bottlenecks during rendering is high resolution geometries. The system is flexible enough to be able to reuse the same functionality in a displacement shader as when doing view port previews. This enables extremely high detailed surfaces even when feeding low resolution geometry to the shader and also assures the same results with or without the displacement shader. Huge waves will usually be feed to the shader as geometry while the finest details are generated at render time.

## 5.2 Surface

### 5.2.1 Fresnel

The most important render pass to make a surface look like an ocean is the Fresnel pass that implements the Fresnel equations.

$$R = \left[\frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)}\right]^2 + \left[\frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)}\right]^2 \tag{5.1}$$

$$T = 1 - R \tag{5.2}$$

where R is the reflected light and T is the refracted (transmittance) light.

These equations describe how light is reflected and refracted when moved from a medium with refractive index $n_1$ into a second medium with the refractive index $n_2$. The relationship between the angles of incoming light, reflected light and refracted light is given by the law of reflection

$$\theta_1 = \theta_2 \tag{5.3}$$

and Snell's law (law of refraction)

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{5.4}$$

### 5.2.2 Diffuse

The diffuse pass is used to brighten the surface and thereby be able to adjust to any weather conditions or time of day. The diffuse light is mostly the refracted part of the Fresnel equations which is scaled and colored.

### 5.2.3 Diffuse2

Another diffuse pass but this time generated from light sources to handle special lighting conditions or simply break up the surface even further.

### 5.2.4 Occlusion

The occlusion pass is based on an ambient occlusion shader that tries to find out where waves are occluded by other waves or objects. This is an approximation to full global illumination.

### 5.2.5 Environment map

Most of the light in an outdoor environment is indirect sun light that comes from the atmosphere. An environment map is therefore used to simulate the atmosphere. The texture used is a standard high dynamic range (HDR) image and can be modified to simulate different weather conditions or time of day.

### 5.2.6 Sub Surface Scattering

Subsurface scattering (SSS) is a mechanism of light transport in which light penetrates the surface of a translucent object and is scattered around inside it before exiting the surface at a different point. It is used to make the waves translucent at their peaks where they are thin. The SSS makes light shine through the surface and introduces a more dynamic look. Computing SSS is usually quite expensive but I have found that for an ocean surface it works well even at row resolutions.

### 5.2.7 Fake Subsurface Scattering

The sub surface scattering is expensive to calculate and obviously not practical for a very large ocean. I therefore only calculate the SSS where distance to the camera is small. To also achieve realistic waves further away from the camera I generate a fake SSS render pass which is based on wave heights. This pass can be used in compositing to brighten the waves near their peaks and make them look translucent. This is a cheat but it works quite well.

### 5.2.8 Turbulence layers

Several layers with turbulent noise can also be rendered and used for post processing to break up parts of the surface even further if needed.

## 5.3 Foam

Foam proved to be the most difficult and time consuming shader to write. The shader is fully procedural and therefore based on positions in space and UV-coordinates used to generate noise. The UV-space is used to prevent the surface from sliding in the same way as when stacking several surfaces on top of each other, but also to enable internal stretching of the foam.

The foam shader is split up into three different parts or layers to enable a very flexible system. The foam layers are generated using several types of noise including Simplex noise, Voronoi and Alligator noise. Turbulence is generated using a large span of frequencies and everything is mixed together using various scaling and blending functions. The hardest part is not to make it look like foam but to remove the traces of noise. The only way to do this is by mixing even more noise patterns and choosing frequencies that make repetitions visible only at a larger scale than our camera distance to the surface. Animation of the foam as it dissolves is also an important aspect to keep in mind which unfortunately does not make things easier. Using only noise to create large chunks of realistic looking foam is almost impossible. Therefore another approach which introduces a great amount of flexibility and control together with the possibility to create extremely unique foam patterns is used. I call it PaintedFoam and it will be described later in this thesis.

### 5.3.1 Components

The first layer is called BumpFoam and is a thicker looking foam where I also generate normals used for light calculations to make it more alive and also thicker than surface noise pattern. This layer is used to simulate fresh foam from a very recent splash or breaking wave.

The second layer is called TopFoam and is very similar to BumpFoam except for being in 2D and with a slightly longer lifespan.

The third and final layer is something I call UnderFoam and it is used to simulate the very small bubbles created from splashes and breaking waves. These bubbles make the surface a little bit brighter and will be visible long after the top layers have disappeared. In the wake after a boat those bubbles can be visible for several kilometers if the surface is calm enough.

