Methods of Deep Ocean Simulation

SANJAY SEN

Master of Science,

Computer Animation and Visual Effects

Bournemouth University

August, 2013

Contents

Abstract

This thesis analyses current methods of ocean simulation. The two models under consideration have been chosen to be the Gerstner wave model and the fft-statistical model. Using various research such as Tessendorf (2001), Fournier and Reeves (1986), Jensen and Goliáš (2008) a conceptual understanding of various ideas are broken down in order to fully develop the functionality required in designing a computer program. This is taken forward by uncovering the underlying mathematical concepts coupled with mathematical theory and oceanographic observations.
Both the fft-statistical and the Gerstner ocean model have been successfully integrated into the 3D application using C++ and the NCCA's NGL library and also has a dependency on the FFTW library in order to achieve optimized run time speeds. The fft simulation can use up to 256x256 resolution grid at real time and up to 1024x1024 waves, all entirely run off the CPU. This thesis forms the foundations to create a realistic simulation with a wide scope to extend with further functionality relevant to oceanic phenomena.

Chapter 1

Introduction and Motivation

Understanding the underlying physical phenomena of large bodies of water is an area of great interest in a number of fields such as physics, mathematics, oceanography, and with advent of visual effects, has also inspired a large part of computer graphics. Creating time evolving ocean animation and the development of efficient algorithms have numerous applications in the film and game industry. This thesis investigates current methods of implementing ocean waves that uncover two popular models: the Gerstner wave model and the statistical wave model.

The thesis initially analyses the fundamental mathematical concepts in both models, highlighting the most significant ideas that led to the code design and methodology behind developing an ocean simulation program.

The application has been written in C++ using the NCCA's NGL library under the QT Creator development platform and also has an additional library dependency on the FFTW library.

Chapter 2

Related Work

Both the Gerstner and statistical model have been outlined in “Simulating Ocean Waves” (Tessendorf 2001). This paper discusses a wide range of concepts from simulation to rendering aspects of ocean waves. Similarly “Deep Water Animation and Rendering” (Jensen and Goliáš 2008) and “Rendering Natural Waters” (Premože & Ashikhmin 2001) covers some parts of the statistical representation of ocean wave animation whilst mainly focusing on rendering to achieve effects such as accurate reflections refraction shading models. Further literature transfers this implementation on to the GPU such as Mitchell (2005).

The statistical model is considered to be a more realistic ocean model since it takes into account observations from oceanographic literature in combination with mathematical modelling. It uses a very large number of waves and special decomposition properties of the mathematical model using fast Fourier transformations. Achieving this model is far more complex than the Gerstner wave model first implemented by Fournier and Reeves (1986) in a computer graphics context. However, the Gerstner waves still have applications in areas such as gaming where photorealism may not be prime objective but efficiency may be desired. The Gerstner model can still prove to be a more appropriate model in some circumstances. This thesis focuses purely on the simulation side of oceanic animation encapsulated by the Gerstner and the statistical wave model.

Chapter 3

The Gerstner Wave Model

3.1 Gerstner Wave Theory

One of the first models to be used in computer graphics as a means of simulating an ocean surface was based off Gerstner wave theory and applied by Fournier and Reeves (1986). The model is based off a simplified version of the Navier-Stokes equation and describes the motion of individual points on an ocean surface that yields an approximate solution to their position at any time of the simulation. The water surface generated is of the trochoid family (a generalisation of a cycloid) and demonstrates the ocean to move in circular motions as a wave traverses its surface.

For a single wave passing any point on the undisturbed surface, $\mathbf{x_0} = (x_0, z_0)$ and $y_0 = 0$, the following equations explicitly express a point's new position , $\mathbf{x} = (x,z)$ in the horizontal plane, and y in the "up" vector:

$$\mathbf{x} = \mathbf{x}_0 - \frac{\mathbf{k}}{k} A \sin(k \cdot \mathbf{x}_0 - \omega t) \qquad (1)$$

$$y = A \cos(\mathbf{k} \cdot \mathbf{x}_0 - \omega t\ ) \qquad (2)$$

where A is the amplitude (size of a wave), **k** is the wavevector (the direction and magnitude of travelling wave) and omega is the frequency and is related to the wavevector.

Extending this idea, a number of N such waves profiles can be generated from a set of wavevectors, **k**i, amplitudes Ai, frequencies omega_i and phases Phi_i, where 0 <= i < N.

$$\mathbf{x} = \mathbf{x}_0 - \sum_{i=1}^{N} \frac{\mathbf{k}_i}{k_i} A_i \sin(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t\ + \phi_i) \qquad (3)$$

$$y = \sum_{i=1}^{N} A_i \cos(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t\ + \phi_i) \qquad (4)$$

Hence by choosing a suitable set of values for the above parameters, one can generate much more complex profiles. In particular, the values of the wavevector and corresponding amplitude must be chosen appropriately. The model predicts that a wave profile will alter its surface structure depending on the value of a dimensionless measure kA, (where k is the magnitude of the wavevector). The following figure shows the wave profile of differing wavevectors and amplitudes in relation the result of kA.

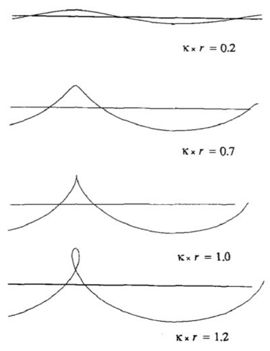


Figure 1a) shows the varying Gerstner waves with differing kA (Fournier & Reeves 1986)
(Note that the K x r is equivalent to kA in this thesis)

In figure 1a), the wave profile that results with kA = 0.7 is a seemingly standard wave that expected to be observed in the sea. The trough is relatively smooth while the peak is sharp. By pushing kA higher, the wave gets sharper at the tip until the wave intersects itself forming a loop instead of a peak. The threshold value is kA = 1. If kA is below this, the waves are smoother with peaked tips. If kA = 1, the wave profile attains its sharpest potential peak. With kA > 1, then the looping artefact will occur. The only overlooked variable at this point is omega which is described in the following section.

3.2 Omega and the Dispersion Relation

The way in which waves animate are dependent on the conditions of the ocean. In this case, the dispersion of (omega) can fall in to two main categories. The first case is when the ocean waves are occurring in deep water and the second is when the ocean floor is significantly close to the surface of the water (i.e. shallow water near a shore) where the distance of the ocean floor will start having an effect of the wave behaviour.

More accurately, the terms “deep” and “shallow” are determined by the distance of the sea floor in comparison to the wavelength of the average wave profile. If the wavelengths of the wave profiles are small in relation to the distance from the ocean floor then the ocean is considered “deep”, otherwise the shallow dispersion model should be used.

For waves in deep ocean conditions, the dispersion relation is as follows:

$$\omega^2(k) = gk \tag{5}$$

where g is the gravitational constant, g = 9.8ms-2. For shallow waves where the distance D is comparable to the ocean wave's wavelength, the following holds true:

$$\omega^2(k) = gk\tanh(kD) \tag{6}$$

where D is the distance of the water surface from the ocean floor (Oceanworld 2007). Note that for very large values of D, the tanh term converges to 1 and hence converges to the deep ocean dispersion relation. For the purposes of the simulation discussed in this thesis, the deep ocean dispersion relation will suffice and is used in both the Gerstner and the statistical ocean implementation.

Chapter 4

The Statistical Wave Model and the Fourier Transformation

4.1 Calculating the Height Field

The alternative approach to modelling an ocean simulation is to use a statistical model to express the height of the ocean at any point h(**x**,t) as a random variable of horizontal position and time. This model uses experimental observations that are in line with oceanography to create a more realistic model of the ocean with waves driven by wind in a particular direction and is of the form of a Fourier transformation.

