School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Realistic Ocean Simulation using Fourier Transform

Saulius Vincevičius

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 (DD/MM/YY)
<Example> Scanned participant consent forms	PDF file / file archive	Uploaded to Minerva (DD/MM/YY)
<Example> Link to online code repository	URL	Sent to supervisor and assessor (DD/MM/YY)
<Example> User manuals	PDF file	Sent to client and supervisor (DD/MM/YY)

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) ____________________

Summary

Other Solutions:

- Sum of sines
- Gerstner waves
- Particle based ocean simulation

Problems:

- Computaiontally expensive to calculate huge amouts of sins.
- Gerstner waves are not physically accurate.
- Both are not easily modifiable.
- Does not look good with stormy weather.
- Tiling is noticable.
- Lacks water detail.

Problems I intend to solve:

- Simulate water based on emipirical data for a more realistic look.
- Have an easy modifiable ocean system.
- Have an ocean that can simulate stormy weather.
- Reduce tiling.
- Have a more detailed ocean.

Main achivements:

- Calculate on GPU.
- Implement JONSWAP spectrum.
- Implement FFT algorithm.
- Simulate foam.
- Shade the ocean.

Acknowledgements

<The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.>

Note that it is not acceptable to solicit assistance on ‘proof reading’ which is defined as the “the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test”; see
`https://www.leeds.ac.uk/secretariat/documents/proof_reading_policy.pdf`

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 inteactivity and visual apeal, while in movies it is used for visual apeal.

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

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.

[Missing Logic Linking]

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 desireble 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 Fourier Transform

The Fourier Transform is a mathematical method that enables us to convert our data from the time domain to the frequency domain, and vice versa. To simplify, imagine we have a smoothie that's too sour. With the Fourier Transform, we could deconstruct our smoothie (time domain) into its ingredients (frequency domain), remove the sour component, and then reconstruct it back. It was invented by Joseph Fourier[1] in 1822, altough at his time it didn't have much practical use, however, nowdays it is used in many fields, including signal processing, 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):

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

1.3 Fourier Transform Ocean

A significant challenge associated with the utilization of Gerstner waves, as will be elaborated upon in the related works section, 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 thousands vertices. In response to this computational demand, J. Tessendorf [2] proposed a method for generating ocean waves using the Fourier Transform.

The main idea is to generate a height map of the ocean surface in the frequency domain, and then convert it back to the time domain using the IFT. Since the IFT produces a periodic height map, we can tile it to create an infinite ocean surface. This means that we can generate one texture and apply it to the entire water body.

The advantage of this method becomes apparent when considering a 512x512 texture, which combines 262,144 distinct waves. In contrast, Gerstner waves experience noticeable performance loss after 65 waves.

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 that 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 complex number made from real and imaginary part that was drawn from a gaussian random number generator, with mean 0 and stardart deviation 1.

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"[3]:

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

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, where $\mathbf{x} = (x, y)$ is the position in the texture.

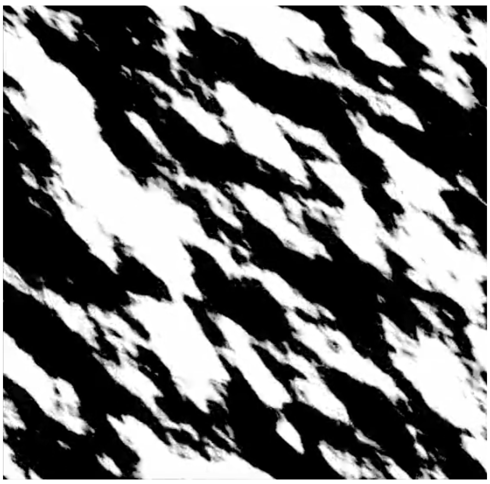

Figure 1.2: Height Map using J. Tessendorf's Spectrum

1.4 Ocean Spectrums

1.4.1 Jerry Tessendorf's Ocean Spectrum

The main thing that defines how ocean will looks is the spectrum. Inside Jerry Tessendorf's [2] paper, he proposed modified Phillips spectrum 2.3:

