School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

Realistic Ocean Simulation using Fourier Transform

Saulius Vincevičius
sc21sv

Submitted in accordance with the requirements for the degree of
BSc Computer Science

2023/24

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (30/04/24)
Git repository	URL	Provided by university

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) Saulius Vincevičius

Summary

This research is centered on the development and implementation of a real-time ocean simulation model. The model addresses several significant challenges in the field, including the elimination of tiling artifacts, adaptability to a range of weather conditions from calm to stormy, and efficient rendering capabilities across both low and high-end devices.

The cornerstone of this research is the introduction of a Fourier Transform-based ocean model. This model is unique in its use of empirical data, which allows it to accurately simulate various weather conditions. Additionally, the model employs multiple cascades to effectively eliminate the common issue of tiling, enhancing the visual continuity of the simulated ocean surface.

To ensure broad device compatibility and efficient performance, the entire solution has been implemented on a GPU. This includes the integration of a Fast Fourier Transform algorithm, which enables the model to function efficiently across a wide range of devices.

Acknowledgements

I would like to acknowledge the guidance and support provided by Hamish Carr in his role as my supervisor.

Contents

Chapter 1

Introduction and Background Research

1.1 Introduction

Ocean surface simulation 1.1 is a field of computer graphics that aims to create realistic representations of the ocean surface. It is an important area of research as it has a wide range of applications, including video games, movies. In video games it used for interactivity and visual appeal, while in movies it is used for visual appeal.

There are many techniques that have been developed to simulate ocean surfaces, from the simple algorithms as sum of sines or Gerstner waves to more complex techniques such as particle simulations or Fast Fourier Transform (FFT) based Ocean 1.1.

An essential aspect of ocean surface simulation is shading, which adds depth and realism to the rendered image. Two commonly used models for this purpose are Phong Shading and Physically Based Rendering (PBR). Phong Shading provides a balance between simplicity and visual quality, making it a popular choice for various applications. On the other hand, PBR offers a more realistic rendering by accurately simulating the interaction of light with different materials.

The primary objective of this project is to simulate a realistic ocean surface, one that does not exhibit any tiling or repeating patterns, and that can accurately represent stormy weather conditions. An important aspect of achieving this realism is the careful shading of the ocean surface. Furthermore, we aim to achieve this simulation in real time, making it compatible with both low-end and high-end GPUs.

This is achieved by leveraging the power of the Inverse Fourier Transform (IFFT), a mathematical technique that transforms data from the frequency domain back to the time (or spatial) domain. In the context of this project, it allows us to transform the frequency data of the ocean waves into a spatial representation, i.e., the height map of the ocean surface. This is desirable as it allows us to simulate realistic ocean surfaces that are not only visually appealing, but also physically accurate as we are using real-world data to generate frequencies.

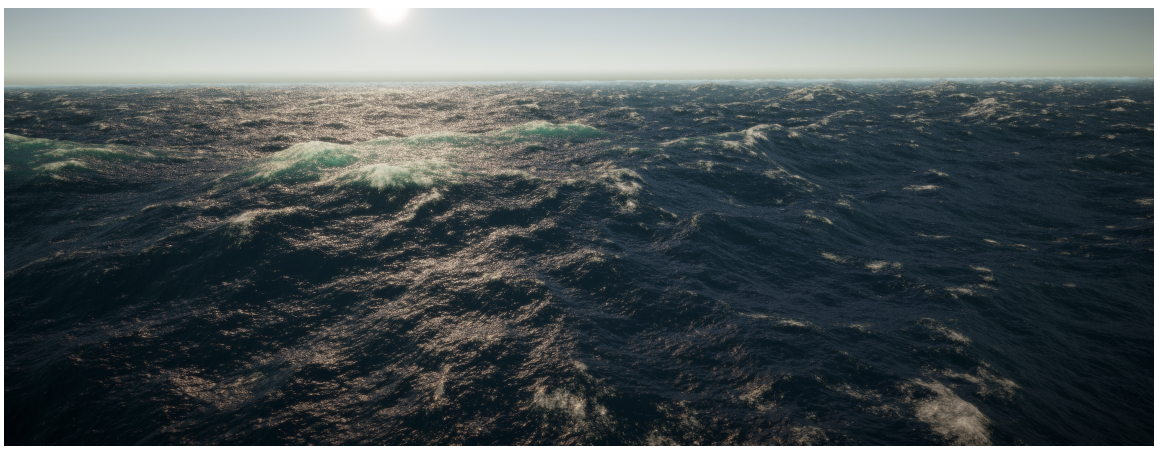

Figure 1.1: FFT Ocean Simulation

1.2 Related Work

1.2.1 Gerstner Waves

Gerstner waves, first introduced by F.J. Gerstner (1802) [1], provide a simple model for generating somewhat realistic ocean waves. This model is predominantly used in video games, including major titles like "Pokemon Legends: Arceus". The model approximates the motion of ocean waves by assuming that each point in the ocean undergoes a circular motion.

The Gerstner waves model is limited to a single wave. To simulate more complex ocean surfaces, multiple waves are summed together. However, this approach can be computationally expensive when simulating realistic ocean surfaces, as it requires summing a large number of waves together.

Furthermore, Gerstner waves offer limited artistic control, can result in visible tiling, and underperformed in simulating stormy weather conditions.

1.2.2 Fourier Transform

Understanding the Fourier Transform is crucial as it forms the backbone of our ocean simulation. This mathematical method, invented by Joseph Fourier (1822) [2], allows us to convert data from the time domain to the frequency domain and vice versa. To put it simply, imagine having a smoothie that's too sour. The Fourier Transform enables us to deconstruct our smoothie (time domain) into its ingredients (frequency domain), remove the sour component, and then reconstruct it back. It finds applications in numerous fields, including signal and image processing, and in our case, ocean simulation. To convert our data from the time domain to the frequency domain, we use the Fourier Transform:

$$\tilde{f}(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \tag{1.1}$$

, where $f(x)$ is the function in the time domain, $\tilde{f}(\omega)$ is the function in the frequency domain, and ω is a frequency. To convert our data from the frequency domain back to the time domain, we use the Inverse Fourier Transform:

$$f(x) = \int_{-\infty}^{\infty} \tilde{f}(\omega) e^{i\omega x} d\omega \tag{1.2}$$

As, we going to work with discrete data, we are going to use the Discrete Fourier Transform (DFT):

$$x_n = \sum_{k=0}^{N-1} \tilde{x}_k e^{-i2\pi kn/N} \tag{1.3}$$

and the Inverse Discrete Fourier Transform (IDFT):

$$\tilde{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{i2\pi kn/N} \tag{1.4}$$

1.2.3 Fourier Transform Ocean

Figure 1.2: Height Map using J. Tessendorf's Spectrum

A significant challenge associated with the utilization of Gerstner waves is the necessity to generate multiple waves for each vertex in order to simulate an ocean. This process can be computationally intensive, particularly considering that an oceanic simulation may be constructed of hundred of thousands vertices. In response to this computational demand, J. Tessendorf (2001) [3] proposed a method for generating ocean waves using the Fourier Transform.

The core concept involves generating a height map of the ocean surface in the frequency domain and converting it back to the time domain using the Inverse Fourier Transform (IFT). The IFT yields a periodic height map, which can be tiled to simulate an infinite ocean surface. This allows us to generate a single texture and apply it across the entire water body.

The benefits of this method are evident when considering a 512x512 texture, which combines 262,144 distinct waves. In contrast, Gerstner waves start to show performance degradation after combining 65 distinct waves for each vertex, which is insufficient for a realistic ocean representation.

To produce the height map, we first need a function that approximates the frequencies. We call this function a spectrum. J. Tessendorf uses a modified Phillips spectrum. This spectrum generates Fourier amplitudes in the frequency domain:

$$\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}}(\xi_r + \xi_i)\sqrt{P_h(\mathbf{k})} \tag{1.5}$$

where ξ_r and ξ_i is a complex number drawn from a Gaussian random number generator with mean 0 and standard deviation 1, P_h is modified Phillips spectrum, and $\mathbf{k}$ is a wave vector.

We then combine $\tilde{h}_0(\mathbf{k})$ with its conjugate $\tilde{h}_0^*(-\mathbf{k})$, to "produce waves towards and against the wave direction when propagating"[4]:

$$\tilde{h}(\mathbf{k}, t) = \tilde{h}_0(\mathbf{k})e^{i\omega(k)t} + \tilde{h}_0^*(-\mathbf{k})e^{-i\omega(k)t} \tag{1.6}$$

By performing the Inverse Discrete Fourier Transform (IDFT):

$$h(\mathbf{x}) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t)e^{i\mathbf{k}\cdot\mathbf{x}} \tag{1.7}$$

we obtain the height map 1.2, where $\mathbf{x} = (x, y)$ represents the position in the texture.

1.2.4 Dispersion Relationship

In the study of wave spectra, it is essential to establish a functional relationship "between the travel speed of waves and their wavelengths", as stated by Christopher J Horvath (2015) [4]. This relationship, known as the dispersion relation, is typically denoted by ω. In the context of deep water wave dynamics, the dispersion relation is expressed as follows:

$$\omega^2 = gk$$

where g is gravity and k denotes the magnitude of the wave vector.

1.2.5 Tessendorf's Spectrum

The appearance of the ocean is largely determined by the spectrum. In his paper, Jerry Tessendorf (2001) [3] proposed a modified version of the Phillips spectrum, which we will refer to as Tessendorf's spectrum. This spectrum takes into account various factors such as the amplitude, wind speed, gravity, and the direction of the waves. However, Tessendorf acknowledges that this spectrum has "poor convergence properties at high values of the wave vectors" [3].

1.2.6 JONSWAP Spectrum

In response to the challenges experienced with the "Tessendorf's Spectrum", Horvath (2015) [4] proposed an alternative: the JONSWAP (Joint North Sea Wave Project) spectrum. The spectrum was initially developed by Hasselmann et al. (1973) [5]. This spectrum represents an improvement over the Pierson-Moskowitz Spectrum (1964) [6]. Hasselmann's key observation was that the wave spectrum is never fully developed. This led him to introduce an extra peak enhancement factor, denoted as γ^r, into the JONSWAP spectrum. The JONSWAP spectrum is defined as follows:

$$\begin{aligned}
S_{\text{JONSWAP}}(\omega) &= \frac{\alpha g^2}{\omega^5} \exp\left(-\frac{5}{4}\left(\frac{\omega_p}{\omega}\right)^4\right)\gamma^r \\
r &= \exp\left(-\frac{(\omega-\omega_p)^2}{2\sigma^2\omega_p^2}\right) \\
\alpha &= 0.076\left(\frac{U_{10}^2}{Fg}\right)^{0.22} \\
\omega_p &= 22\left(\frac{g^2}{U_{10}F}\right)^{1/3} \\
\gamma &= 3.3 \\
\sigma &= \begin{cases} 0.07 & \text{if } \omega \leq \omega_p \\ 0.09 & \text{if } \omega > \omega_p \end{cases}
\end{aligned} \tag{1.8}$$

where F is the fetch (distance to lee shore), U_{10} is the wind speed at 10 meters above the sea level and ω is dispersion relationship.