The following equation shows the representation of the height field over the space of the horizontal position.

$$h(\mathbf{x}, t) = \sum_{k} \tilde{h}(\mathbf{k}, t) . \exp(i\mathbf{k}.\mathbf{x}) \tag{7}$$

A logical deduction to be made is that although equation (7) uses complex exponentials, the model must ensure that the height field at any **x** yield real values. That is, that h(**x**,t) = h*(**x**,t) i.e. the complex part is zero. This will set conditions on the form that h(**k**,t) can take, which will eventually determine the simulation. As highlighted in “Numerical Recipes in C++” (Press, Teukolsky, Vetterling & Flannery 2002), for h(**x**,t) to be a real function, h*(**k**,t) = h(-**k**,t).

Similar to the Gerstner representation, **k** are wavevectors that represent the direction of traversing wave. Differing from the Gerstner wave model however are the methods of choosing the set of wavevectors. The set of **k** are chosen in a specific way depending on the number of waves chosen in their simulation. The values that are assigned to k are chosen in the following way.

$$\mathbf{k} = (k_x, k_z)$$

$$k_x = \frac{2\pi n}{L_x} \tag{8}$$

$$k_z = \frac{2\pi m}{L_z}$$

Where n and m are integer values in the range

$$-N/2 \leq n < N/2$$

$$-M/2 \leq m < M/2 \tag{9}$$

The h(**k**,t) are the time dependent Fourier components that affect and animate the height field over time. What h(**k,**t) is and how we find out its form for each **k** involves a number of steps, first starting with the calculation of a wind, and **k** dependent constant for every wavevector. This parameter is called the Phillips spectrum and is given by the following:

$$P_h(\mathbf{k}) = A\frac{exp\left(-1/k^2L^2\right)}{k^4}\left|\hat{\mathbf{k}}\cdot\hat{\mathbf{w}}\right|^2 \tag{10}$$

where k denotes the magnitude of **k**, A is a numerical constant and L=V^2/g is size of the largest wave, V is the wind speed and g is the gravitational constant. Note the last term in the Phillips spectrum uses the dot product of the wavevector with the wind (both are unit vectors) takes the magnitude and squares it. Thus the waves which are perpendicular to the wind to be cancelled out have a zero valued Phillips spectrum while vectors in the same direction will be fully maintained. One should also note that due to the magnitude squared term, even waves in the opposite direction to the wind are retained since Ph(**k**) is an even function of **k**.

With the Phillips spectrum calculated, the Fourier amplitudes can be calculated directly in the Fourier domain (in this case, the k space) in the following form:

$$\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}}(\xi_r(\mathbf{k}) + i\xi_i(\mathbf{k}))\sqrt{P_h(\mathbf{k})} \tag{11}$$

where Zr and Zi are randomly generated gaussian numbers with mean = 0 and standard deviation = 1. With the the Fourier height amplitudes calculated, h(**k**,t) for any particular wavevector can be calculated by the following expression:

$$\tilde{h}(\mathbf{k},t) = \tilde{h}_0(\mathbf{k})\exp(i\omega(k)t) + \tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t) \tag{12}$$

At this point it is very important to understand the significance of why this particular form has been chosen. Recall from earlier in this section the assertion that for h(x,t) to be real, h*(**k,**t) = h(-**k**,t). By inserting the above definition of h(k,t) from equation (12) one finds that the condition is satisfied and this will guarantee that the eventual solution of h(**x**,t) will be real. In a physical sense, equation (12) represents a wave that propagates in a certain direction will have another wave propagating in the opposite direction, conceptually making sense of h(**k**,t).

Finally the terms of all the h(**k**,t) are summated over all **k**, thus yielding a real solution to the height field at a certain time t (Tessendorf 2001).

4.2 Choppiness

When calculating the height field of the statistical wave model, the outcome generates smooth waves with regularly rounded tops. It is observed, however, that waves under significant wind factors will produce “choppy” waves, meaning that the wave tips are sharper while the troughs are wider. In terms of the gradient, this implies that peaks will have greater gradients while the troughs have smaller gradients. According to Tessendorf (reference), the Lie Transform technique (reference) extrapolated to two dimensions can be used to create choppy waves. This entails a shift by **D** on **x** = (x,z), while calculating the height field from the statistical model. By shifting the points in the x-z plane the desired effect can be brought about through the above mentioned transformation. The form of **D(x,**t) is as follows:

$$\mathbf{D}(\mathbf{x},t) = \sum_k -i\frac{\mathbf{k}}{k}\tilde{h}(\mathbf{k},t).\exp(i\mathbf{k}.\mathbf{x}) \tag{13}$$

$$\mathbf{x} = \mathbf{x} + \lambda\mathbf{D}(\mathbf{x},t) \tag{14}$$

Finally this D(**x**,t) can be used to shift **x** by some factor lambda as seen in equation (14). Note that the actual calculation for equation (13) is a variant of the height field and can hence may not need to be re-calculated once completing the computation of h(**x**,t).

Chapter 5

General Implementation

The ocean simulation program underwent many iterations to appropriately model the given problem. In the case of both the Gerstner and statistical wave model, they are mathematical descriptions of how the surface of a body of water will change over time and have been derived as approximate or realistic extensions to the Navier-Stokes equation. Hence the design followed that the ocean element would be abstracted as a grid of points displaced equally and evenly around the x-z axis. Each point will be displaced according to the chosen algorithm at every time frame. In theory the grid of points could be chosen arbitrarily. Having the Ocean class in place, it would require waves to act on it.

The first implementation abstracted the type of waves, i.e the FFTWaves and GerstnerWaves classes. Since many such waves exist in any desired simulation, a "wave manager" class was formed to deal with the management of each specific wave type. For example, the GerstnerWaveSet class contained an std::vector of GerstnerWaves as well as all of the necessary member functions to mutate and return various attributes. A similar concept extends to FFTWaves and the FFTWaveSet class.

Hence, the overall pipeline encompassed the Ocean class which contained an instance of wave sets for each wave type, which in turn managed the wave calculations. These constructed wave managers would contain arrays of individual and most fundamental wave objects of their respective type (i.e GerstnerWaves or FFTWaves).

Chapter 6

Implementation 1: The Gerstner Wave Model

The design of the Gerstner implementation was simply outlined the previous section and once implemented, did not require many further alterations due to its simplicity. As will be seen in the fft implementation, the mathematical properties of the statistical model is its ability to decompose a multitude of wave calculations into an optimised problem. The Gerstner wave model has no such property and the calculations from waves are required to be called at every grid point for every created wave in the waveset.

The GerstnerWaveSet object is created with in the Ocean instance and initialized to contain an std::vector of sub units called GerstnerWaves. A single Gerstner wave is defined by and constructed with the parameters: amplitude, wavevector and phase. These three attributes impact the way the wave will contribute to the overall ocean. It is the functionality of the GerstnerWaveSet to create large numbers of individual Gerstner waves that will populate the std::vector of waves and create a diverse wave set. Since there is no specified way to create a wave set (unlike the fft model) the GerstnerWavSet uses randomness to create large numbers of unique wave vectors. Upon design, the randomness had to be controlled to some extent so that a user could attempt to customize the kind of ocean desired. More important to consider is the values of kA that could be potentially achieved. Recall from section 3.1, figure 1a) that in order to avoid creating self intersecting vertically looping waves, the dimensionless quantity had to be less than 1. Using this constraint, the random values has to be generated within desired ranges. The only completely random attribute in the GerstnerWaveSet would be the wavevector which used gaussian distribution of random draws to decide k_x and $k_{z.}$. Once this vector is normalised a unit vector is created with a random direction. The GerstnerWaveSet manages a range of values to choose from randomly. The Gerstner waves will be created by the wave set. A minimum and maximum wave magnitude, amplitude and phase are passed to create waves of a random yet controlled nature.