$$P_h(\mathbf{k}) = A\frac{e^{-1/(kL)^2}}{k^4}|\hat{\mathbf{k}} \cdot \mathbf{w}|^6 \tag{1.8}$$

where A is numeric constant, $L = V^2/g$ is the largest possible wave arising from wind speed V and gravity g. The $|\hat{\mathbf{k}} \cdot \mathbf{w}|^6$ is used to surpress waves that are moving perpendicular to the wind direction. However, J. Tessendorf admits that this spectrum has "poor convergence properties at high values of the wavenumber $|\mathbf{k}|$ and suggests to surpress waves that are $l \leq L$, where l is some small constant and multiply the Phillips spectrum by $e^{-k^2l^2}$.

1.4.2 JONSWAP Spectrum

Regarding all the issues that we experiance with "Phillips Spectrum", Horvath Christopher [3] proposed to use JONSWAP spectrum. JONSWAP stands for "Joint North Sea Wave Project" and was developed by Hasselmann [4, et al. in 1973]. This spectrum is improved version of Pierson-Moskowitz Spectrum [5] when Horvath noticed that the wave spectrum is never fully developed. Therefore he added extra peak enchancement factor γ^r.
The JONSWAP spectrum is defined as follows:

$$
\begin{aligned}
S_{\text{JONSWAP}}(\omega) &= \frac{\alpha g^2}{\omega^5} \exp\left(-\frac{5}{4}\left(\frac{\omega_p}{\omega}\right)^4\right)\gamma^r \\
r &= \exp\left(-\frac{(\omega-\omega_p)^2}{2\sigma^2\omega_p^2}\right) \\
\alpha &= 0.076\left(\frac{U_{10}^2}{Fg}\right)^{0.22} \\
\omega_p &= 22\left(\frac{g^2}{U_{10}F}\right)^{1/3} \\
\gamma &= 3.3 \\
\sigma &= \begin{cases} 0.07 & \text{if } \omega \leq \omega_p \\ 0.09 & \text{if } \omega > \omega_p \end{cases}
\end{aligned}
\tag{1.9}
$$

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 angular frequency.

1.4.3 The Texel MARSEN ARSLOE (TMA) Spectrum

The biggest problem with with JONSWAP spectrum is that it only works with deep waters. Therefore, Horvath Christopher [3] proposed to use TMA correction for deep water spectrum. TMA was developed by Steven A. Hughes [6], based on the observations made by Kitaigordskii [7]. this is known as Kitaigordskii depth Attenuation function and it takes form:

$$
\begin{aligned}
\Phi(\omega_h) &= \left[\frac{(k(\omega,k))^{-3}\frac{\partial k(\omega,h)}{\partial\omega}}{(k(\omega,\infty))^{-3}\frac{\partial k(\omega,\infty)}{\partial\omega}}\right] \\
\omega_h &= \omega\sqrt{h/g}
\end{aligned}
\tag{1.10}
$$

where h is the depth of the water. At first glance this function seems to be difficult, however looking at 1.3 we can see that this can be easilly approximated.

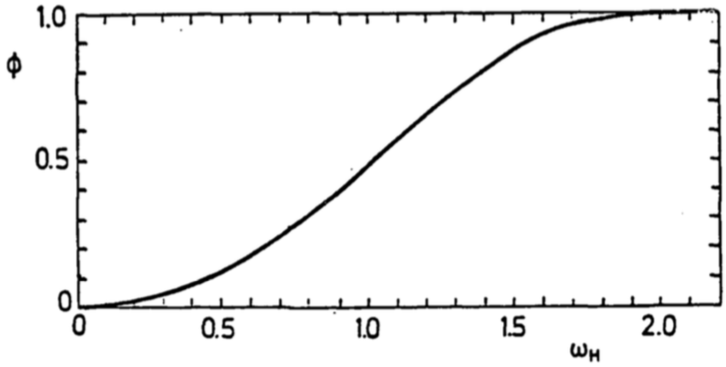

Figure 1.3: $\Phi(\omega, h)$ as a function of ω_h [6]