1.2.7 The Texel MARSEN ARSLOE (TMA) Spectrum

One of the limitations of the JONSWAP spectrum is that it only applies to deep waters. To address this, Horvath Christopher [4] proposed the use of the TMA correction for the deep water spectrum. The TMA was developed by Steven A. Hughes (1984) [7], based on observations made by Kitaigordskii (1975) [8]. This is known as the Kitaigordskii depth Attenuation function, which is defined as follows:

$$\Phi(\omega_h) = \left[\frac{(k(\omega,k))^{-3} \frac{\partial k(\omega,h)}{\partial \omega}}{(k(\omega,\infty))^{-3} \frac{\partial k(\omega,\infty)}{\partial \omega}} \right] \tag{1.9}$$
$$\omega_h = \omega\sqrt{h/g}$$

where h is the depth of the water. At first glance, this function seems complex, but as shown in Figure 1.3, it can be easily approximated.

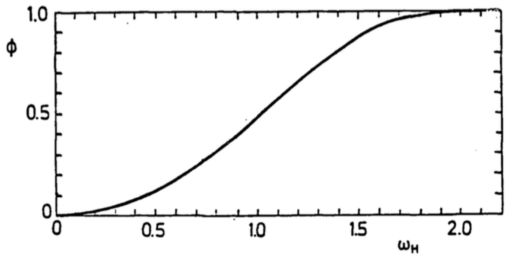

Figure 1.3: $\Phi(\omega, h)$ as a function of ω_h [7]

Approximation given by Thomson and Vincent (1983) [9] is:

$$\Phi(\omega,h) \approx \begin{cases} \frac{1}{2}\omega_h^2 & \text{if } \omega_h \leq 1 \\ 1 - \frac{1}{2}(2-\omega_h)^2 & \text{if } \omega_h > 1 \end{cases} \tag{1.10}$$

With $\Phi(\omega, h)$ we can now correct the JONSWAP spectrum for shallow waters:

$$S_{\text{TMA}}(\omega,h) = S_{\text{JONSWAP}}(\omega)\Phi(\omega,h) \tag{1.11}$$

Donelan-Banner Directional Spreading

Currently, the TMA spectrum cannot be used for our project due to a few issues. Firstly, this spectrum is non-directional, so we need to add directionality to it. Secondly, this spectrum accepts ω as input, but as we are following J. Tessendorf's [3] paper, we need to use $\mathbf{k}$ as input. To solve the first problem Horvath Christopher [4] proposes to use Donelan-Banner Directional Spreading (1999) [10]:

$$D(\omega,\theta) = \frac{\beta_s}{2\tanh(\beta_s\pi)}\text{sech}(\beta_s\theta)^2 \tag{1.12}$$

where,

$$\beta_s = \begin{cases} 2.61(\omega/\omega_p)^{1.3} & \text{for } 0.56 < \omega/\omega_p < 0.95 \\ 2.28(\omega/\omega_p)^{-1.3} & \text{for } 0.95 \leq \omega/\omega_p < 1.6 \\ 10^{\epsilon} & \text{for } \omega/\omega_p \geq 1.6 \end{cases}$$

$$\epsilon = -0.4 + 0.8393 \cdot e^{-0.567\ln(\omega/\omega_p)^2}$$

$$\theta = \arctan(k.y/k.x) - \theta_{\text{wind}}$$

In our project we will use $\omega/\omega_p < 0.95$ for first case as it produces more smooth results. Now we can combine the TMA spectrum with the Donelan-Banner Directional Spreading:

$$D_{\text{TMA}}(\omega, \theta) = S_{\text{TMA}}(\omega) \cdot D(\omega, \theta) \tag{1.13}$$

TMA transformation

Lastly, to make this spectrum usable, we need to transform it from ω to $\mathbf{k}$ as we follow J. Tessendorf's paper [3]. According to Horvath Christopher [4], we can transform it as follows:

$$S_{\text{TMA}}(\mathbf{k}) = 2S_{\text{TMA}}(\omega, h) \cdot \frac{d\omega}{dk}/k \cdot \Delta k_x \cdot \Delta k_y \tag{1.14}$$

where in our case,

$$\frac{d\omega}{dk} = \frac{g}{2\sqrt{g * \mathbf{k}}}$$

1.2.8 Cooley-Tukey Fast Fourier Transform (FFT)

When simulating fourier transform ocean the majority of time is spent on converting from frequency domain to time domain, i.e., performing the IFT. The previously mentioned IDFT 1.4 has time complexity of $O(n^2)$, which is clearly insufficient for simulating higher resolution ocean surfaces in real time. To address this, the Fast Fourier Transform (FFT) algorithm was invented by Gauss Carl Friedrich (1805) [11] and later reinvented by Cooley James W. and Tukey John W. (1965) [12]. The FFT algorithm, with a time complexity of $O(n \log n)$, exploits the redundancy in the computation of the DFT to reduce the time complexity. It's important to note that the FFT algorithm only works when the number of samples is a power of 2.

The basic idea of the FFT algorithm is as follows:

1. Split the input data into two subsets: one containing the even sample indices and the other containing the odd sample indices. Repeat this process until each subset contains only 2 samples (this is known as bit-reverse sort order).
2. Generate twiddle factor $W_N = e^{-i2\pi k/N}$, raised by k k:

   $$k = i * N/2^{\text{stage}+1} \bmod N \tag{1.15}$$

   where i is the index of the sample, stage is the current stage and N is the number of samples.
3. Perform butterfly operations 1.4, where x and y are complex numbers and W is the twiddle factor

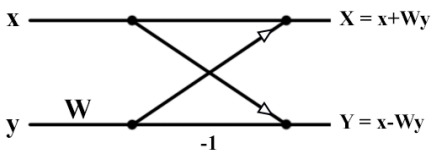

Figure 1.4: Butterfly Diagram

4. The butterfly operations are performed in stages 1.5. For example, if we have 8 samples, we will have $\log(8) = 3$ stages. The first stage is for pairs of points (2-point DFTs), the second stage is for groups of four points (4-point DFTs), and so on.

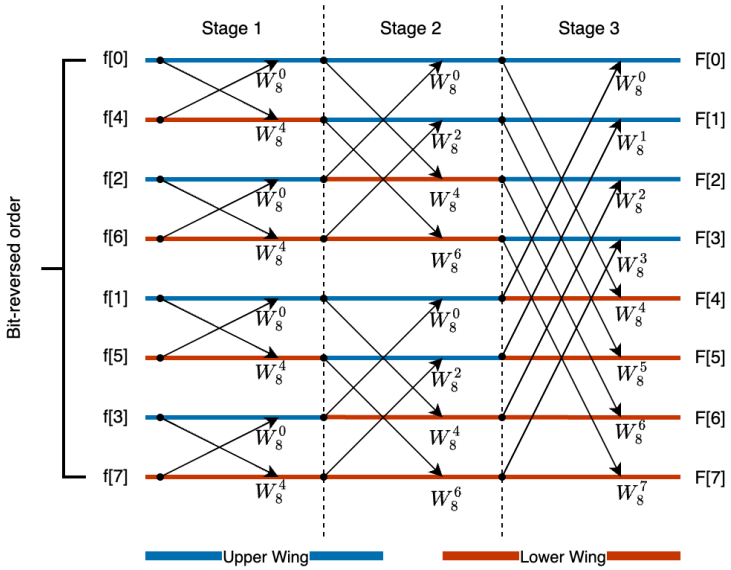

Figure 1.5: 8-point FFT Butterfly Diagram

1.2.9 Phong Shading

The Phong shading model, proposed by Phong Bui Tuong (1975) [13], is a simple yet effective method for approximating realistic shading. This model consists of three components: ambient shading, diffuse shading, and specular shading, as shown in Figure 1.6.

Figure 1.6: Phong Shading
Credits: learnopengl.com

The ambient component is a constant value that represents the light scattered throughout the entire scene. The diffuse component can be calculated using the following equation:

$$\text{Diffuse} = \max(\text{Normal} \cdot \text{LightDirection}, 0.0); \tag{1.16}$$

In this equation, the Normal is a normalized vector that is perpendicular to the surface and points away from it. The specular component can be calculated as follows:

$$\text{Specular} = max(\text{ViewDir} \cdot \text{ReflectionDir}, 0.0)^{\text{Shininess}} \tag{1.17}$$

Here, the ViewDir is a normalized vector pointing towards the camera, and the ReflectionDir is a normalized reflection vector with respect to the Normal. These vectors are illustrated in Figure 1.7.

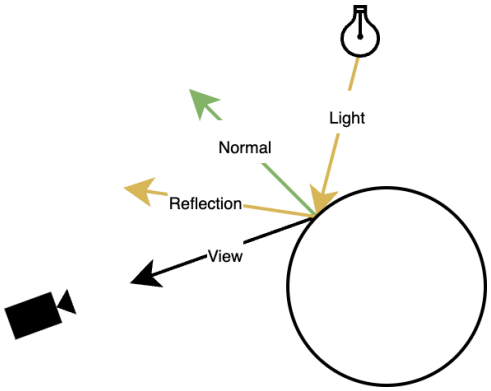

Figure 1.7: Phong Vectors

1.2.10 Physically Based Rendering (PBR)

Earlier mentioned Phong Shading model, does not provide a highly realistic approximation of lighting. To achieve a realistic representation of ocean shading, it is necessary to incorporate a multitude of custom parameters.

Physically Based Rendering (PBR) attempts to address this issue. However, it is important to note that "PBR is more of a conceptual framework than a set of rigid rules" [14]. The PBR rendering model adheres to three primary principles: energy conservation, microfacet support, and the Fresnel effect.

For the specific application of PBR in ocean shading, one can refer to “Wakes Explosions and Lighting: Interactive Water Simulation in Atlas” by Mark Mihelich and Tim Tcheblokov (2021) [15], which provides an in-depth discussion on the subject. For a more general understanding of PBR, “Physically Based Shading in Theory and Practice” by Stephen Hill and Stephen McAuley [16] offers comprehensive coverage of the topic from 2012 to 2020.

1.2.11 Particle Simulation and Machine Learning

Particle-based methodologies, known for their ability to simulate water in a realistic and visually compelling manner, are frequently employed in the field of computer graphics. However, these methods come with a high computational cost, which can be a limiting factor.

Recent advancements in Graphics Processing Unit (GPU) technology have started to alleviate this issue. Research, such as that conducted by Libo Huang et al. (2021) [17], is making particle simulations increasingly viable for ocean simulation. However, these techniques still demand a high-performance GPU and are not yet suitable for widespread real-time applications.

In addition to these developments, there have been significant strides in the application of machine learning to accelerate fluid simulations. This is evident in the work of Dmitrii Kochkov et al. (2021) [18]. Despite these advancements, real-time performance on lower-end GPUs remains a challenges which is one of the primary objectives.