Each individual Gerstner wave has all the attributes required in order calculate its contribution to the ocean grid provided it is passed the specific points on the grid. Since this happens for every wave in the wave set, the GerstnerWaveSet is passed the the grid points sequentially, passing this data on to each of the GerstnerWaves, invokes them to return their displacement contribution to the the GerstnerWaveSet that sums all the components together and changes the current ocean grid position using the equations (3) and (4) from section 3.1.

Various other management tasks have been allocated to the GerstnerWaveSet class such as the removal and addition of new waves to the set as well as starting a new simulation. This functionality can filter down from the UI in order to give the user a great deal of control of spawning user defined waves.

One thing to note is the efficiency side of the Gerstner implementation. By taking the above mentioned procedure to update the wave set, the result is effectively a brute force method. The Gerstner wave model has no special mathematical properties that can allow for faster summations to occur. This therefore creates a limitation on the number of waves that can be executed in a simulation. The statistical fft version described in the next section will reveal that due to the nature of the mathematical problem, a very significant efficiency can be used to boost simulation speed to allow for the possibility of hundreds of thousands and even millions of waves.

Chapter 7

Implementation 2: The Statistical Wave Model

7.1 Initial implementation : The Brute Force Method

The FFT implementation initially did not incorporate any usage of an FFT library. The FFT library would be used only as a means of speed gain and would make no difference to a simulation aside from efficiency. Instead, a brute force method was first used to ensure that the pipeline and mathematical calculations were accurate to the theory. This brute force method could also then be used as a cross reference when using the FFT method, resulting in the same numerical values. However, certain requirements were necessary to ensure that an FFT method would integrate at a later stage. One such consideration was to set up the ocean grid such that if the wavevectors were being generated from equation (7) then the ocean grid would be constructed such that the ocean grid is generated at the discrete points (nL_x /N, mL_z/M). This is due to the way the that the FFT method would eventually optimise the calculation of the grid points over time but is best to be initially set up from the offset. This configuration of the ocean grid will have no effect on the Gerstner implementation since the Gerstner calculation can in theory act on any set of points provided their original positions are known to the Gerstner implementation. More on the FFT library integration will be discussed later. Similar to the final Gerstner model, the brute force method iterates over every point on the ocean grid, calculating the sum of h(k,t)exp(ik.x) for the entire wave set.

7.2 Implementing the height field

As mentioned above, the statistical model of the wave ocean model are determined by the dimensions of the ocean grid. The only inputs required are the grid resolution and the grid length. This will determine a corresponding set of wavevectors given by equation (8). The FFTWaveSet object contained in ocean sets up the necessary wave set such that a std::vector array of FFTWaves are populated uniquely. Following the statistical model, it then follows that the unique wavevector for every FFTWave object can calculate its own Phillips spectrum and h0(k) in the Fourier space. Note that these quantities are independent from time and only need to be initialized on the object's construction.
On the execution of a time update, the time dependent Fourier coefficients need to be worked out. This quantity is dependent on both h0(**k**) and h0*(-**k**) for every wave at any given time. This complication led to an interesting problem that for every wavevector, **k** with attribute h0(**k**), one needs to know about the corresponding "-**k**" and use it's h0*(-**k**) in order to calculate h(**k**,t) to then be summed over all **k** to work out h(**x**,t), the height at any given horizontal position **x**. The converse is also true for the -**k**th vector. A simple solution to this problem lies in considering the form of h(**x**,t) for both instances of **k** and -**k**.

For +**k** , the h(x,t) expands to the following:

$$h(\mathbf{x},t) = \left(\tilde{h}_0(\mathbf{k})\exp(i\omega(k)t) + \tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t)\right).\exp(i\mathbf{k}.\mathbf{x}) \qquad (15)$$

For -**k**, the following equation is obtained:

$$h(\mathbf{x},t) = \left(\tilde{h}_0(-\mathbf{k})\exp(i\omega(k)t) + \tilde{h}_0^*(\mathbf{k})\exp(-i\omega(k)t)\right).\exp(-i\mathbf{k}.\mathbf{x}) \quad (16)$$

Note that the gaussian random draws have the notation dependency on k or -k. This can be misleading without the explicit notation. Zr(k) and Zi(k) are the random numbers generated on construction with association to the **k**th wavevector. It follows that similarly Zr(-k) and Zi(-k) are the random numbers from the -**k**th wavevector. However, by omitting this notation, Zr and Zi have no indication to the wavevector they are associated to and may be open to misinterpretation.

Note that the first term in equation (15) and the second term in equation (16) have only a dependency on +**k**. Similarly the second term in equation (15) and the first term in equation (16) only have a dependency on -**k**. For every time update, rather than trying to "find" the corresponding **-k** to each **k**, the following contributing terms can be returned to the FFTWaveSet from each wavevector:

$$\Delta_k h(x,t) = \tilde{h}_0(\mathbf{k})\exp(i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) + \tilde{h}_0^*(\mathbf{k})\exp(-i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) \quad (17)$$

When considering the contribution received by the **-k**th term (where ever in the array it may be), the following term is returned to the FFTWaveSet:

$$\Delta_{-k} h(x,t) = \tilde{h}_0(-\mathbf{k})\exp(i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) + \tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) \quad (18)$$

Since the commutative nature of the h(**x**,t) summation, the first term from (17) and the second term (18) would match up to form (19) written below. Similarly the remaining two terms will match up to create (20).

$$h(\mathbf{k},t)\exp(i\mathbf{k}.\mathbf{x}) = \tilde{h}_0(\mathbf{k})\exp(i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) + \tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) \quad (19)$$

$$h(-\mathbf{k},t)\exp(-i\mathbf{k}.\mathbf{x}) = \tilde{h}_0(-\mathbf{k})\exp(i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) + \tilde{h}_0^*(\mathbf{k})\exp(-i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) \quad (20)$$

These are exactly equivalent to working out the h(**x**,t) contributions from **k** and **-k**. This is true for the sum over all **k** and results in the correct real values outputting for the height field once the entire summation is finished after every time step.

7.3 Choppy Scale

The initial implementation calculates the scale of choppiness through an analogous method to the height field calculation. Inspecting the form of D(x,t), (the choppiness factor) for both k and -k yield the following observations:

D(x,t) for k

$$\Delta\mathbf{D}(\mathbf{x},t) = \left[-i\tilde{h}_0(\mathbf{k})\exp(i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) - i\tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x})\right]\hat{\mathbf{k}} \quad (21)$$

D(x,t) for -k

$$\Delta\mathbf{D}(\mathbf{x},t) = \left[i\tilde{h}_0(-\mathbf{k})\exp(i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) + i\tilde{h}_0^*(\mathbf{k})\exp(-i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x})\right]\hat{\mathbf{k}} \quad (22)$$

from the above equations, the previous method used for summation over k would not directly work.

Notice that when trying to match corresponding terms to understand what each D(x,t) contribution needs to be returned to the FFTWaveSet, the coefficients of the first term in (21) and the last term (22) are -i and i respectively. The coefficients of the first term of equation (22) and the last term of (21) however are i and -i. The signs have switched and this added complication needs to be dealt with in order to ensure that D(**x**,t) is calculated correctly when sum over all **k**. The method adopted involved defining a boolean that split the grid in to two halves, one being **k** and the other being **-k**.