Approximation given by Thomson and Vincent [8] is:

$$\Phi(\omega, h) \approx \begin{cases} \frac{1}{2}\omega_h^2 & \text{if } \omega_h \leq 1 \\ 1 - \frac{1}{2}(2 - \omega_h)^2 & \text{if } \omega_h > 1 \end{cases} \tag{1.11}$$

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

$$S_{\text{TMA}}(\omega, h) = S_{\text{JONSWAP}}(\omega)\Phi(\omega, h) \tag{1.12}$$

Donelan-Banner Directional Spreading

Currentlly we can't use the TMA sprectrum for our project as it has few problems. Firstlly, this spectrum is non-directional, so we need to add directionality to it. Secondly, this spectrum accepts ω as input, but as we following J. Tessendorf's [2] paper, we need to use $\mathbf{k}$ as input. To solve this problem Horvath Christopher [3] proposes to use Donelan-Banner Directional Spreading [9]:

$$D(\omega, \theta) = \frac{\beta_s}{2\tanh(\beta_s \pi)} \operatorname{sech}(\beta_s \theta)^2 \tag{1.13}$$

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:

$$D_{\text{TMA}}(\omega, \theta) = S_{\text{TMA}}(\omega) \cdot D(\omega, \theta) \tag{1.14}$$

TMA transformation

Lastlly, to make this spectrum usable we need to transform it from ω to $\mathbf{k}$ and according to Horvath Christopher [3] we can transform it like this:

$$S_{\text{TMA}}(\mathbf{k}) = 2S_{\text{TMA}}(\omega, h) \cdot \frac{d\omega}{dk} / k \cdot \Delta k_x \cdot \Delta k_y \tag{1.15}$$

where in our case,

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

1.5 Cooley-Tukey Fast Fourier Transform (FFT)

When simulating fourier transform ocean majority times is spent on converting from frequency domain to time domain, in other words performing Inverse Fourier Transform. Earlier mentioned IDFT 1.4 has time complexity of $O(n^2)$, and it's clearly that this algorithm is not sufficient to simulate heiger resolution ocean surfaces in real time. In 1805 Gauss Carl Friedrich [10] invented FFT algorithm $O(n \log n)$ and in 1965 Cooley James W. and Tukey John W. [11] reinvented it. The FFT algorithm is based on that there is redundancy in the computation of the DFT, and that we can exploit it to reduce the time complexity, it is important to mention that the FFT algorithm works only 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 sets, one containing the even samples and the other containing the odd samples and repeat this proccess until a set has 2 samples (bit-reverse sort order).
2. Generate twiddle factor $W_N = e^{-i2\pi k/N}$ and rise it to the power of

$$k = i * N/2^{\text{stage}+1} \mod N \tag{1.16}$$

   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

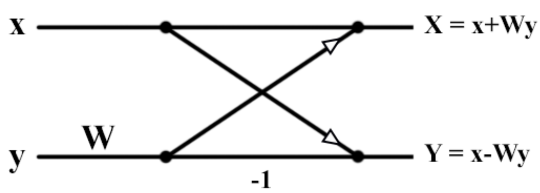

Figure 1.4: Butterfly Diagram

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.