Chapter 2

Methods

2.1 Algorithm Overview

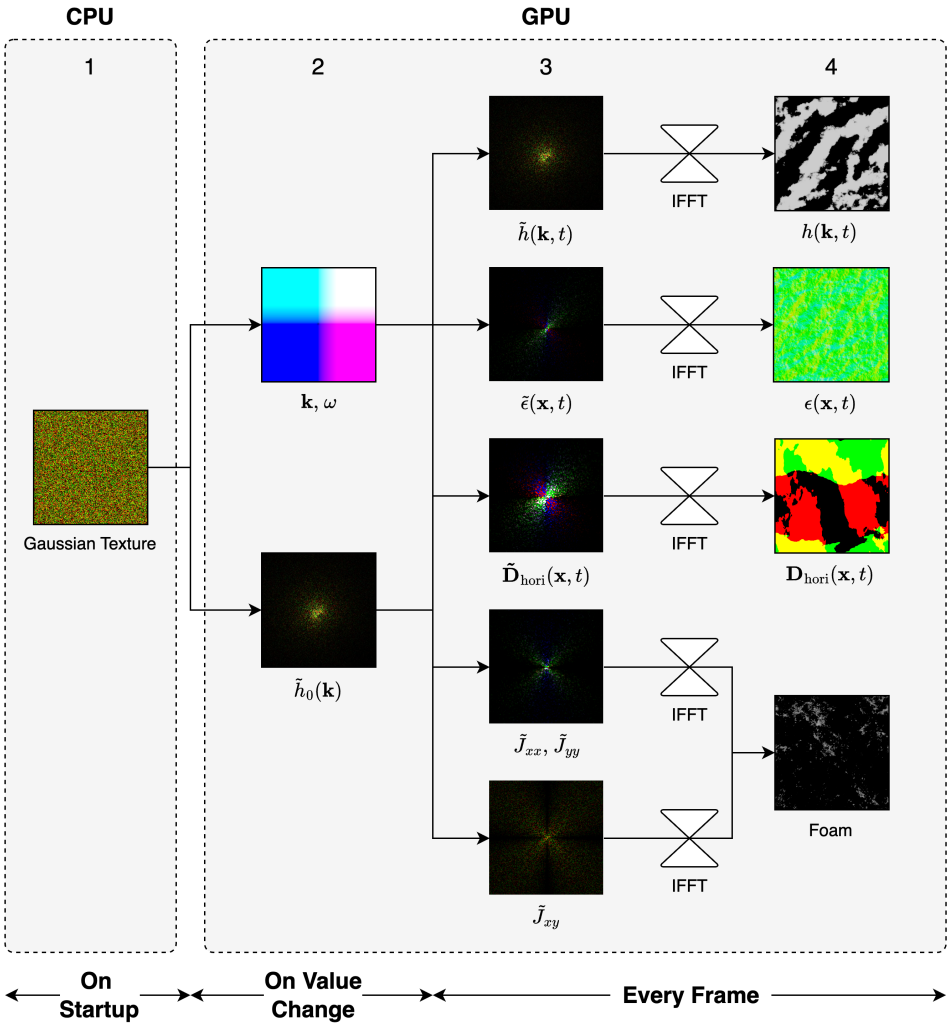

Figure 2.1: Ocean Generation Algorithm Stages: (1) Gaussian Noise Generation, (2) Initial Spectrum Creation, (3) Frequency Map Production, and (4) Conversion to Time Domain via IFFT.

The ocean generation algorithm utilizes the Fast Fourier Transform (FFT) and is designed for the Unity engine. It uses the High-Level Shader Language (HLSL) for GPU computations. The algorithm processes textures of size NxN, with N being a power of 2, to satisfy the Inverse Fast Fourier Transform (IFFT) data sample requirement. For this project, we use a texture size of 512x512, balancing computational performance and visual realism. The algorithm comprises four main stages, as shown in Figure 2.1.

1. **Gaussian Noise Generation:** The first stage involves the generation of Gaussian noise with a mean of 0 and a standard deviation of 1. Two values are generated, one for the real

part and the other for the complex part, as shown in Equation 1.5.

2. **Spectrum Generation:** The second stage involves the generation of the initial spectrum, $\tilde{h}_0(\mathbf{k})$, which is used to generate all subsequent results. The real and imaginary parts of the spectrum are stored as the red and green channels of a texture, respectively. Additionally, a separate texture is used to store the wave vector, $\mathbf{k}$, and the dispersion relation, ω.

3. **Frequency Generation:** In the third stage, the initial spectrum is used to generate a frequency height map for vertical vertex displacement, a frequency normal map for shading, and a frequency horizontal displacement map for horizontal vertex displacement. Furthermore, six values are calculated for the Jacobian, which is used to generate foam. Due to the limitation of a maximum of four values per pixel, two textures are used to store the Jacobian values.

4. **Inverse FFT (IFFT) Conversion:** The final stage involves the use of the Inverse Fast Fourier Transform (IFFT) algorithm to convert the maps from the frequency domain to the time domain.

2.2 IFFT

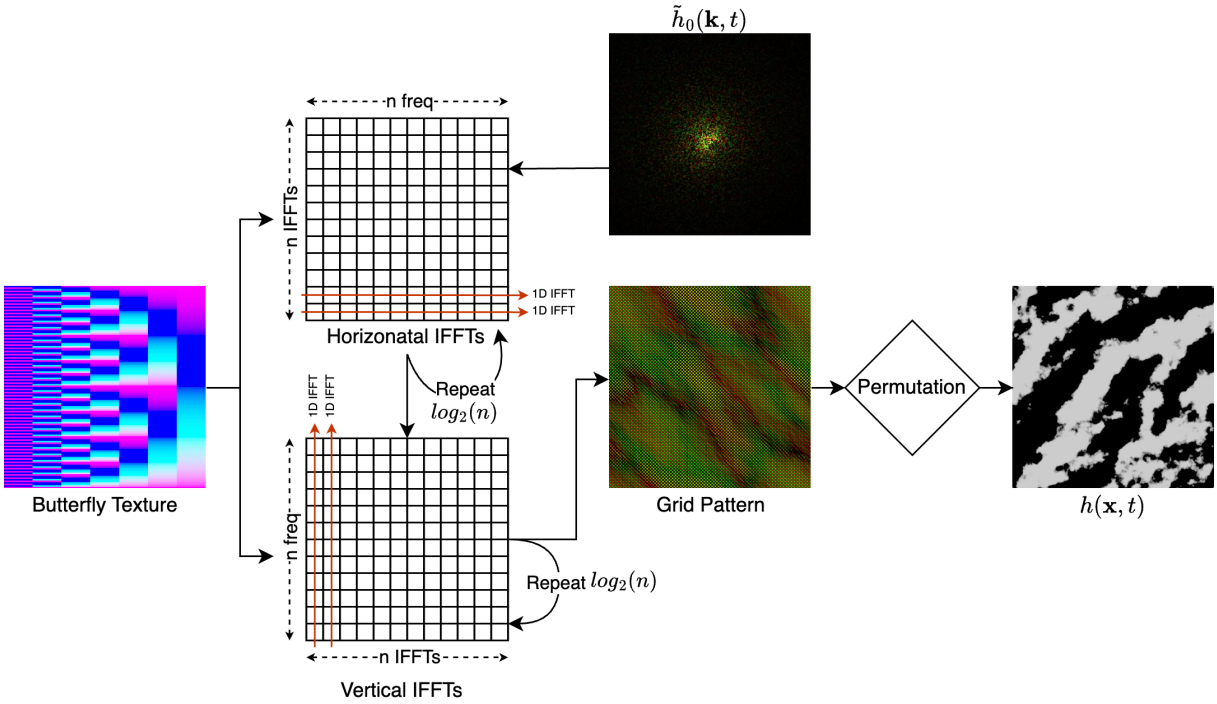

Figure 2.2: During the initialization phase, we generate a butterfly texture that stores the twiddle factors and indices. Subsequently, we apply a horizontal IFFT to the signal of size n_2, repeating this process $\log_2(n)$ times. The same procedure is then applied to vertical IFFT. Finally, we permute the data to transition from the frequency domain to the time domain.

2.2.1 Butterfly Texture

Our focus is primarily on the Inverse Fast Fourier Transform (IFFT) algorithm, and we will not be utilizing the FFT. To execute the IFFT, we generate a butterfly texture, as proposed by Flügge Fynn-Jorin (2017) [19]. This texture, which is precomputed and stored in the GPU

memory, has a width of $\log_2(n)$ and a height of n. It is a four-channel texture (W_r, W_i, y_t, y_b), where W_r and W_i are the real and imaginary parts of the twiddle factor, and y_t and y_b are the top and bottom butterfly indices, respectively (as shown in Figure 1.4).

In butterfly texture coordinate x represents a stage 1.5, while each y represents a butterfly operation between two data points y_t and y_b.

In the first stage we sort our data in bit reversed order, In the subsequent stages, we perform the following operations:

- If the butterfly wing is top half as shown in 1.5

$$\begin{aligned} y_t &= y_{\text{current}} \\ y_b &= y_{\text{current}} + 2^{\text{stage}} \end{aligned} \tag{2.1}$$

- If the butterfly wing is bottom half

$$\begin{aligned} y_t &= y_{\text{current}} - 2^{\text{stage}} \\ y_b &= y_{\text{current}} \end{aligned} \tag{2.2}$$

We can determine whether the wing is in the upper or lower half using the following equation:

$$\text{wing} = y_{\text{current}} \bmod 2^{(\text{stage}+1)} \tag{2.3}$$

Finally, we calculate the twiddle factors as follows:

$$\begin{aligned} k &= (y_{\text{current}} \cdot n/2^{\text{stage}+1}) \bmod n \\ W &= \exp(-2\pi i k/n) \end{aligned} \tag{2.4}$$

2.2.2 Performing IFFT

Our data is currently stored in a 2D texture. However, the IFFT algorithm is designed for 1D data. As a result, we need to perform a 1D IFFT on each row (horizontally **??**) and then on each column (vertically), as shown in Figure 2.2. By following pseudo code proposed by Flügge Fynn-Jorin (2017) [19] we can perform IFFT in 2D:

```
butterflyData = ButterflyTexture[Stage, id.x];
twiddle = butterflyData.xy;
// fetch top butterfly input sample
topSignal = PingPong0[butterflyData.z, id.y].xy;
// fetch bottom butterfly input sample
bottomSignal = PingPong0[butterflyData.w, id.y].xy;
// perform butterfly operation
h = topSignal + ComplexMult(twiddle, bottomSignal);
```

Listing 2.1: Horizontal Butterfly Operation

Notice that we are using ping-pong buffers to store intermediate results, as we need to perform multiple IFFT passes, in total $log_2(n)$ passes. After we perform IFFT on each row, we need to

perform for each column:

```
butterflyData = ButterflyTexture[Stage, id.y];
twiddle = butterflyData.xy;
// fetch top butterfly input sample
topSignal = PingPong0[id.x, butterflyData.z].xy;
// fetch bottom butterfly input sample
bottomSignal = PingPong0[id.x, butterflyData.w].xy;
// perform butterfly operation
h = topSignal + ComplexMult(twiddle, bottomSignal);
```

Listing 2.2: Vertical Butterfly Operation

Permutation

Lastly, our data needs to be permuted as you will see later our data is offset:

$$[\text{freq}(-N/2),\ ...,\ \text{freq}(-1),\ \text{freq}(0),\ \text{freq}(1),\ ...,\ \text{freq}(N/2-1)] \tag{2.5}$$

While, our IFFT algorithm expected data to be in the following order:

$$[\text{freq}(0),\ \text{freq}(1),\ ...,\ \text{freq}(N-1)] \tag{2.6}$$

This causes our data to flip sign in grid like pattern therefore we need to permute our data:

```
signs = {-1, 1};
index = (id.x + id.y) % 2;
sign = signs[index];
h = sign * PingPong1[id.xy].x;

PingPong0[id.xy] = h
```

Listing 2.3: Data Permutation [19]

2.3 Ocean Geometry

Symbol	Meaning
$S(\omega)$	Non directional wave spectrum
$S(\omega, \theta)$	Directional wave spectrum
$S(\mathbf{k})$	Directional wave spectrum
$\mathbf{k}$	wave vector
k	Magnitude of wave vector
l	Length scale of the ocean
ω	dispersion relationship (angular frequency)
U_{10}	Wind speed at 10m above the sea level
F	Fetch (Distance from lee shore)
θ_{wind}	Wind angle
λ	Wave Choppy factor

Table 2.1: Symbols for ocean geometry

We will utilize symbols inside table 2.1 to describe formulas needed for ocean geometry.