Although there is no such inherent distinction of k to -k since every wavevector has a negative wavevector, but in order to allow the program to know which combinations of the exponential terms to use, a method was created to segregate the wavevectors into "+k wavevectors" and "-k wavevectors". Since the wave set is created evenly around the grid (see equation (8)) one can arbitrarily choose a certain axis to divide to wavevectors into the two categories. Each wavevector then stores the four complex exponential variations (arrays of complex numbers) written in the below expressions at every time frame

Combinations for +**k**

$$-i\tilde{h}_0(\mathbf{k})\exp(i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) \tag{a}$$

$$+i\tilde{h}_0(\mathbf{k})\exp(i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) \tag{b}$$

$$-i\tilde{h}_0^*(\mathbf{k})\exp(-i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) \tag{c}$$

$$+i\tilde{h}_0^*(\mathbf{k})\exp(-i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) \tag{d}$$

Combinations for -**k**

$$-i\tilde{h}_0(-\mathbf{k})\exp(i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) \tag{e}$$

$$+i\tilde{h}_0(-\mathbf{k})\exp(i\omega(k)t)\exp(-i\mathbf{k}.\mathbf{x}) \tag{f}$$

$$-i\tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) \tag{g}$$

$$+i\tilde{h}_0^*(-\mathbf{k})\exp(-i\omega(k)t)\exp(i\mathbf{k}.\mathbf{x}) \tag{h}$$

By inspecting the requirements of equation (21) and (22), if the wave is defined to be a "negative wavevector", it sums expressions (f) and (g) and returns this contribution to h(x,t). If the wave is a "positive wavevector" it sums the expressions (a) and (d). The final result of this computation yields a real number which scaling an overall direction vector for the ocean point to displace to.

7.4 Efficiency: Reimplementation of the FFTW library

The brute force method designed a means of creating the statistical model of ocean waves. The next step was to improve the efficiency of the program. In the brute force method, every point required to perform a summation of all the Fourier coefficients multiplied by an exponential based on the current ocean grid's spatial position. Since the set up required that there be an equal number of wavevectors to spatial points and in general, the ocean is designed as a square grid of evenly spaced points, the ocean grid contains NxN points on the surface and the corresponding wavevectors also contains NxN wavevectors. This means that the overall calculation of iterating through every grid point and calculating every wave's h(x,t) contribution ends up with an order of calculation of N^4 with the use of a double “for loop”. Hence the order of calculations as N increases becomes computationally very expensive. However, since the equation of h(x,t) can be interpreted in the form of a Fourier transformation, the Fast Fourier Transformation algorithm can be applied to speed up the order of calculations. The order of calculations for an analogous 1D problem of N^2 is reduced to an order of (N ln(N)), see figure 2a). The FFT algorithm is most efficient when the ocean grid (and therefore) the wavevector set are exact powers of 2. The chosen library to perform the task of the FFT was the FFTW or “Fastest Fourier Transformation in the West”. In order to appropriately use the FFTW library, many elements of the brute force design needed to be modified.

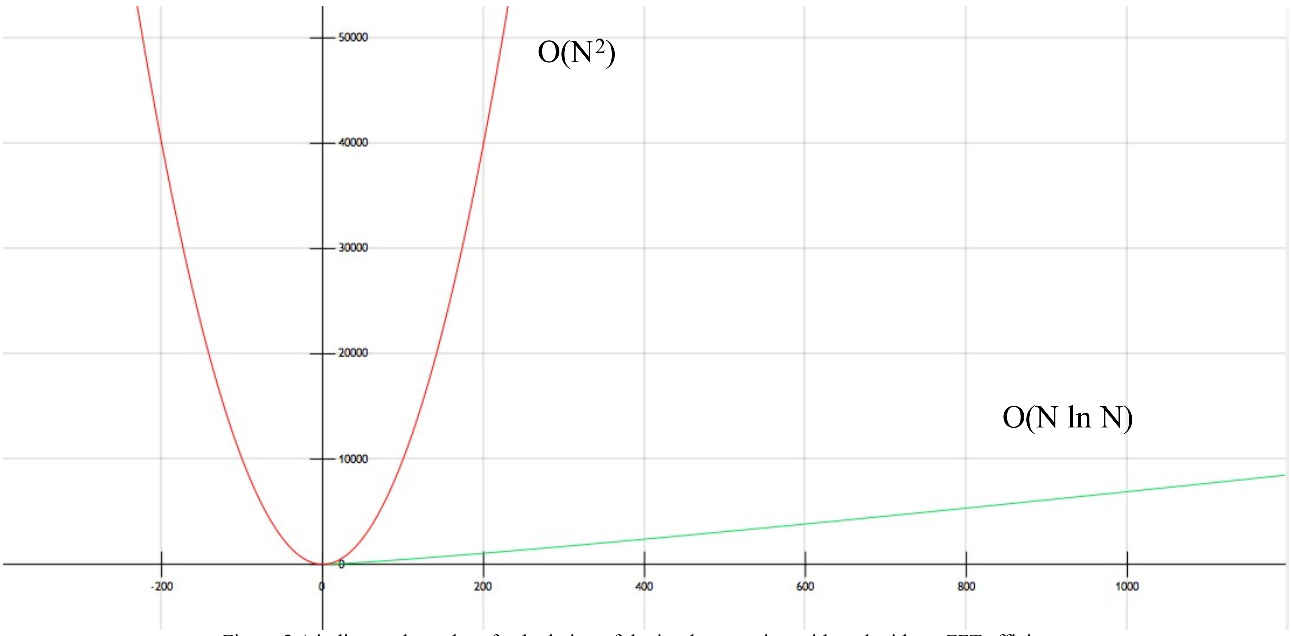


Figure 2a) indicates the order of calculation of the implementation with and without FFT efficiency

equation(7) is a summation over k, where k is expressed in terms of n and m. Hence, equation (7) can be rewritten as a double summation over n and m. in terms of programming convenience, rather than use -N/2 <= n < N/2 and M/2<=m<M/2, the application shifts these indices to two new variables. The double summation, new dummy variables and the corresponding wavevectors are indicated as follows:

$$0 \leq n' < N$$
$$0 \leq m' < M \tag{23}$$

$$(k_{n'}, k_{m'}) = \left(\frac{(2\pi n' - N\pi)}{L_x}, \frac{(2\pi m' - M\pi)}{L_z}\right) \tag{24}$$

$$h'(x,z,t) = \sum_{m'=0}^{M-1}\sum_{n'=0}^{N-1} \tilde{h}'(n',m',t)\exp\left(\frac{ix(2\pi n' - N\pi)}{L_x} + \frac{iz(2\pi m' - M\pi)}{L_z}\right) \tag{25}$$

By rewriting the double summation and manipulating the expression, the following compounded one dimensional expression can be used to interpret h(x,t) (Lantz 2013):

$$h''(x, m', t) = (-1)^x \sum_{n'=0}^{N-1} \tilde{h}'(n', m', t) \exp\left(\frac{i2\pi n' x}{N}\right) \tag{26}$$

$$h'(x, z, t) = (-1)^z \sum_{m'=0}^{M-1} h''(x, m', t) \exp\left(\frac{i2\pi m' z}{M}\right) \tag{27}$$

Note that the extraction of the -1^x and -1^z from the exponent terms brings the expression in the form of an fft that the FFTW library can use. Given that x and z are always integers and are required by the FFT process to be in powers of 2, these terms will yield +1 or -1 (Bale 2010).

From equation (26) the input of data for the compounded one dimensional Fourier transformations are the Fourier coefficients h(n',m',t). This is the equivalent to h(**k**,t) from the original height field equation (7). The equations in terms of n' and m' also reflect in the code. All the data arrays were co-migrated to exist within the Ocean class for initial simplicity. The Ocean class contains arrays of n' and m', the wavevectors k(n', m'), the phillips spectrum and all other necessary arrays. The arrays were created as two dimensional std::vectors of specific types (e.g. std::vector of std::vector<float> was used to create the 2D data array for storing all calculated phillips spectral data). In this way, access to all attributes and specific indices n' and m' such that phillips_spectrum_Array[2][4] would access the Phillips spectrum corresponding to n' = 2 and m' = 4. In this way all the necessary data was reformatted into 2D arrays rather than the one dimensional unpacked vector from the original brute force method. With the redesigned data structures in place, a similar brute force implementation was remade in order to make sure that the new calculations correctly replicated the ocean simulation before the last stage of optimization through the FFTW library.