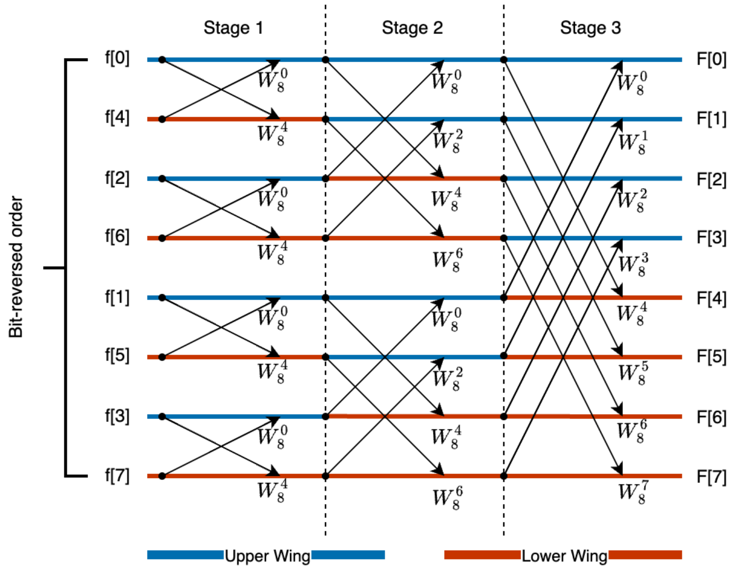

Figure 1.5: 8-point FFT Butterfly Diagram

1.6 Ocean Shading

1.6.1 Phong Shading

The Phong shading model, one of the simplest yet effective methods for approximating realistic shading, was proposed by [12, Phong Bui Tuong, 1975]. This model comprises three components: ambient shading, diffuse shading, and specular shading, as depicted in Figure 1.6. The ambient component is a constant value representing the light that is scattered about the entire scene. The diffuse component, on the other hand, can be calculated using the following equation:

$$\text{Diffuse} = \max(\text{Normal} \cdot \text{LightDirection}, 0.0); \tag{1.17}$$

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

$$\text{Specular} = max(\text{ViewDir} \cdot \text{ReflectionDir}, 0.0)^{\text{Shininess}} \tag{1.18}$$

In this equation, the ViewDir is a normalized vector pointing towards the camera, and the ReflectionDir is a normalized reflection vector with respect to the Normal, as illustrated in Figure 1.7.

Figure 1.6: Phong Shading
Credits: learnopengl.com

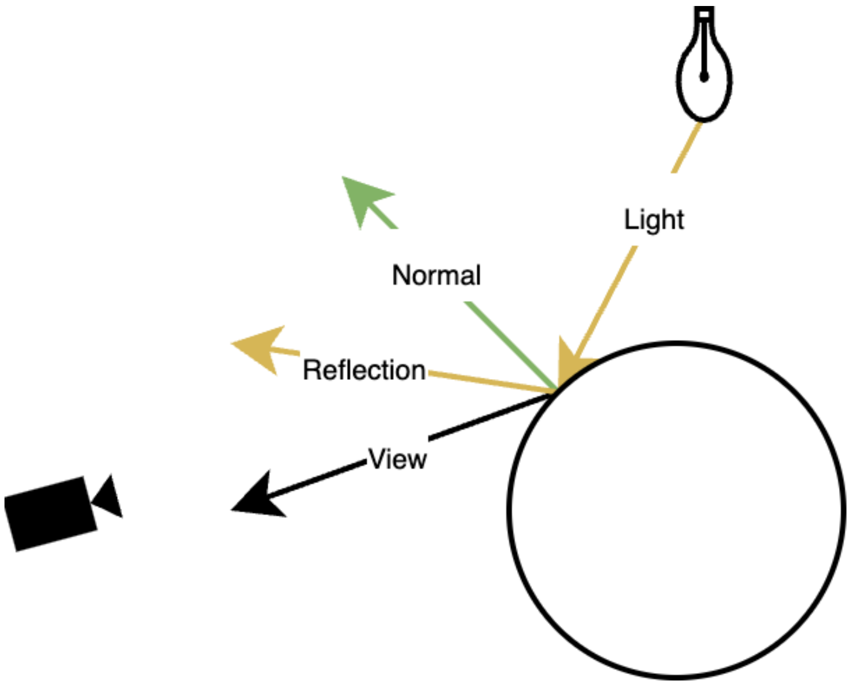

Figure 1.7: Phong Vectors

1.7 Related Work

1.7.1 Gerstner Waves

Gerstner waves provide a simple model for generating somewhat realistic ocean waves. This model is straightforward to implement and is predominantly used in video games. Even major titles, such as "Pokemon Legends: Arceus", utilize something similar to Gerstner waves to simulate oceanic environments. The concept of Gerstner waves was first introduced by F.J. Gerstner [13] in 1802. The initial known implementation of Gerstner waves in computer graphics was carried out by Fournier and Reeves [14] in 1986. The model is an attempt to approximate the motion of ocean waves, based on the principle that each point in the ocean undergoes a circular motion. If a point is represented as $\mathbf{x_0} = (x_0, z_0)$, the height and horizontal displacements can be calculated as follows:

$$\mathbf{x} = \mathbf{x_0} - (\mathbf{k}/k)A\sin(\mathbf{k} \cdot \mathbf{x_0} - \omega t) \tag{1.19}$$

$$y = A\cos(\mathbf{k} \cdot \mathbf{x_0} - \omega t) \tag{1.20}$$

$$\omega = \sqrt{gk} \tag{1.21}$$

Here, A is the amplitude, ω is dispertion relationship, and $\mathbf{k}$ is a wave vector that points horizontally in the direction of wave travel and is proportional to the wavelength λ of the wave.

$$k = 2\pi/\lambda \tag{1.22}$$

As presented, the Gerstner waves model is limited to a single wave. To simulate more complex ocean surfaces, multiple waves need to be summed together. This is achieved by summing the waves as follows:

$$\mathbf{x} = \mathbf{x_0} - \sum_{i=1}^{N} (\mathbf{k_i}/k_i) A_i \sin(\mathbf{k_i} \cdot \mathbf{x_0} - \omega_i t) \tag{1.23}$$

Here, N is the number of waves, A_i is the set of amplitudes, ω_i is the set of frequencies, and $\mathbf{k}_i$ is the set of wave vectors. The primary challenge with this approach is that simulating realistic ocean surfaces requires summing a large number of waves together, which can be computationally expensive. Furthermore, Gerstner waves offer limited artistic control, has visible tiling 1.8, and underperforms in simulating stormy weather conditions.

Figure 1.8: Water Tiling in Pokemon Legends: Arceus

1.7.2 Particle Simulation and Machine Learning

Particle-based methodologies are frequently employed for the simulation of water in a manner that is both realistic and visually compelling. However, these methods are associated with a high computational cost.

Recent advancements in Graphics Processing Unit (GPU) technology, in conjunction with research such as that conducted by Libo Huang [15], are incrementally making particle simulations more viable for ocean simulation. Despite this progress, these techniques demand a high-performance GPU and are not yet amenable to widespread real-time applications.

Moreover, there have been considerable advancements in the application of machine learning to accelerate fluid simulations, as demonstrated by the work of Dmitrii Kochkov [16]. However, this is out of this projects scope and won't be disccused.

1.7.3 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 [17, Joe Wilson, 2017]. The PBR rendering model adheres to three primary principles: energy conservation, microfacet support, and the Fresnel effect.

[ELABORATE]

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 [18], 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 [19] offers comprehensive coverage of the topic from 2012 to 2020.

Chapter 2

Methods

2.1 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. The decision was influenced by Unity's comprehensive suite of tools, including UI design, GPU programming, and a built-in build system. These features significantly streamline the development process, making Unity an optimal choice for our project's needs.

2.2 Utilization of Unity Tools

In our project, we strategically employed Unity's toolset. 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.

2.3 Version Controll

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.

2.4 Algorithm Overview

The algorithm can be split into 3 main parts as shown in figure 2.1

1. Spectrum generation (Only calculated on parameter value change)
2. Frequency generation
3. Convertion from frequency to time domain using IFFT

As shown in figure 2.1 majority of calculations are done on GPU using HLSL executed inside Unity. As FFT algorith requires input size to be 2^n we going to use 512x512 textures as this provides good performance and high enought details. For simplicity in this project we assume that the texture size is NxN, where N is the size of a texture and N is power of 2.