2.3.1 Spectrum Generation

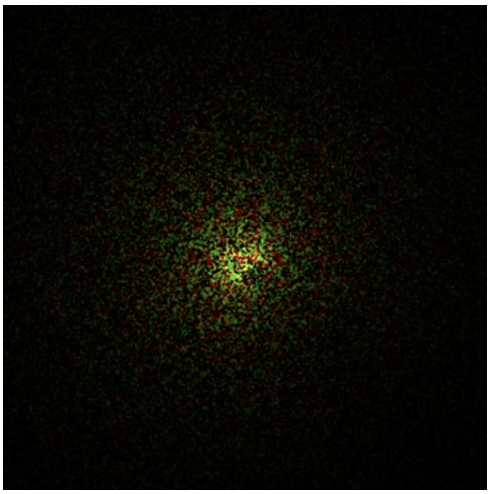

Figure 2.3: TMA spectrum, Frequency Domain

The choice of spectrum is pivotal to the visual fidelity of an ocean simulation. The Tessendorf's spectrum, initially implemented due to its straightforward application, did not yield satisfactory results. The wave energy transfer was unconvincing, the waves did not appear to adhere to the wave direction, and artistic control over the ocean's appearance was challenging. To ensure the realism of the simulated ocean, the TMA spectrum 1.14, grounded in empirical data, was subsequently implemented:

$$S_{\text{TMA}}(\mathbf{k}) = 2S_{\text{TMA}}(\omega, h) \cdot \frac{d\omega}{dk}/k \cdot \Delta k_x \cdot \Delta k_y$$

where $\mathbf{k} = (k_x, k_y)$, $k_x = 2 * \pi(x_x - n/2)/l$, $k_y = 2 * \pi(x_y - n/2)/l$, l is the the length scale of the ocean, and $\mathbf{x} = (x_x, x_y)$ is current position in the texture.

The TMA spectrum 2.3 resulted in a more realistic ocean simulation, offering intuitive controls over fetch, wind speed, wind angle, and depth. It required minimal effort to achieve a realistic appearance and provided satisfactory results out of the box.

2.3.2 Height Map Generation

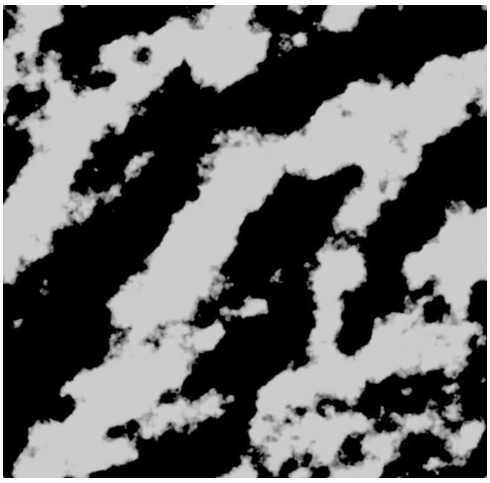

Figure 2.4: Height Map using S_{TMA}

The generation of a height map begins with the computation of Fourier amplitudes, as shown in Equation 1.5. Here, P_h represents the TMA Spectrum $S_{TMA}(\mathbf{k})$. The symmetry of the Fourier series allows us to mirror the amplitudes, eliminating the need for complex conjugate recalculation. This is represented as:

$$\tilde{h}_0^* = T_{h_0}(x^*, y^*) \tag{2.7}$$

where $T_{h_0}(x, y)$ is fourier amplitude in precomputed texture at $x^* = (n - x) \bmod n$, $y^* = (n - y) \bmod n$, (x, y) is current position in the texture.

By having combined amplitudes 1.6 we can perform IFFT as shown in 2.2 to produce height map as shown in figure 2.4.

2.3.3 Normal Map Generation

The generation of a normal map is an essential step in ocean simulation as it provides critical information for calculating ocean shading. A normal map represents the perpendicular direction to the ocean surface at each point. The computation of a normal map necessitates the calculation of the gradient, which is the derivative of the height map. As per Tessendorf (2001) [3], the derivative is given by:

$$\epsilon(\mathbf{x}, t) = i\mathbf{k}\tilde{h}(\mathbf{k}, t) \tag{2.8}$$

Then we need to perform IFFT 2.2 to get normal map 2.5 in the time domain.

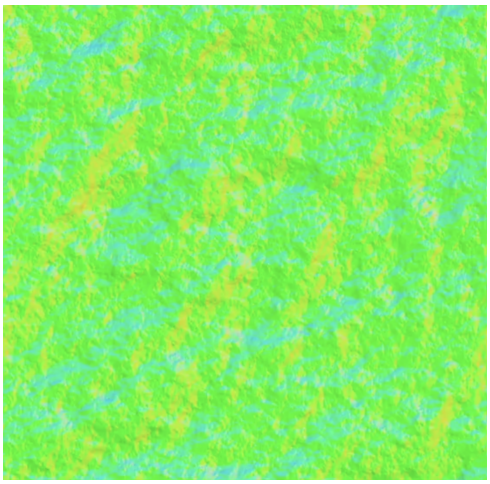

Figure 2.5: Normal Map using S_{TMA}

2.3.4 Choppy Waves

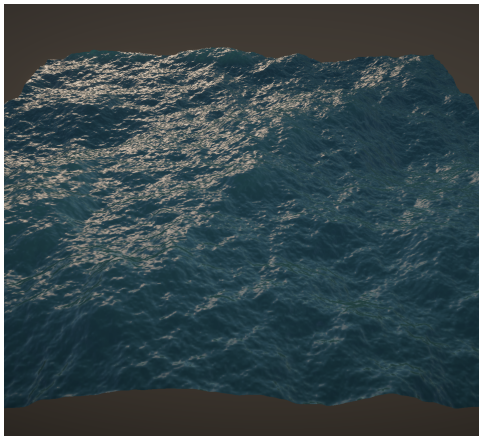

Figure 2.6: Ocean without Choppy Waves

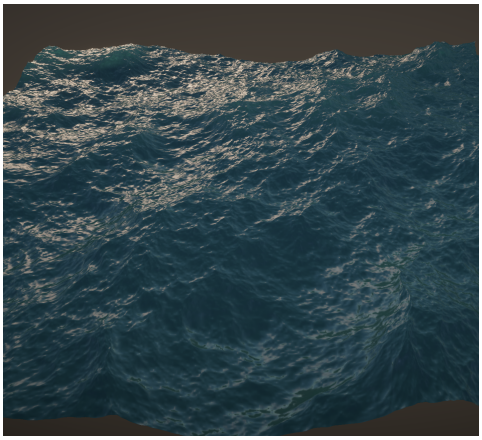

Figure 2.7: Ocean with Choppy Waves

The current rendering of the ocean (Figure 2.6) appears excessively smooth and lacks the characteristic choppiness of real-world ocean waves. To address this, we introduce horizontal displacement into our model. This not only enhances the choppiness of the waves but also improves the realism of energy transfer between waves (Figure 2.7). The calculation of horizontal displacement follows the formula proposed by Tessendorf (2001) [3]:

$$\mathbf{D}_{\text{hori}}(\mathbf{x}, t) = -i\frac{\mathbf{k}}{k}h(\tilde{\mathbf{k}}, t) \tag{2.9}$$

Utilizing this horizontal displacement, we can adjust the position of our vertices as follows:

$$\mathbf{x} + \lambda\mathbf{D}_{\text{hori}}(\mathbf{x}, t) \tag{2.10}$$

, where λ is "choppy factor". Once again we need to perform IFFT 2.2 to get horizontal displacement in the time domain.

2.4 Ocean Shading

Symbol	Parameter	Symbol	Parameter
L_a	Ambient Light	C_s	Specular Color
L_{ss}	Subsurface Scatter Light	C_{ws}	Water Scattering Color
L_s	Specular Light	C_{sky}	Sky Color
L_r	Environment Reflection	H	Ocean Height
N	Normal	F	Fresnel Effect
D_s	Sun Direction	ρ_a	Air Bubble Density
D_v	View Direction	k_a	Ambient Light Intensity
D_i	Camera To Fragment Direction	k_{ss_1}	Subsurface Scattering Intensity
C_a	Ambient Light Color	k_{ss_2}	Subsurface Scattering Intensity
C_l	Light Color	k_r	Reflection Intensity
C_b	Air Bubble Color		

Table 2.2: Symbols for shading

We will utilize symbols inside table 2.2 to describe formulas needed for ocean shading.

2.4.1 Lighting Model

Output

The lighting model employed in our simulation, as shown in Equation 2.11, is a combination of four fundamental components: ambient light, subsurface scattering, specular light, and environmental reflection. This model is the outcome of integrating the Phong model with concepts derived from the Game Developers Conference "Wakes, Explosions and Lighting: Interactive Water Simulation in Atlas" by Mark Mihelich and Tim Tcheblokov (2021) [15]. The resultant visual output of this composite lighting model can be observed in Figure 2.8.