### 5.3.2 Painted Foam

Painted foam is actually as trivial as the name suggests. I have simply made use of the built in painting tools in Houdini and connected them to our shaders. This enables the user to paint directly on to the ocean surface by using a pressure sensitive digital pen and is thereby able to manually adjust the amount of foam at certain places. Painting by hand makes it possibly to create extremely unique patterns at the correct spots while also giving artists great control over our foam system. The painted foam is mainly used to generate large chunks of foam and make it look like waves have been breaking at certain places but also to create certain characteristics such as the foam bands often visible on large waves. Painting can be done in several layers that are connected to different shaders. I also combine painted foam with point clouds making foam appear only at spots where there is paint and points.

### 5.3.3 GeoBreaks

GeoBreaks are as discussed earlier used to generate foam from waves that are falling over. A point cloud was generated when simulating the GeoBreaks and includes a point for each sample on the surface where self intersection occurred. The points have attributes such as birth time, amount of self intersection and UV-coordinates that will be used when generating foam. The shader loads the point cloud during rendering and locates all points closer than a certain distance to the current render sample in UV-space. A blending function is used to merge values from all neighboring points based on distance. The final values are fed into our foam system which calculates the different foam layers and opacity based on birth time and amount of foam at the current sample.

### 5.3.4 JacobiBreaks

JacobiBreaks used to generate fine detailed foam from small waves falling over. They are not very well suited for larger waves since waves might appear at the wrong places as a direct result of the surface repetition removal system. Other difficulties include how to keep information between frames in memory where solutions once again are limited because of the repetition removal system and the wave spectrum method. These are the reasons for not using the JacobiBreaks as our primer foam generation system but merely as a complement. The JacobiBreaks introduce very fine surface details and enables foam from even the smallest of waves. It is also great for breaking up the surface to make it less homogeneous. The problem with keeping data in memory between frames is not really an issue anymore since the JacobiBreaks are very small and will fade away after only a second. It is therefore possible to load cache files into memory all at once without using too much memory or network resources.