The final stage involves the pipeline between the program inputs and the FFTW library to ultimately give the output. An understanding of the compounded one dimensional representation shows that the initial input is an array of the Fourier components, h(n',m',t). Recovering this was not as easily achieved as when using the brute force method. The Fourier coefficients needed to be specifically identified for all wavevectors. This problem is easily overcome when considering the periodicity of the problem for any **k** represented by n' and m', its corresponding **-k** can be found by the following procedure. Taking n' and m', and negate them. This will alter the range which n' and m' should be in. Due to the periodicity of the problem, an equivalent -**k** can be found by adding N (the size of the grid) to each component -m' and -n' and take the modulo of this number with N and the -**k** value has been retrieved for any k in terms of the index m and n.

Rather than performing a compounded set of 1D Fourier transformations the final implementation uses the functionality inbuilt to FFTW to perform a 2D inverse discrete Fourier Transformation on all the Fourier components such that the input array of Fourier coefficients in the Fourier space are transformed to the corresponding height field output array in the time domain for all discrete points on the x-z plane.

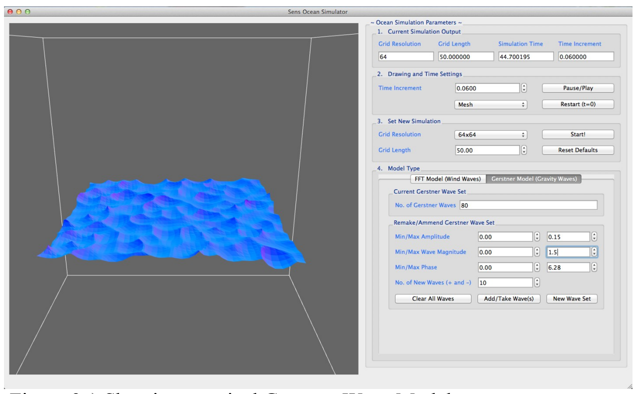
Although the calculation for the choppiness is nearly identical to the method in order to create the height field, the output erroneously produced purely complex numbers. This is possibly due to various complexities that may arise when considering the possible scenarios that the wavevector may be under. The incorrect version of the choppiness created when using the complex terms leads to (as expected) a convergence and divergence of grid points around the origin.

From a design perspective, the Ocean class containing numerous arrays of the necessary FFT attributes is not a desired choice. These arrays could be representative of class member data of a fundamental FFTWave unit, as originally desired. The FFTWaveSet would then manage the array of FFTWaves, as originally planned. However, with the code working successfully within the Ocean class, later development could easily allow the migration of the data into better abstracted and incapsulated classes, allowing the ocean class to become as minimal as possible.

Chapter 8

Application and Results

The project has successfully created two ocean models based on the Gerstner and Statistical wave implementation. From an animation simulation perspective both versions create outputs that could, if rendered appropriately, be visually understood and applied into a film or game context. This application has mainly focused on the way the surface representation of an ocean would evolve in time. In this regard, the project has achieved the goals that it originally set out to.



~ Ocean Simulation Parameters ~

1. Current Simulation Output

Grid Resolution: 64 | Grid Length: 50.000000 | Simulation Time: 44.700195 | Time Increment: 0.060000

2. Drawing and Time Settings

Time Increment: 0.0600

Mesh

3. Set New Simulation

Grid Resolution: 64x64

Grid Length: 50.00

4. Model Type

Current Gerstner Wave Set

No. of Gerstner Waves: 80

Remake/Ammend Gerstner Wave Set

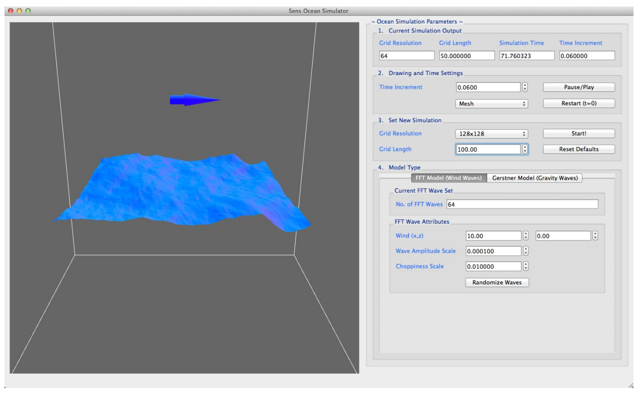
Min/Max Amplitude: 0.00 | 0.15

Min/Max Wave Magnitude: 0.00 | 1.5

Min/Max Phase: 0.00 | 6.28

No. of New Waves (+ and -): 10

Figure 3a) Showing a typical Gerstner Wave Model



~ Ocean Simulation Parameters ~

1. Current Simulation Output

Grid Resolution: 64 | Grid Length: 50.000000 | Simulation Time: 71.760323 | Time Increment: 0.060000