$$L = L_a + L_{ss} + L_s + L_r \tag{2.11}$$

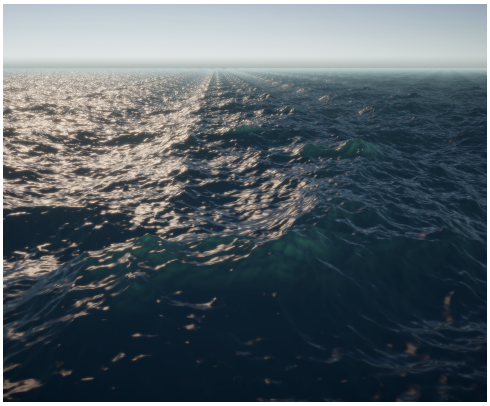

Figure 2.8: Final Shader Output

Ambient Light

Ambient light 2.9, a constant illumination that does not depend on the direction of the light source, is the result of light scattering in the environment. For oceanic simulations, we utilize an approximation formula for ambient light derived from the GDC conference by Mark Mihelich and Tim Tcheblokov (2021) [15]:

$$L_a = k_a N C_a C_l + \rho_a C_b C_l \tag{2.12}$$

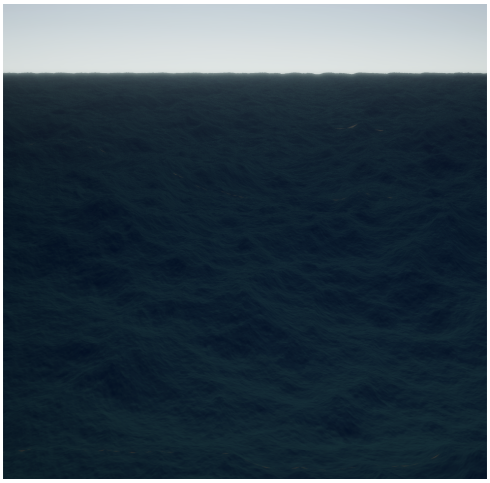

Figure 2.9: Ambient Light

Fresnel Effect

The Fresnel effect, a phenomenon where light is more reflective at grazing angles as shown in figure 2.10, is crucial for calculations involving specular and reflection:

$$F = (1 - \max(D_v \cdot N, 0.15))^5 \tag{2.13}$$

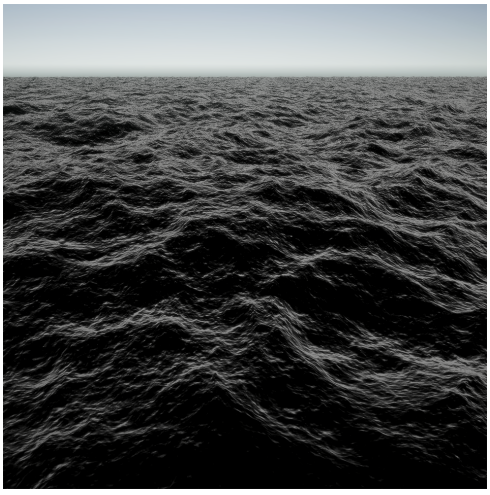

Figure 2.10: Fresnel Effect

Subsurface Scattering

Subsurface scattering describes the interaction of light when it penetrates an object, in this case, water. While ray tracing would typically be required for realistic results, we can use an approximation from the GDC conference [15] due to the majority of our light being trapped in the ocean, with only light at wave peaks able to escape. This can be observed in the following figures 2.11, 2.12 and 2.13

$$\begin{aligned} L_{ss} &= (1-F)(L_{ss_1} + L_{ss_2})C_{ws}C_l \\ L_{ss_1} &= k_{ss_1} \max(0, H)([D_s, -D_v])^4(0.5 - 0.5(D_s \cdot N))^3 \\ L_{ss_2} &= k_{ss_2}([D_v, N])^2 \end{aligned} \tag{2.14}$$

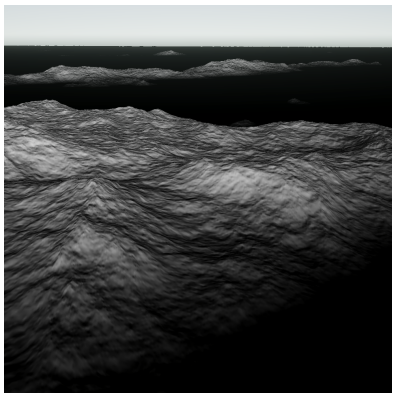

Figure 2.11: L_{ss_1}

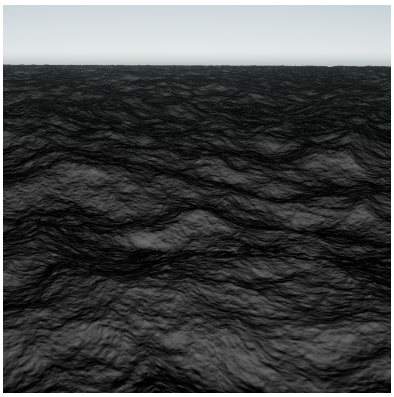

Figure 2.12: L_{ss_2}

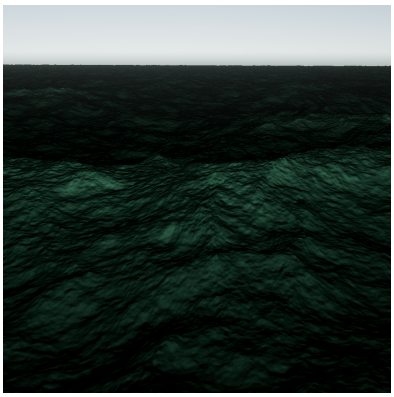

Figure 2.13: Subsurface Scattering

Specular Reflection

Specular reflection is like the shiny reflection we see on smooth surfaces. It's what makes things like metals, plastics, and in our case water look glossy. In our simulation, we utilize the Phong specular reflection model (Equation 1.17) and incorporate the Fresnel effect, denoted by F. The specular reflection can be observed in Figure 2.14.

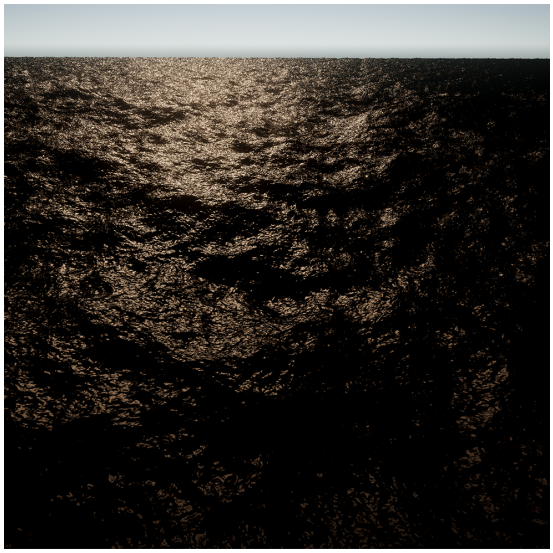

Figure 2.14: Specular Reflection

Enviroment Reflection

In our project, we only consider the reflection of the skybox on the water surface. As per Figure 2.15, we illustrate how skybox reflection operates on a calm ocean surface to better visualize environmental reflection.

$$\begin{aligned} L_r &= Fk_r C_{\text{sky}} \\ C_{\text{sky}} &= \text{Sky}[\text{reflect}(D_i, N)] \end{aligned} \tag{2.15}$$

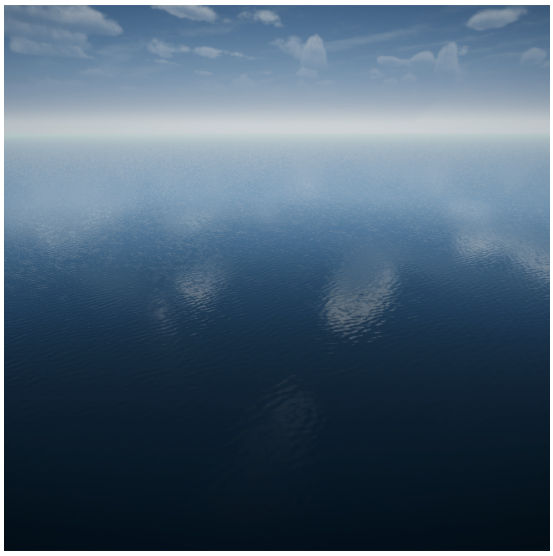

Figure 2.15: Skybox reflection on calm ocean

2.4.2 Foam

Foam Generation

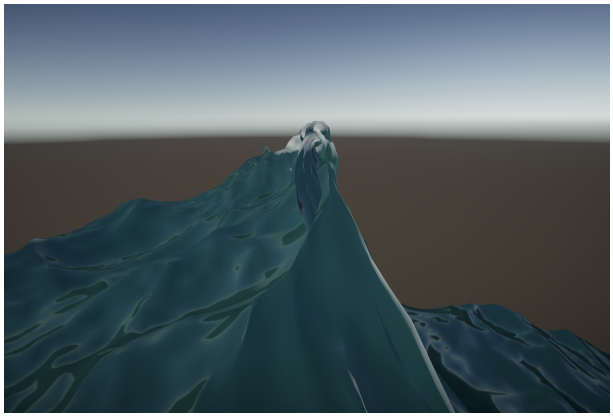

Figure 2.16: Wave Curl At Wave Peek

Ocean foam is a salient and realistic feature of the ocean, contributing to its distinct and authentic appearance. The primary occurrence of foam is observed during wave crashes, a phenomenon facilitated by horizontal displacement. This can be visually discerned at the peaks of waves where the water curls up, as illustrated in Figure 2.16. In essence, our horizontal transformation undergoes an inversion.

As proposed by Tessendorf (2001) [3], the rendering of foam can be achieved by calculating

the determinant of the Jacobian matrix for horizontal displacement, which helps identify these inversions. In our ocean simulation, the Jacobian matrix provides insights into the changes over the x and z axes. When determinant of the Jacobian matrix is bellow zero the "wave crash" happens:

$$\text{Det}(\mathbf{x}) = J_{xx} + J_{yy} - J_{xy}J_{yx} \tag{2.16}$$

where,

$$J(\mathbf{x}) = \begin{bmatrix} 1 + \lambda\frac{\partial D_x(\mathbf{x})}{\partial x} & 1 + \lambda\frac{\partial D_x(\mathbf{x})}{\partial y} \\ 1 + \lambda\frac{\partial D_y(\mathbf{x})}{\partial x} & 1 + \lambda\frac{\partial D_y(\mathbf{x})}{\partial y} \end{bmatrix} \tag{2.17}$$

in this case $J_{xy} = J_{yx}$. After applying IFFT, the results can be observed in figure 2.17.

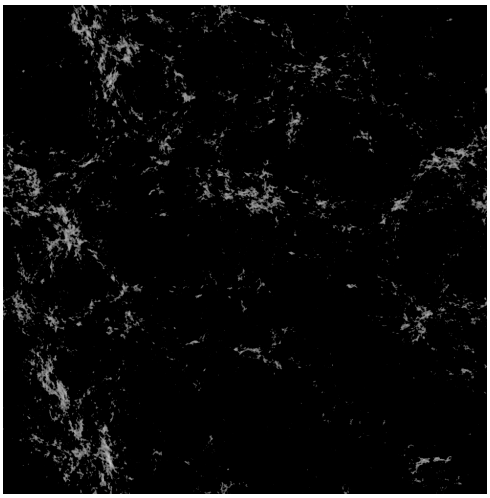

Figure 2.17: Foam Texture

Foam Accumilation

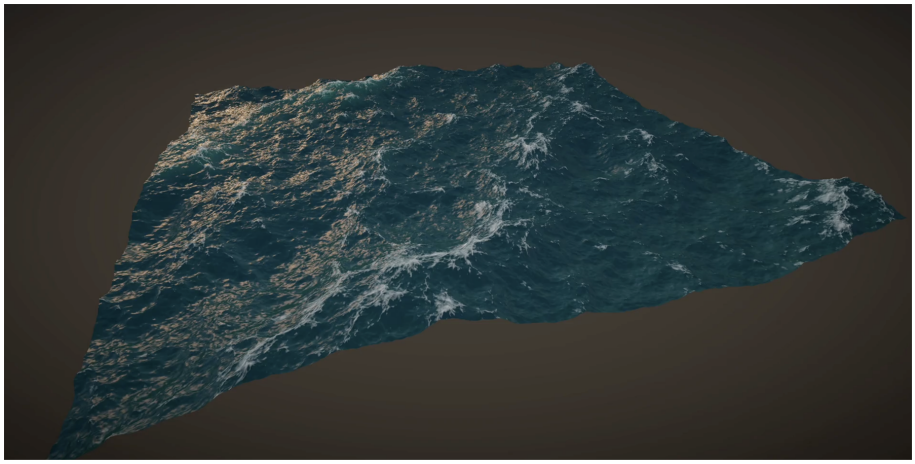

Figure 2.18: Ocean With Foam

Currently the foam appears and disappears quickly, however in real life the foam accumulates and disappears over time. We can introduce foam accumulation by compering previous and current foam values:

```
accumulation =
LastFoamValue - FoamDecay * DeltaTime / max(currentFoam, 0.5);
```

```
foam = max(accumulation, currentFoam);
```

Listing 2.4: Foam Accumilation

This results in pleasing foam accumulation (Figure 2.18):

2.5 Multiple Cascades

At this stage our ocean with texture 512x512 simulates 262,144 distinct waves however the tilling is still noticeable 2.19.