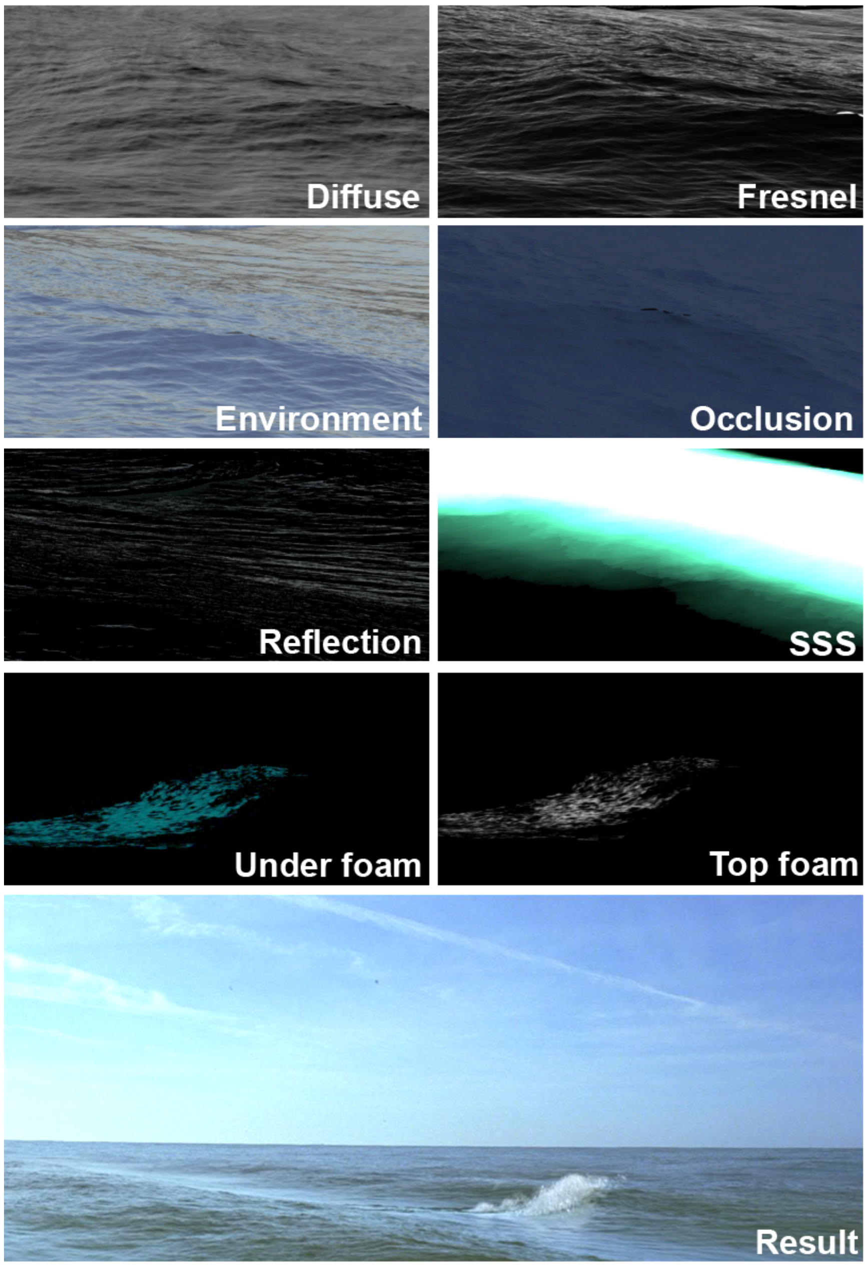

Figure 5.1: Some of the most common shaders used to create a realistic looking ocean surface.

# Chapter 6

# Results

## 6.1 Performance

The overall performance of the system is very good even though there is room for lots of improvements. The bottleneck is not the system itself but rather the rendering process which has to do with shading. The time it takes to calculate the surface displacement is hardly noticeable compared to the shading even with the simplest of shaders. The performance of the system while working with it and when doing previews is also quite fast but it is easy to hit the roof when adding several wave spectrums together or increasing the resolution of the grid or simulation objects. The problem when increasing the resolution is mostly because it takes time to update the geometry and the viewport which is taken care of by houdini.

## 6.2 Example 1

This is a test shot created for a film project where extreme weather conditions were needed and therefore making it impossible to use real action footage. We really tried to push the system to its limits by stacking 5 different wave spectrums on top of eachother. The shot required almost 50 separate render passes where 15 of them where different foam layers. This sequence was rendered in a resolution of 2K and is 10 seconds long. It took about 20-30 minutes per frame on a standard workstation. Since we stacked 5 spectrums on top of each other using a huge grid with over half a million polygons, the preview also ran quite slow with everything turned on. By turning off everything except the layer I was currently working on it ran smoothly.