2. Drawing and Time Settings

Time Increment: 0.0600

Mesh

3. Set New Simulation

Grid Resolution: 128x128

Grid Length: 100.00

4. Model Type

Current FFT Wave Set

No. of FFT Waves: 64

FFT Wave Attributes

Wind (x,z): 10.00 | 0.00

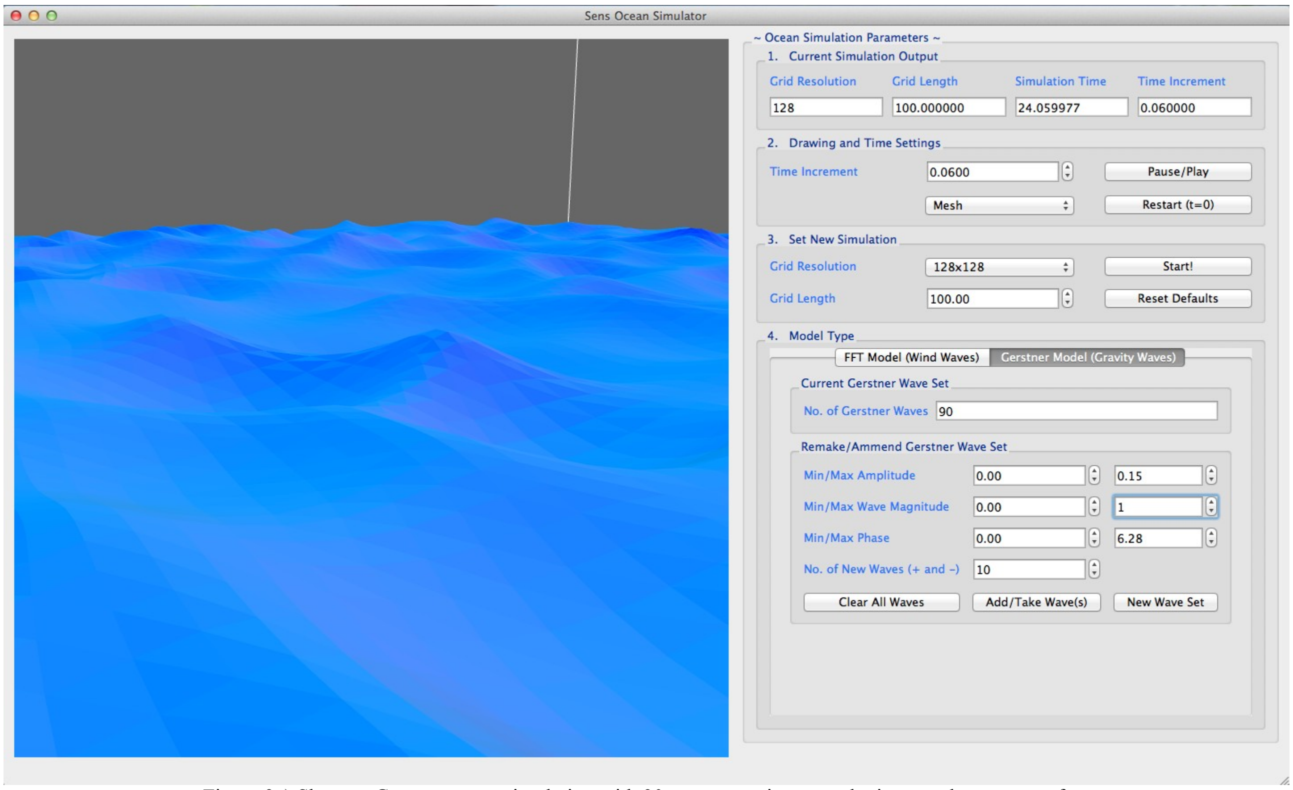
Wave Amplitude Scale: 0.000100

Choppiness Scale: 0.010000

Figure 3b) showing a typical fft simulation of

Figures 3a) and 3b) show two typical outputs of the ocean simulation. Even in the absence of a realistic shader it is possible to understand the structures to be that of an ocean surface. When animating at real time speeds, the application also outputs correct wave motions. Where applicable in the wind generated statistical model, the ocean is dominated by waves of the orientation parallel to the wind direction vector. The FFT model can run a 128x128 resolution ocean at real time on a 2.66GHz Intel Core Duo processor with 4GB memory. The simulation can begin to slow down from real above this size grid size but can still output the program with a slight time overhead at resolutions of 256x256 and 512x512. The simulation can even run for 1024x1024 (i.e. over a million waves and grid positions) with out crashing the program. Putting this in to context, feature films such as *Titanic* and *Waterworld* used grid sizes of 2048x2048 to create the ocean scenes. On a better processor, it is possible that larger grid sizes may still yield realtime results

Figure 3c) and 3d) below show the ocean simulation from a horizon perspective. Although the statistical model has proven to be a more realistic model of ocean simulation, the problem is entirely specified by certain fixed parameters, i.e. the grid resolution, grid length and wind. One benefit that has arisen from the Gerstner wave implementation is that the waves produced can be much less constrained and therefore much more customizable. Using the provided UI, a user can easily create one dominant wave that will serve as the main wave direction. Upon this, smaller user defined waves can be produced, thus keeping the main wave direction fixed but then super-positioning many smaller waves that generate randomness in the application. No such “per wave” functionality exists for the fft representation. One can only change the wind parameter, the grid resolution parameter. Both have unique styles that may be appropriate for a certain purpose.


Sens Ocean Simulator

~ Ocean Simulation Parameters ~

1. Current Simulation Output

Grid Resolution	Grid Length	Simulation Time	Time Increment
128	100.000000	24.059977	0.060000

2. Drawing and Time Settings

Time Increment: 0.0600
Mesh

3. Set New Simulation

Grid Resolution: 128x128
Grid Length: 100.00

4. Model Type

Gerstner Model (Gravity Waves)

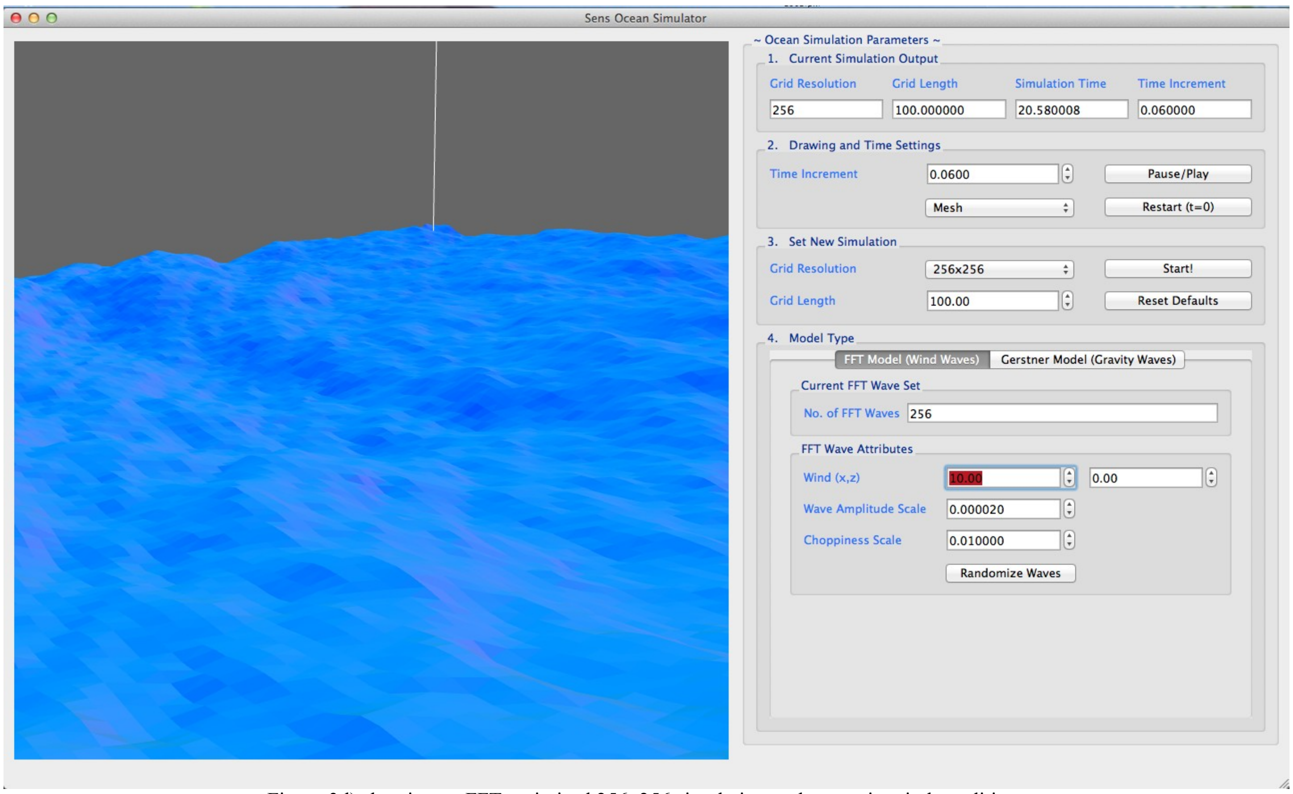
Current Gerstner Wave Set

No. of Gerstner Waves: 90

Remake/Ammend Gerstner Wave Set

Min/Max Amplitude: 0.00 / 0.15
Min/Max Wave Magnitude: 0.00 / 1
Min/Max Phase: 0.00 / 6.28
No. of New Waves (+ and -): 10

Figure 3c) Shows a Gerstner wave simulation with 90 waves causing perturbations on the ocean surface


Sens Ocean Simulator

~ Ocean Simulation Parameters ~

1. Current Simulation Output

Grid Resolution	Grid Length	Simulation Time	Time Increment
256	100.000000	20.580008	0.060000

2. Drawing and Time Settings

Time Increment: 0.0600
Mesh

3. Set New Simulation

Grid Resolution: 256x256
Grid Length: 100.00

4. Model Type

FFT Model (Wind Waves)