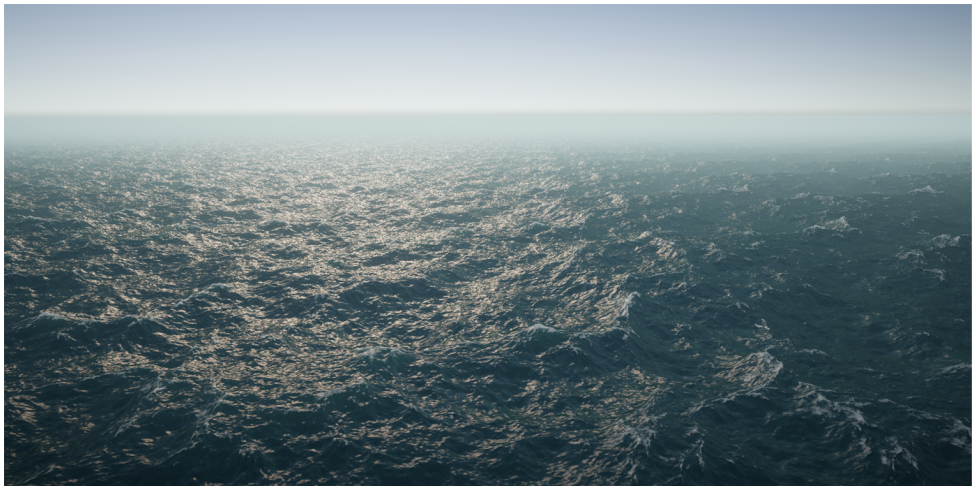

Figure 2.19: Ocean with Tilling, using 512x512 texture

To counter this we could increase our simulation texture however even FFT becomes expensive really fast. Another approach is to simulate multiple cascades for different wave lengths k 2.20 and different l length scales. We can split our simulation into 3 parts, big waves, medium waves and small waves.

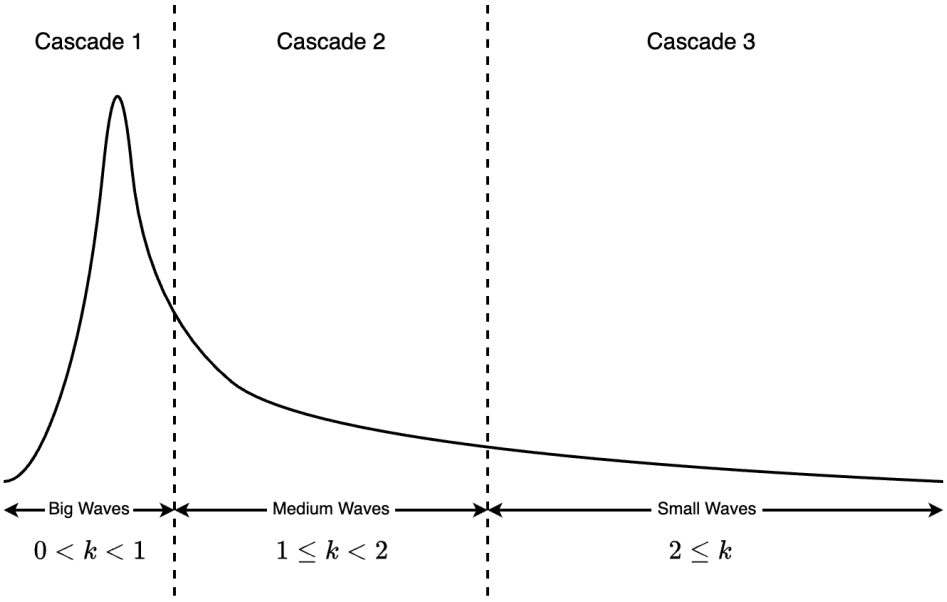

Figure 2.20: 3 cascades

When it comes to choosing l we need to follow these steps:

1. We chose bigger l for bigger waves as we are more likely to notice tilling with big waves

2. We chose smaller l for smaller waves as we are less likely to notice tilling with small waves

This results ocean that has less tilling and more detail 2.21.

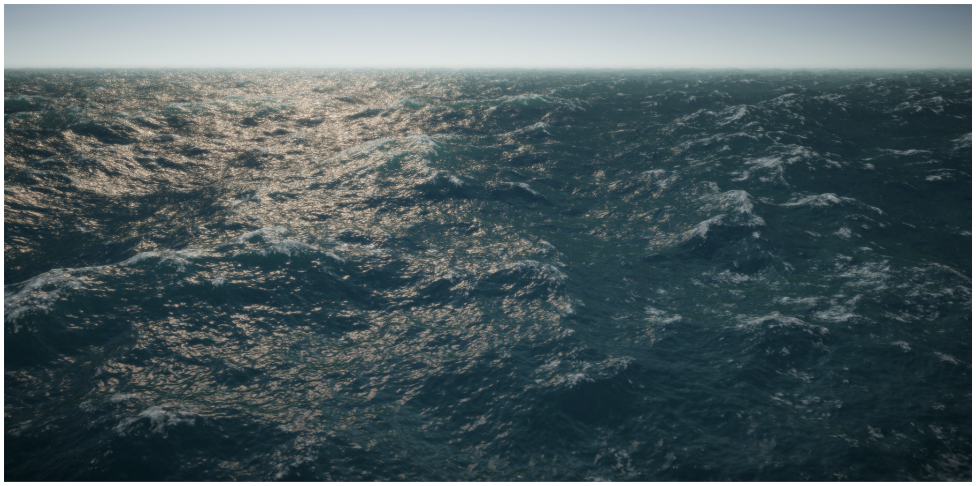

Figure 2.21: Ocean with Cascades, using 3x(512x512) textures

One of the main pros of having multiple cascades is reduction in performance cost, as we can have similar results of 512x512 with 3x(256x256) textures as shown in figure 2.22.

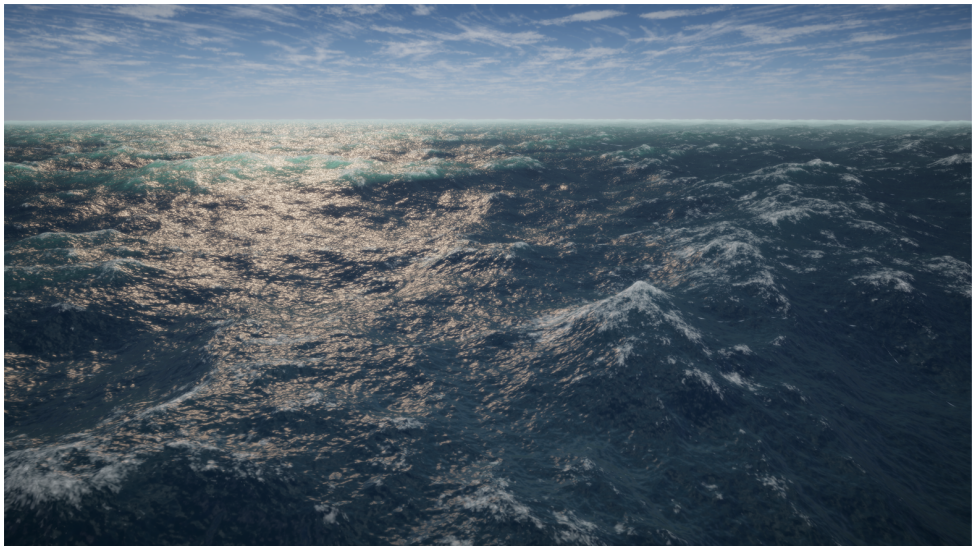

Figure 2.22: Ocean with Cascades, using 3x(256x256) textures

2.6 Library Choice

In the context of our ocean generation project, we evaluated six potential technologies, which can be categorized into three groups: older APIs, modern APIs, and game engines.

The older APIs category includes OpenGL, an ideal API for learning and cross-platform use. It's relatively easy to use, but it's becoming outdated.

The second category, modern APIs, includes Metal, Vulkan, and DirectX. These APIs are more complex to use, but they provide more control over the GPU and lower CPU usage. However, Metal and DirectX are limited to Apple and Microsoft platforms, respectively. In the case of Vulkan, it requires significantly more code to accomplish the same task.

The final category is game engines, specifically Unity and Unreal. These engines are excellent

for computer graphics as they offer a wealth of functionality, most importantly, useful UI tools that make development easier. However, they are massive tools with many custom needs, and you need experience to use them correctly.

All these options are viable as they all have the capability to fulfill our project requirements. However, after careful consideration, we chose Unity. This choice was primarily driven by our greater familiarity with Unity compared to other tools. Furthermore, Unity offers a comprehensive suite of tools, encompassing UI design, GPU programming, and an integrated build system. These features significantly streamline the development process, rendering Unity as the optimal choice for our project's needs.

2.7 Utilization of Unity Tools

In our project, we strategically employed Unity's tool set. We used C# for startup computations, setting the initial conditions and parameters for our project including communication between CPU and GPU.

For the intensive mathematical computations inside compute shaders and vertex and fragment shaders, we utilized the High-Level Shading Language (HLSL) for GPU programming. This approach allowed us to leverage the computational power of the GPU, enhancing the performance and visual fidelity of our project.

Additionally, Unity's built-in User Interface (UI) system and build system were used for creating interactive UIs and streamlining the compilation process, respectively. These tools collectively contributed to the effective and efficient development of our project.

2.8 Version Control

In the development of our project, we adopted GitHub as our version control system. This platform facilitated the systematic management of different versions of our project files. By organizing our project into branches, we were able to work on individual features without affecting the main codebase. This approach not only enhanced the efficiency of our development process but also ensured the integrity and stability of our project.

2.9 Testing

The outputs from each GPU kernel were rendered as UI images. This facilitated a direct visual comparison of our results with the graphical data presented in established research papers such as Tessendorf (2001) [3] and Flügge Fynn-Jorin (2017) [19].

Chapter 3

Results

3.1 Performance

The table below 3.1 presents the performance metrics of our FFT-based ocean generation technique on two different GPUs: M1 and Nvidia GeForce RTX 4070. The metrics were recorded for three different texture sizes: 256x256, 512x512, and 1024x1024.

GPU	Texture Size	Cascade Count	Time
M1	256x256	3	8.56ms
	512x512	3	43.50ms
	1024x1024	3	128.25ms
Nvidia GeForce RTX 4070	256x256	3	1.65ms
	512x512	3	3.29ms
	1024x1024	3	10.70ms

Table 3.1: Performance on high and and low end GPU

For a texture size of 256x256, the M1 GPU took 8.56ms and the Nvidia GeForce RTX 4070 took 1.65ms. This demonstrates that both GPUs are capable of rendering the ocean in real-time, showcasing the broad applicability of our technique.