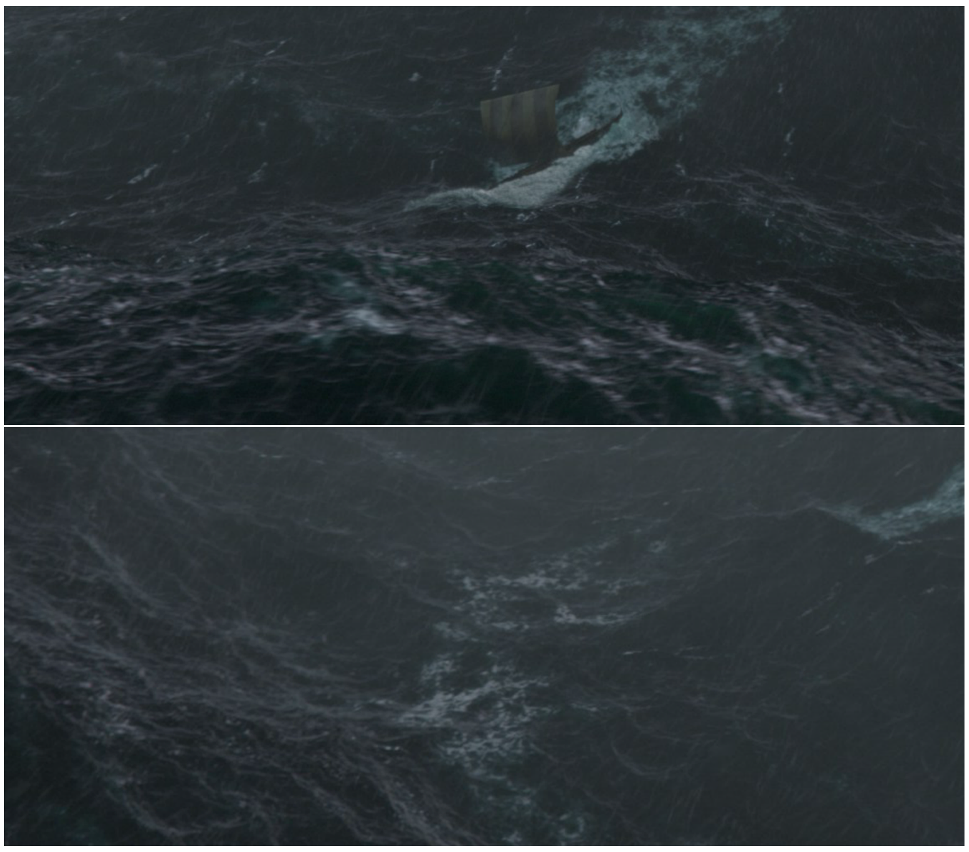

Figure 6.1: Example 1: Pushing the system to the limit by stacking 5 layers on top of each other.

## 6.3 Example 2

A commercial created for the German power company RWE. Ambient waves were used together with the interaction system. The simulated objects move around very rapidly and therefore subframes (see page 37) had to be used for the simulation. For this project several shots were made in a resolution of only 1K and therefore rendered very fast. They all rendered on a single workstation with an average per frame time of only a couple of minutes. Simulation of the interactions ran almost in real time since low resolution spheres where used instead of the high resolution character.