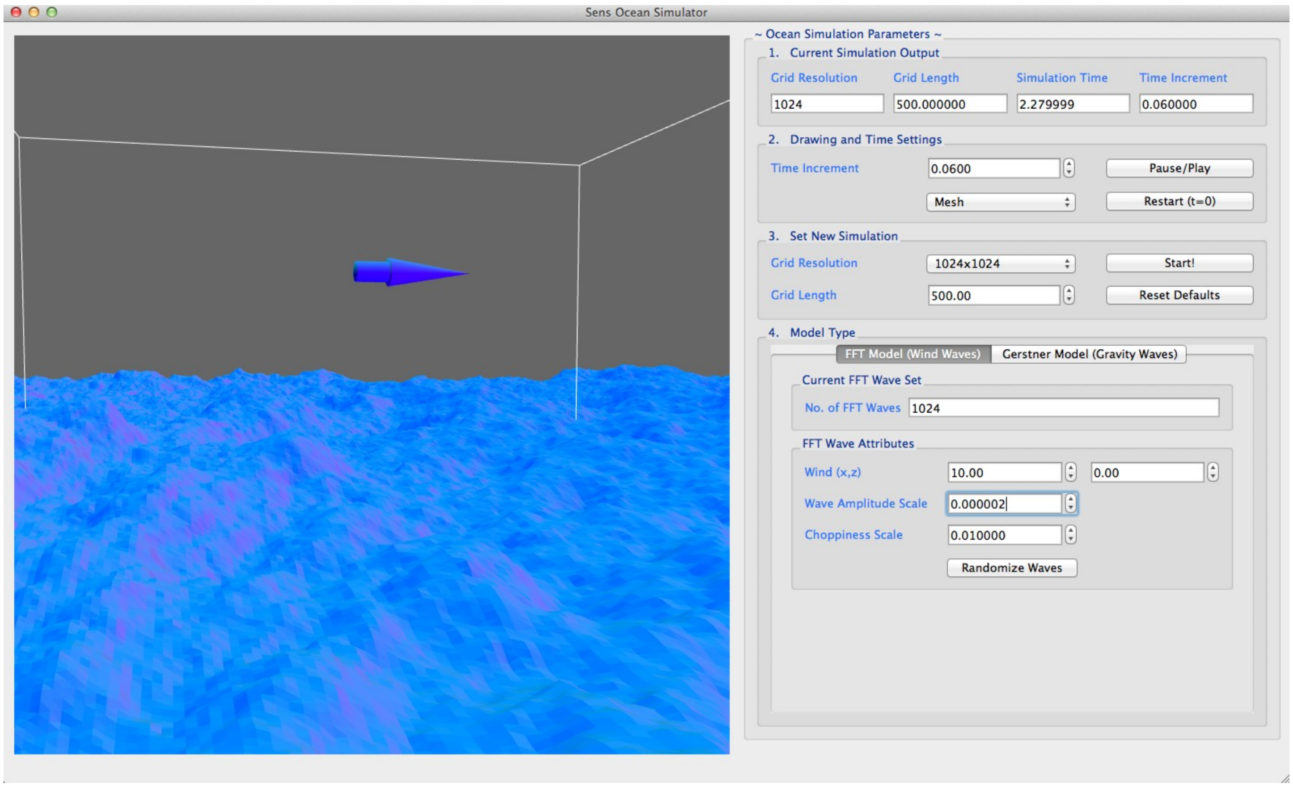
Current FFT Wave Set

No. of FFT Waves: 256

FFT Wave Attributes

Wind (x,z): 10.00 / 0.00
Wave Amplitude Scale: 0.000020
Choppiness Scale: 0.010000

Figure 3d) showing an FFT optimised 256x256 simulation under certain wind conditions


Sens Ocean Simulator

~ Ocean Simulation Parameters ~

1. Current Simulation Output

Grid Resolution	Grid Length	Simulation Time	Time Increment
1024	500.000000	2.279999	0.060000

2. Drawing and Time Settings

Time Increment: 0.0600
Mesh

3. Set New Simulation

Grid Resolution: 1024x1024
Grid Length: 500.00

4. Model Type

FFT Model (Wind Waves) | Gerstner Model (Gravity Waves)

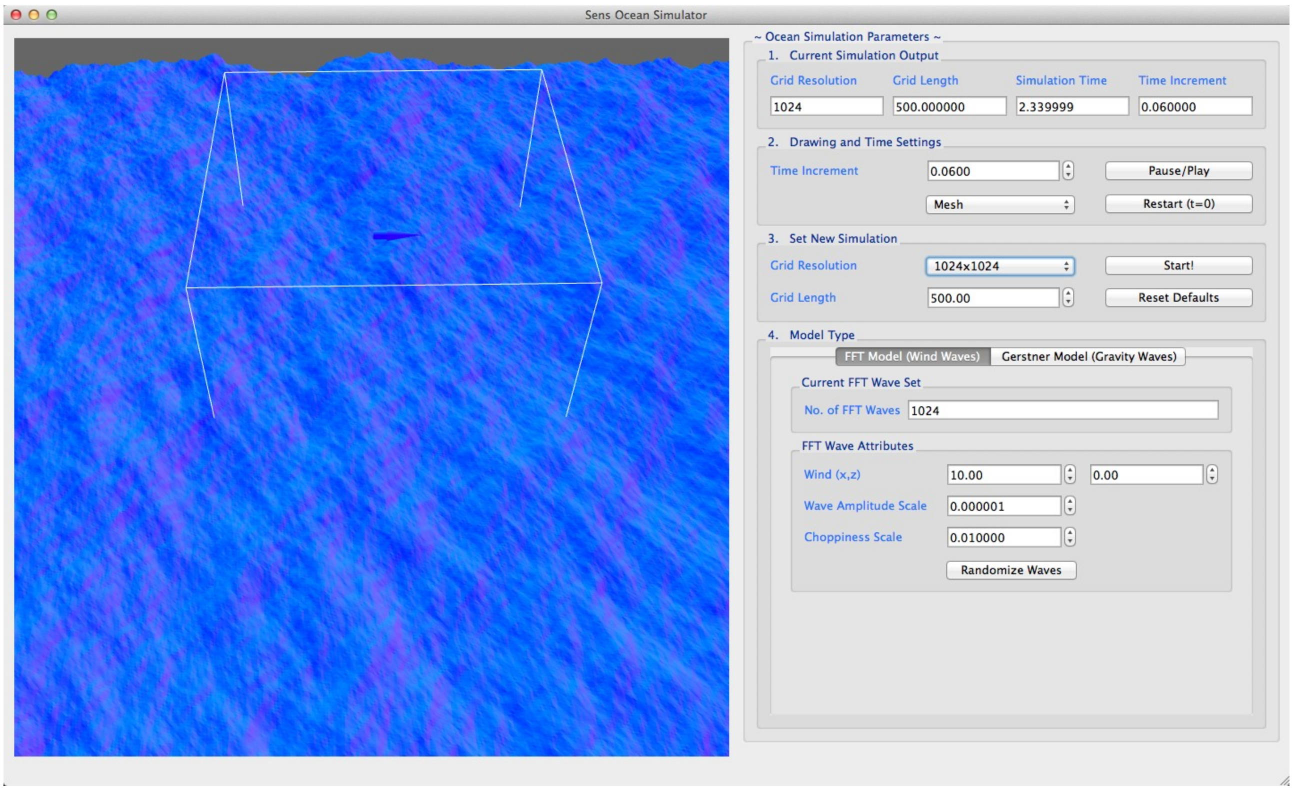
Current FFT Wave Set

No. of FFT Waves: 1024

FFT Wave Attributes

Wind (x,z): 10.00 0.00
Wave Amplitude Scale: 0.000002
Choppiness Scale: 0.010000

Figure 3e) (above) and 3f) (below) showing the fft-ocean with over a million waves and grid points


Sens Ocean Simulator

~ Ocean Simulation Parameters ~

1. Current Simulation Output

Grid Resolution	Grid Length	Simulation Time	Time Increment
1024	500.000000	2.339999	0.060000

2. Drawing and Time Settings

Time Increment: 0.0600
Mesh

3. Set New Simulation

Grid Resolution: 1024x1024
Grid Length: 500.00

4. Model Type

FFT Model (Wind Waves) | Gerstner Model (Gravity Waves)

Current FFT Wave Set

No. of FFT Waves: 1024

FFT Wave Attributes

Wind (x,z): 10.00 0.00
Wave Amplitude Scale: 0.000001
Choppiness Scale: 0.010000

Chapter 9

Conclusion

9.1 Summary

The thesis thus far has reviewed methods of ocean simulation. Categorising the main methods, the theory of Gerstner and statistical wave models were reviewed in detail in order recreate a realistic model. Having proceeded with attempting to implement these ideas, the following outcomes were achieved:

- A real time ocean simulation of Gerstner waves with the ability to independently customise and create waves of a desired type animating in a realistic plausible way.
- A real time ocean simulation using the statistical wave model optimised using the FFTW library with the ability to run a realistic ocean simulation with up to 256x256 resolution at real time and 1024x1024 with slightly delayed frame rates.
- Integrated a UI that makes full use of the functionality available to both simulations allowing the user to test each wave model acting under varying conditions.