However, as the texture size increases, the performance on the M1 GPU deteriorates significantly. For instance, the M1 GPU took 43.50ms for a 512x512 texture and 128.25ms for a 1024x1024 texture. This indicates that our technique becomes computationally expensive for weaker GPUs at higher texture sizes.

On the other hand, the Nvidia GeForce RTX 4070 GPU maintained a relatively consistent performance even with increased texture sizes, taking 3.29ms for a 512x512 texture and 10.70ms for a 1024x1024 texture.

Interestingly, the use of cascades in our technique mitigates this issue to an extent. Cascades allow us to maintain a relatively consistent level of detail in the ocean's visual representation, even when the texture size is increased. This feature makes our technique a viable option for game development across a wide range of computer systems, despite the performance variations observed with different GPUs and texture sizes. Thus, our FFT-based ocean generation technique exhibits a balance between visual fidelity and computational efficiency, making it a promising approach for real-time ocean rendering on low and high end GPUs.

3.2 Comparison

3.2.1 Spectral Analysis

The primary distinction in ocean simulation arises from the type of spectrum utilized in Fast Fourier Transform (FFT) based oceans. The "Phillips" spectrum, proposed by J. Tessendorf [3], as shown in Figure 3.1, exhibits challenges in energy transformation and adherence to wind direction.

In contrast, the proposed TMA spectrum, depicted in Figure 3.2, handles energy transformation in a more realistic manner and does not encounter issues in following the wind direction. This superior performance is primarily attributed to the fact that the TMA spectrum is based on empirical data.

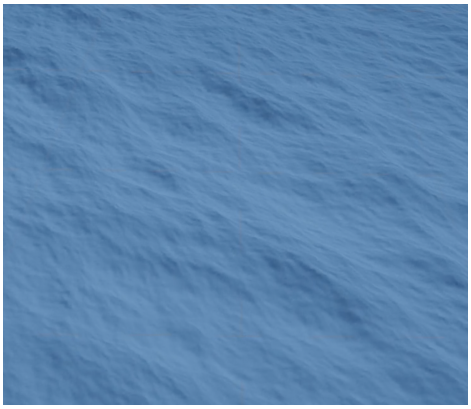

Figure 3.1: Height Map with P_h

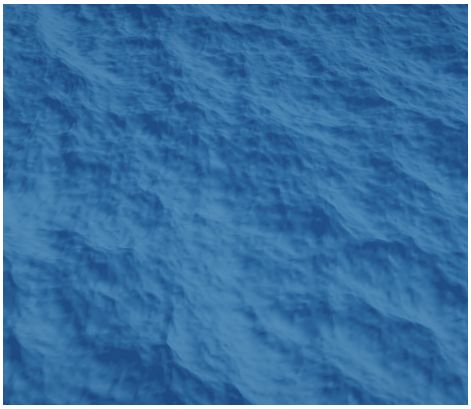

Figure 3.2: Height Map using TMA Spectrum

3.2.2 DFT and FFT

The computational complexity of the DFT and FFT presents a significant contrast. With DFT operating at $O(n^2)$ and FFT at $O(n \log(n))$, the difference becomes strikingly apparent when comparing the tables 3.2 and 3.1. It's clear that the FFT algorithm, even when run on a lower-end M1 GPU, outperforms the DFT algorithm on a high-end Nvidia GeForce RTX 4070 GPU. For instance, the FFT on the M1 GPU for a texture size of 256x256 and 3 cascades takes 8.56ms, while the DFT on the Nvidia GPU for the same texture size but only 1 cascade takes 18.65ms. This is over twice as long, despite the Nvidia GPU being a high-end model. The difference is even more pronounced for larger texture sizes. It should be noted that the DFT was not run on the M1 GPU as it didn't have enough power to execute the algorithm.

GPU	Texture Size	Cascade Count	Time
Nvidia GeForce RTX 4070	256x256	1	18.65ms
	512x512	1	81.26ms

Table 3.2: Performance of DFT

3.2.3 Sum of Sins and FFT Based Ocean

A comparative analysis of the visual aspects of the Sum of Sins and FFT-based approaches reveals distinct differences. The Sum of Sins approach, while capable of generating waveforms, exhibits a lack of detail that results in less realistic wave movements and energy transfers as shown in figure 3.3. This under performance becomes dramatically apparent in stormy conditions 3.4, where the complexity and intensity of wave interactions are not adequately captured. However, the visual fidelity can be enhanced by integrating a pre-generated detailed water normal map, which can add complexity and depth to the visual representation.

In contrast, the FFT-based approach inherently produces more realistic results. It automatically generates highly detailed normal maps, produces choppier waves, and accurately simulates energy transfers, contributing to a more authentic representation of oceanic conditions.

Figure 3.3: Sum Of Sines Calm Ocean

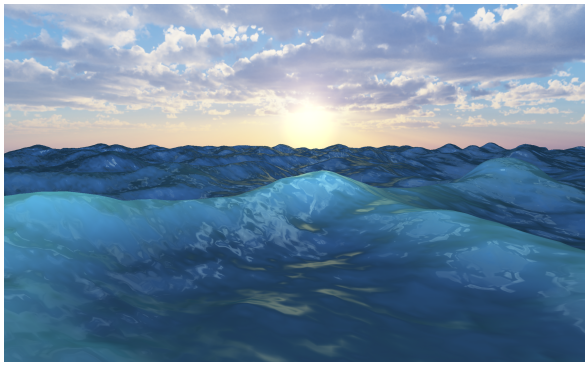

Figure 3.4: Sum Of Sines Stormy Ocean

3.2.4 Real world oceans

When it comes to calm ocean, the FFT based ocean with TMA spectrum performs extremely well and holds the desired shape as shown in figures 3.5 and 3.6.

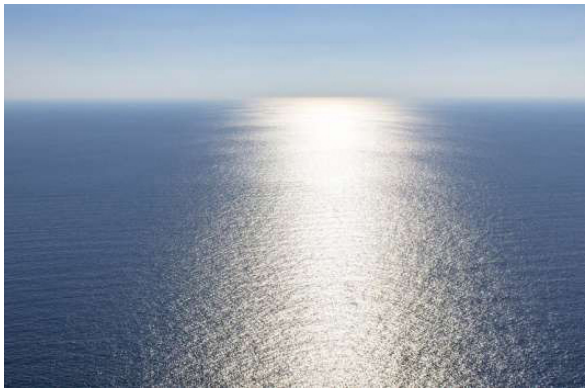

Figure 3.5: Calm Atlantic Ocean
Credits: CC0 Public Domain

Figure 3.6: Spectrum: TMA, Size: 256x256, l_1: 256, l_2: 100, l_3: 10, λ: O.9, U_{10}: 0.5 m/s, Fetch: 1000km, Depth: 500m

where each l is the wave length scale of each cascade.

When it comes to stormy ocean where huge waves is expected the general shape of an ocean is still realistic and can handle the big waves without any trouble, and because of different cascades the tilling is berley noticeable as shown in the following figures 3.7 and 3.8.

Figure 3.7: Calm Atlantic Ocean
Credits: CC0 Public Domain

Figure 3.8: Spectrum: TMA, Size: 256x256, l_1: 700, l_2: 256, l_3: 70, λ: O.9, U_{10}: 21 m/s, Fetch: 1000km, Depth: 500m

Diffrent Outputs

Following figures 3.9, 3.10 and 3.11 are oceans with different parameters.

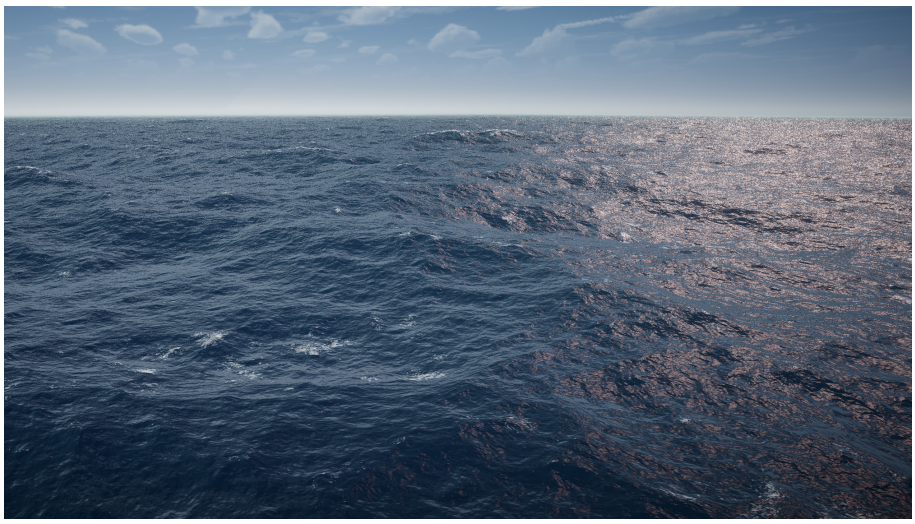

Figure 3.9: Spectrum: TMA, Size: 256x256, l_1: 400, l_2: 100, l_3: 10, λ: O.9, U_{10}: 0.5 m/s, Fetch: 1000km, Depth: 500m

Figure 3.10: Spectrum: TMA, Size: 256x256, l_1: 800, l_2: 256, l_3: 10, λ: O.95, U_{10}: 31 m/s, Fetch: 1000km, Depth: 500m

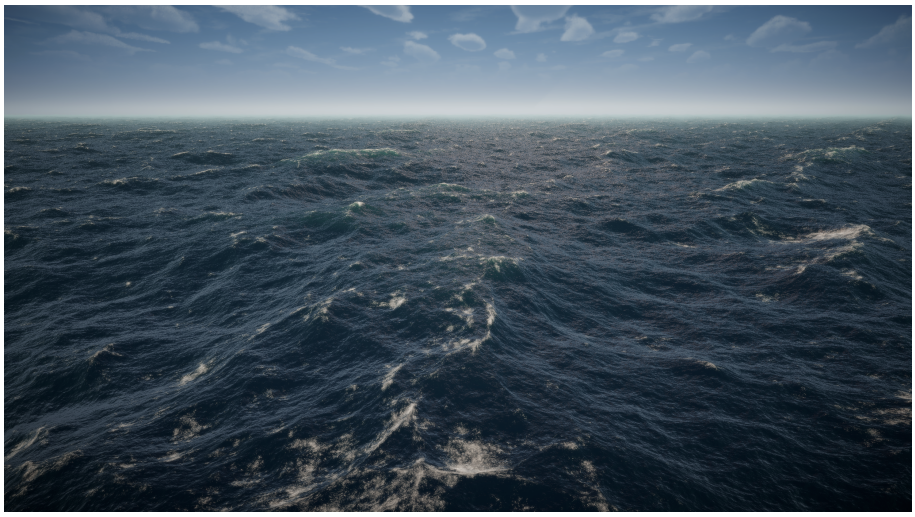

Figure 3.11: Spectrum: TMA, Size: 256x256, l_1: 600, l_2: 256, l_3: 10, λ: 1.3, U_{10}: 12 m/s, Fetch: 1000km, Depth: 500m

3.3 Limitations

During the implementation of our FFT-based ocean generation technique, we encountered challenges related to the global application of FFT-generated maps. This global application restricts us from selectively altering specific sections of the sea, such as the wake created by a moving ship or the reduced wave height near a shallow beach.