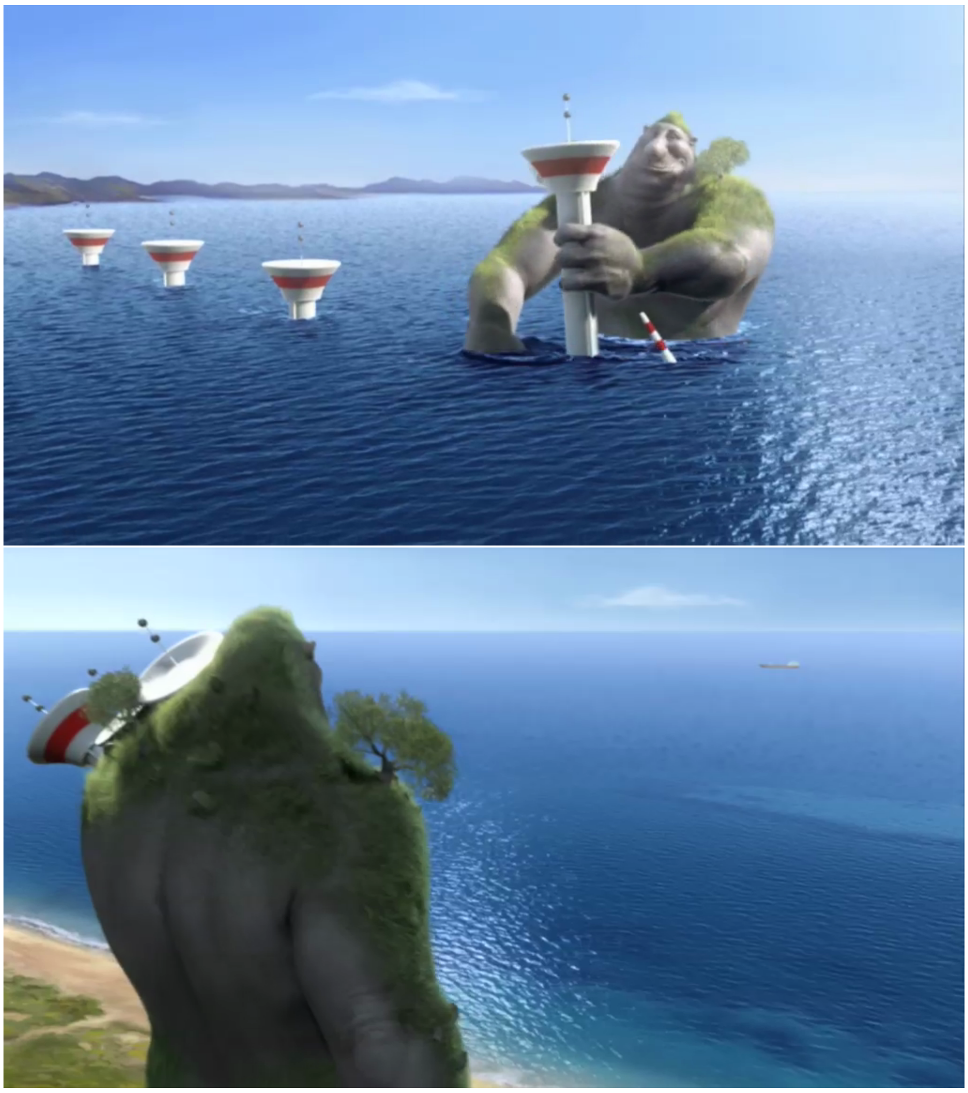

Figure 6.2: Example 2: Project using ambient waves together with the interaction system.

## 6.4 Example 3

A shot created for a film project where real footage was used as reference. The idea was to generate a splash with foam and composite it together with the captured footage. Making the splash and foam integrate well with a real ocean is a very difficult task. We therefore generated a very similar surface using 2 wave spectrums stacked on top of each other. Creating a splash with foam that sticks to the surface proved to be much easier using a computer generated surface. The surface was rendered in a resolution of 2K and took about 15 minutes per frame on a standard workstation. Since it is only two wave spectrum layers and the fine detail was added in a displacement shader the grid resolution could be kept low which made it easy to work with.

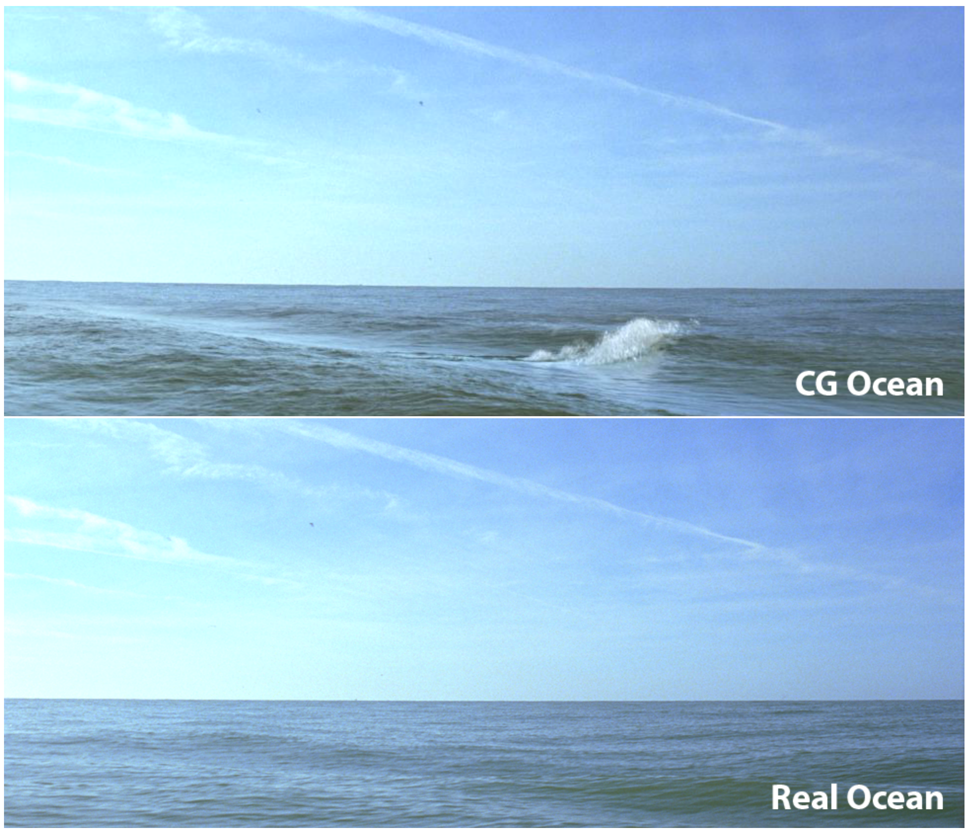

Figure 6.3: Example 3: A computer generated ocean surface compared to real footage.

# Chapter 7

# Conclusion

This report presents how to build an ocean system for visual effects out of recent research combined with own ideas to fulfill the requirements of a very flexible and usable system. The VFX ocean toolkit is designed to integrate seamlessly with Houdini and has adopted its procedural nodebased operators. My operators can therefore be combined with native Houdini operators to push the results far beyond its own limits.