9.2 Known bugs and issues

Although the project has yielded largely positive results, certain details have remained unsolved so far. The creation of choppy waves in the fft model would be a primary bug to fix. This is an essential part of creating a visually appealing sea that has a greater realism. The limitations of the program have also been discussed. Currently, a 1024x1024 resolution grid is the maximum that the application can output before slowing down greatly. A GPU implementation could greatly speed this processing time up by parallel programming paradigms, thus allowing the system resources to be used in better managing

9.3 Further Work

The project has provided a basis for a large potential of further work.

The code design is an aspect of the project which has been open to improvement. The fft implementation has been integrated in to the Ocean class causing the Ocean class to have too much functionality, with ideas of abstraction being neglected. The code design may not give any improvements on speed but will greatly improve the maintainability and extensibility of the ocean simulation should any other developments can be made.

A number of extensions could be made in terms of the simulation's functionality. A Buoy or Boat class could be made in order to better simulate their motion in a body of water. From this many other possible ideas could be developed such as splashes where objects interact with the water, ripples and ocean wakes. A more challenging problem to tackle would be to simulate non-linear breaking (or crashing) waves in the sea or near a shore line. Even the physical phenomena of ocean tsunamis could be an area of development to be integrated.

Another obvious area of improvement lies in the development of an ocean shader that could potentially bring a photorealistic edge to the application. Although some positives were formed from the results of the project, very little time was left to spend on the rendering aspect. Various academic papers such as *Simulating Ocean Waves* (Tessendorf 2001), and *Deep-Water Animation and Rendering* (Jensen and Goliáš 2008) have been dedicated to the aspect of photorealistic or realtime ocean rendering. The use of GLSL shaders for the overall visual could be the means used to develop such shaders.

The current ocean simulation has been developed to purely compute on the CPU. This leaves a foundation for various computationally expensive when performed sequentially to be processed in parallel on the GPU. The equivalent application written in OpenCL or CUDA could boost the simulation's efficiency to greater proportions, allowing for more detail or further functionality at faster run times, or even real time.

A final potential improvement lies in the portability of the code. The program as a standalone application, whilst good for running simulations, still lacks the ability to be of use to other commercially used software. For example, the application can be ported as a Maya, Houdini or SoftImage as a plug in. Such a feature could then be used in scenes for use in studios to facilitate creative story telling and visual effects in industry.

References

Tessendorf, J. 2001. Simulating ocean water. Simulating Nature: Realistic and Interactive Techniques. SIGGRAPH ‘01.

Tessendorf, J., 2001. *Simulating Ocean Surfaces*. Available from: http://ftp.cg.tuwien.ac.at/courses/Rendering/Tessendorf_SIGGRAPH_Slides2001.pdf [Accessed on 24 May 2013]

Young, I. R., 1999.*Wind Generated Ocean Waves*. Oxford: Elsevier.

Brigham, E. O., 1974. *The Fast Fourier Transform*. New Jersey: Prentice-Hall.

Boggessm A. & Narcowich, F. J., 2001. *A First Course in Wavelets with Fourier Analysis*. New Jersey: Prentice-Hall.

Brigham, E. O., 1988. *The Fast Fourier Transform and its Applications*. New Jersey: Prentice-Hall.

Elliot, D. F. & Rao, K. R., 1982. *Fast Transforms: Algorithm, Analyses, Applications*. London: Academic Press.

EDXGraphics, 2011. *Simulating Ocean Waves with FFT's on GPU*. Available from: http://www.edxgraphics.com/2/post/2011/10/simulating-ocean-waves-with-fft-on-gpu.html [Accessed on 5 June 2013]

Lantz, K., 2013. *Ocean Simulation Part One: Using The Discrete Fourier Transform*. Available from: http://www.keithlantz.net/2011/10/ocean-simulation-part-one-using-the-discrete-fourier-transform/ [Accessed on 03 June 2013]

Gamasutra, 2001. *Deep-Water Animation and Rendering*. Available from: http://web.archive.org/web/20081221123557/http://www.gamasutra.com/gdce/2001/jensen/jensen_01.htm [Accessed on 11 June 2013]

Nvidia, 2013. *GPU Technology Conference*. Available from: https://developer.nvidia.com/sites/default/files/akamai/gamedev/docs/GTC2013_GFX_libs.pdf [Accessed on 01 June 2013]

Nvidia, 2011. *Ocean Surface Simulation*. Available from: https://developer.nvidia.com/sites/default/files/akamai/gamedev/files/sdk/11/OceanCS_Slides.pdf [Accessed on 28 May 2013]

Munshi A., Gaster, B., Mattson, T. , Fung J., Ginsburg D., 2011. *OpenCL Programming Guide*. Boston: Addison-Wesley.

Jensen, L. and Goliáš, R. 2008. Deep-Water Animation and Rendering. Available from: http://whitenight-games.chez-alice.fr/Docs/Water.pdf

[Accessed on 05 June 2013].

Premože, S. and Ashikhmin, M. 2001. Rendering Natural Waters. *Computer Graphics Forum*, 20 (4), pp. 189-200.

Fréchot, J. 2006. Realistic simulation of ocean surface using wave spectra. *GRAPP 2006 - Proceedings of the 1st International Conference on Computer Graphics Theory and Applications* 2006, pp. 76-83.

Leblanc, G., Shouldice, A., Arnold, D. and Brooks, S. 2012. Multi-Band Fourier Synthesis of Ocean Waves. *Journal of Graphics Tools*, 16 (2), pp. 57-70.

//correct the following references...

Fournier, A. and Reeves ,W.T., 1986. *A Simple Model of Ocean Waves*. Computer Graphics, Vol. 20, No. 4, 1986, p 75-84.

Mastin,G.A., Watterger, P.A., Mareda, J.F, 1987. Fourier Synthesis of Ocean Scenes, IEEE CG&A, p 16-23.

Mitchell, J.E., 2005. Real Time Synthesis and Rendering Ocean Water. ATI Research Technical Report.

Press,W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 2002. *Numerical Recipes in C++: The Art of Scientific Computing*. 2nd Edition. Cambridge: Cambridge University Press

Ocean World, 2007. *Chapter 16: Ocean Waves*. Available from:
http://oceanworld.tamu.edu/resources/ocng_textbook/chapter16/chapter16_01.htm
[Accessed on 02 July 2013]

FFTW, 2013. Available from:
http://www.fftw.org
[Accessed on 10 July 2013]

Bale, K., 2010. *osgOcean.* Available from:
http://osgocean.googlecode.com/svn/trunk/src/osgOcean/FFTSimulation.cpp
[Accessed on 10 August 2013]