To address these challenges, J. Tessendorf proposed a hybrid approach in his paper "iWaves" (2004) [20]. This approach combines grid-based ocean generation, where each vertex point is propagated individually and FFT waves serve as ambient waves. Tessendorf later enhanced this approach with the release of “eWaves” [21].

These challenges underscore the complexities involved in ocean wave simulation and the potential of hybrid approaches in advancing the field. The proposed hybrid approach effectively addresses the identified challenges, demonstrating its potential in enhancing the realism and interactivity of ocean wave simulations.

Chapter 4

Discussion

4.1 Conclusions

Our research has demonstrated the significant role of the Fast Fourier Transform (FFT) in the field of ocean rendering. The FFT, when used in conjunction with the TMA spectrum, which is grounded in empirical data, provides a remarkably efficient and persuasive approximation of stormy and calm oceanic scenes. This combination has been instrumental in achieving substantial enhancements in speed, making real-time graphics feasible, which would otherwise be unattainable with the use of the Discrete Fourier Transform (DFT) algorithm.

However, despite the efficiency of FFT-based ocean rendering, it alone is not sufficient to produce an ocean without noticeable tiling. To address this issue, we developed a technique that involves rendering multiple cascades for different wavelengths and combining them. This innovative approach effectively renders the tiling inconspicuous, thereby enhancing the visual quality of our ocean. Additionally, this method reduces computational expense as we can render multiple smaller textures instead of one large texture.

One of the most significant challenges we encountered with FFT-based ocean rendering is its limited interactivity with surroundings. This limitation could potentially affect the realism of the scene. As suggested by Jerry Tessendorf (2004) [20], a hybrid approach that combines FFT with other techniques could be a potential solution to this problem, and is an area worth exploring in future research.

For the shading of the ocean, we utilized a modified version of Phong shading. This method yielded satisfactory results, providing a good balance between visual quality and computational efficiency. However, to further enhance the realism of the ocean and achieve a more convincing representation, it would be prudent to consider transitioning to Physically-Based Rendering (PBR). This advanced rendering technique, which simulates the physical properties of light and materials, could potentially provide a more realistic and visually appealing depiction of the ocean.

4.2 Ideas for future work

4.2.1 Enhancing Ocean Interactivity

Building upon the limitations identified in previous sections, a pivotal direction for future research lies in augmenting the interactivity of the ocean model. This enhancement would enable the water to dynamically respond to environmental changes such as variations in depth, the introduction of obstacles, and the influence of external forces like wakes generated by ships.

The realization of this enhanced interactivity can be guided by the methodologies outlined in the seminal works of Tessendorf (2004) [20] and Tessendorf (2014) [21]. These papers provide a robust foundation for the development of advanced ocean simulation techniques, offering a

promising pathway towards a more realistic and interactive ocean model.

4.2.2 Incorporation of Foam Spray Representation

The current model's representation of the ocean surface lacks a crucial visual element: foam spray, which is typically observed during significant wave crashes. This omission highlights a limitation in the model's ability to accurately simulate real-world oceanic conditions. To address this, we propose the integration of a particle system specifically designed to simulate the foam spray effect.

A potential approach to this challenge could be the implementation of a GPU-based particle system, as suggested by Kipfer et al. (2004) [22]. This method would leverage the computational power of modern GPUs to generate a large number of particles in real-time, thereby creating a visually complex and realistic representation of foam spray.

4.2.3 Transition to Physically-Based Rendering (PBR)

A substantial enhancement to the current model would involve transitioning from the custom Phong model to Physically-Based Rendering (PBR). This transition promises a more accurate depiction of environmental interactions, such as the ocean's response to varying light conditions like the setting sun and nocturnal environments, as well as improved specular lighting effects.

The successful implementation of PBR can be informed by the insights provided in the presentation by Mihelich and Tcheblokov (2021) [15], as well as the comprehensive courses offered by Hill and McAuley (2012) [16]. These resources provide valuable guidance on the principles and practicalities of PBR, offering a robust foundation.

References

[1] Franz Gerstner. Theorie der wellen. *Annalen der Physik*, 32(8):412–445, 1809.

[2] Jean Baptiste Joseph Fourier. *Théorie analytique de la chaleur*, volume 1. Gauthier-Villars, 1822.

[3] Jerry Tessendorf et al. Simulating ocean water. *Simulating nature: realistic and interactive techniques. SIGGRAPH*, 1(2):5, 2001.

[4] Christopher J Horvath. Empirical directional wave spectra for computer graphics. In *Proceedings of the 2015 Symposium on Digital Production*, pages 29–39, 2015.

[5] Klaus Hasselmann, Tim P Barnett, E Bouws, H Carlson, David E Cartwright, K Enke, JA Ewing, A Gienapp, DE Hasselmann, P Kruseman, et al. Measurements of wind-wave growth and swell decay during the joint north sea wave project (jonswap). *Ergaenzungsheft zur Deutschen Hydrographischen Zeitschrift, Reihe A*, 1973.

[6] Willard J Pierson Jr and Lionel Moskowitz. A proposed spectral form for fully developed wind seas based on the similarity theory of sa kitaigorodskii. *Journal of geophysical research*, 69(24):5181–5190, 1964.

[7] Steven A Hughes. The tma shallow-water spectrum description and applications. 1984.

[8] Sergej A Kitaigordskii, VP Krasitskii, and MM Zaslavskii. On phillips' theory of equilibrium range in the spectra of wind-generated gravity waves. *Journal of Physical Oceanography*, 5(3):410–420, 1975.

[9] Edward F Thompson and Charles Linwood Vincent. Prediction of wave height in shallow water. In *Coastal Structures' 83*, pages 1000–1007. ASCE, 1983.

[10] Ian R Young. *Wind generated ocean waves*. Elsevier, 1999.

[11] Carl Friedrich Gauss. Nachlass: Theoria interpolationis methodo nova tractata. *Carl Friedrich Gauss Werke*, 3:265–327, 1866.

[12] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[13] Bui Tuong Phong. Illumination for computer generated pictures. In *Seminal graphics: pioneering efforts that shaped the field*, pages 95–101. 1975.

[14] Joe Wilson. Physically-based rendering, and you can too! 2017. `https://marmoset.co/posts/physically-based-rendering-and-you-can-too/`.

[15] Mark Mihelich and Tim Tcheblokov. Wakes, explosions and lighting: Interactive water simulation in atlas. 2021. `https://youtu.be/Dqld965-Vv0?si=Jy0_un81KjUsWmk6`.

[16] Stephen Hill and Stephen McAuley. 2012-2020. `https://blog.selfshadow.com/publications/`.

[17] Libo Huang, Ziyin Qu, Xun Tan, Xinxin Zhang, Dominik L. Michels, and Chenfanfu Jiang. Ships, splashes, and waves on a vast ocean. *ACM Trans. Graph.*, 40(6), dec 2021.

[18] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[19] Fynn-Jorin Flügge. Realtime gpgpu fft ocean water simulation. 2017.

[20] Jerry Tessendorf. Interactive water surfaces. *Game Programming Gems*, 4(265-274):8, 2004.

[21] Jerry Tessendorf. ewave: Using an exponential solver on the iwave problem. *Technical Note*, 2014.

[22] Peter Kipfer, Mark Segal, and Rüdiger Westermann. Uberflow: a gpu-based particle engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122, 2004.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

In the initial stages, a comprehensive evaluation of multiple libraries was conducted to select the most suitable one for our project. This selection process was underpinned by a thorough understanding of the theoretical aspects of FFT waves, which we developed by reviewing numerous research papers. To further solidify our foundational knowledge, we created a naive sum of sins project to understand the basics of ocean generation.

With a detailed workflow and a graphed algorithm in place, we embarked on the project, ensuring that unnecessary refactoring was minimized and adherence to the project timeline was maintained. The project was initially implemented in a simplistic manner to grasp the underlying theory. Once a solid understanding was established, the project was advanced to a more complex level to optimize computational efficiency. Throughout this process, data visualization was employed at every step to ensure the absence of bugs and validate the accuracy of our results.

Upon completion of the project, the results produced were compared with real-world data. This comparison confirmed the realism of the ocean generated by our project, attesting to the success of our approach. Thus, the project execution demonstrated a high degree of planning, theoretical understanding, practical implementation, and validation, resulting in a realistic and efficient ocean generation technique.

A.2 Personal reflection and lessons learned

This project served as a comprehensive learning experience, enhancing our technical skills, academic research abilities, professional communication, and project management strategies. We learned the importance of clear and timely communication, particularly with superiors, when the project was initially undertaken in Unity without prior approval. This oversight was later rectified by submitting a detailed request analyzing different approaches.

The project also provided an opportunity to delve into academic research, enhancing our ability to effectively search for and read research papers. On the technical front, we gained experience in using compute shaders and expressing complex mathematical equations, which was instrumental in creating the FFT algorithm and generating a realistic ocean.

Additionally, the project honed our report writing skills and reinforced the necessity of obtaining necessary permissions before embarking on significant project decisions.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

In the context of this project, the primary legal considerations pertain to the use of Unity, a third-party tool. As the entirety of the codebase was authored independently, there are no concerns regarding the infringement of external code licenses. However, it is crucial to adhere to the terms and conditions stipulated by Unity's license agreement. This adherence ensures the lawful utilization of Unity's resources and capabilities, thereby aligning the project with the requisite legal standards.

A.3.2 Social issues

The potential applications of this project, particularly in domains such as video games or cinematic productions, could have notable social implications. Specifically, the manner in which the ocean is represented and rendered using this implementation could influence societal perceptions of marine environments. As such, it is crucial to ensure that the oceanic simulations generated are as accurate and realistic as possible, to foster an authentic understanding and appreciation of our oceans.

A.3.3 Ethical issues

In the realm of ethical considerations, it is imperative to ensure that the ocean simulation does not misrepresent or oversimplify the inherently complex marine phenomena. Such misrepresentations could potentially lead to misunderstandings or misuse of the work, thereby violating ethical guidelines of accuracy and truthfulness in scientific representation.

Furthermore, the consideration of making this project open-source aligns with the ethical principle of knowledge sharing in the academic and scientific community. By doing so, the project could serve as a learning resource for others, promoting transparency, collaboration, and collective learning.

A.3.4 Professional issues

A key professional consideration in this project is the adherence to industry standards and best practices in coding and documentation. This adherence ensures the maintainability, readability, and scalability of the code, thereby enhancing its longevity and usability.

Furthermore, it is crucial to stay abreast of the latest advancements in the field. This includes keeping up-to-date with new versions or features of Unity, advancements in FFT algorithms, or GPU programming techniques. Such continual learning and adaptation are essential for maintaining the relevance and effectiveness of our work in a rapidly evolving field.

Appendix B

External Material

B.1 Skyboxes

Additional skyboxes utilized in this project were procured from the Unity Asset Store. The specific asset employed is freely available and grants comprehensive permissions for unrestricted usage. This approach aligns with the principles of ethical use of third-party resources in academic projects. The asset can be accessed at the following URL: `https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-series-free-103633`.

Appendix C

Additional Results

The outcomes of the conducted experiments, encapsulated in video format, are accessible in the designated Git repository.