The idea of stacking several layers with different scales and ocean wave spectrums has proved to be very useful. It not only decreases setup times but also takes the system to a whole new level where very specific surfaces can be generated quite easily out of reference footage. One of the other key features is the repetition removal system which makes, the otherwise quite limited approach of using an ocean wave spectrum, very useful for generating very large ocean surfaces.

Building the object-water interaction system based on ideas from the Wave Particles method proved not only to be flexible but also very efficient. Results are visible directly with hardly no simulation time which makes it possible to spend more time on finding the perfect wave look instead of waiting for the simulation to end. The use of subframes when simulating proved to be necessary when dealing with fast moving objects, which is the case most of the time. The interaction system is also very easy to modify and extend beacuse it is based on physics that have been simplified to the extent of not being very physical anymore. Simulations are great but in the visual effects industry the final images are much more important than physical accuracy.

The main strength of the toolkit is in its ability to quickly produce great results in a cost effective manner together with its flexibility when it comes to directing and controlling the result. The reason for this is mostly thanks to the tight integration with Houdini and the possibility to use existing production tools such as paint nodes to modify surface and foam directly. The toolkit has already been used in several projects with great success and therefore proven itself as a very useful visual effects ocean toolkit.

# Chapter 8

# Future work

The VFX ocean toolkit is not a complete system, actually far from it and there is much room for improvements and development of new features. The three main components work very well but could of course be optimized and developed even further. The main interest for improvements is in extending the whole system by introducing new operators to be able to handle other types of water effects. Some of the most interesting examples are

- 2D geometry based beach breaks
- Real breaking waves with foam and spray for closeup shots
- 3D fluids simulated using the graphics processing unit

The first two are interesting and certainly useful but most appealing is extending the system to handle 3D fluids simulated using the graphics processing unit. This would enable simulation of even more violent splashes and interaction with objects without the simulation times associated with 3D fluids.

# Chapter 9

# Acknowledgements

I would like to thank the following: My supervisor at Fido Film, Fredrik Limsäter for always being positive and contributing with all his visual effects experience. My academic supervisors Björn Gudmundsson and Gunnar Läthén at the Institute of Science and Technology, Linköping University. My family Ingemar, Lisbeth, Anneli and Linn Rydahl for always supporting me. All the very talented people working at Fido Film, in particular Daniel Nordlund for using his magic on my rendering passes. The team at Side FX for the amazing Houdini 3D package and their lightning fast while also very friendly support.

# Bibliography

[1] Tessendorf, J. 2002. *"Simulating Ocean Water",* In Course Notes #9 (Simulating Nature: Realistic and Interactive Techniques), Siggraph 2002, ACM Press / ACM SIGGRAPH.

[2] Tessendorf, J. 2004. *"Interactive Water Surfaces",* Game Programming Gems 4.

[3] Yuksel, C., House, D. H., Keyser, J. 2007. *"Wave Particles",* In Procedings of Siggraph 2007

[4] Larsson, M., Zalzala, J., Duda, G. 2006. *"A Procedural Ocean Toolkit",* In Conference Abstracts and Applications, SIGGRAPH 2006, ACM Press /ACM SIGGRAPH

[5] A. H. Watt *"3D Computer Graphics",* Pearson Addison Wesley, 3rd edition (1999)

[6] J. Stam *"Stable fluids",* ACM SIGGRAPH 99, pp. 121-128. (1999)

[7] R. H. Stewart *"Introduction to Physical Oceanography",* Department of Oceanography, Texas A&M University (2005)

[8] Gary A. Mastin, Peter A. Watterger, and John F. Mareda *"Fourier Synthesis of Ocean Scenes",* IEEE CG&A, p 16-23. March 1987

[9] O. M. Phillips *"On the generation of waves by turbulent wind",* Journal of fluid mechanics, 2:417-445

[10] W. J. Pierson, L. Moskowitz *"A proposed spectral form for fully developed wind seas",* Journal of geophysical research, 69:5181-5203

[11] B. Kinsman *"Wind Waves, Their Generation and Propagation on the Ocean Surface",* Dover Publications, 1984

[12] F. V. Gerstner *"Theory of waves",* Abhandlungen der Koenigl, boehmischen Gesellschaft der Wissenschaften zu Prag.

[13] R. Timman, A. j. Hermanus, G. C. Hsiao *"Water Waves And Ship Hydrodynamics, An Introduction",* Springer, Oct 1985