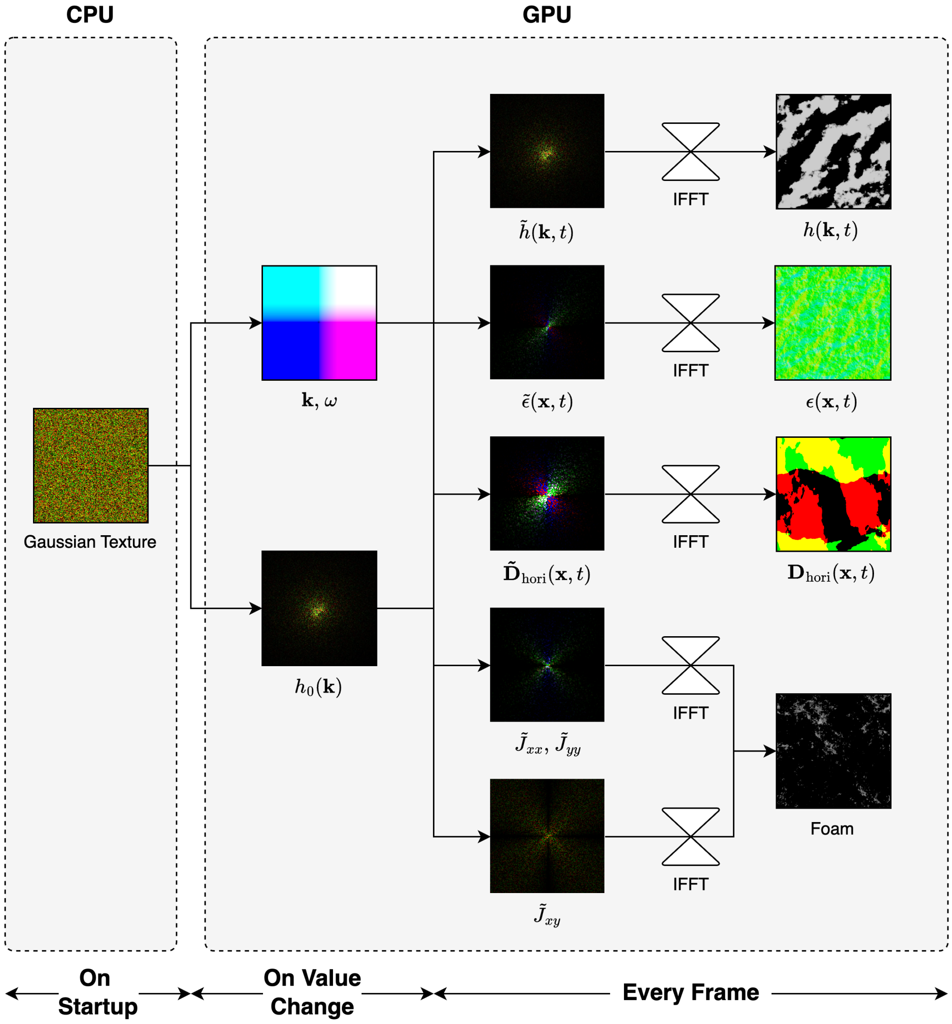

Figure 2.1: Ocean Algorithm

2.5 IFFT

2.5.1 Butterfly Texture 2.2

Firstlly we only intrested in IFFT algorithm and we will not use FFT. To perform IFFT we going to produce so called butterfly texture by [20, Flügge Fynn-Jorin]. This texture has width of $(log_2(n)$ and height of n and is precomputed and stored inside the GPU memory once, unless the data size changes. This is 4 chanel texture (W_r, W_i, y_t, y_b), where W_r and W_i are real and imaginary parts of the twiddle factor, y_t and y_b are top and bottom butterfly indices as shown in 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 assign y_t and y_b in bit reversed order, on the other stages:

- If the butterfly wing is top half

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

- 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 if the wing is upper or lower half:

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

Lastlly we need to calculate twiddle factors:

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

2.5.2 Performing IFFT

Currentlly our data is stored in 2D texture. While this algorithm only accepts 1D data. Therefore, we need to perform 1D IFFT on each row "horizontally" and then on each column "vertically" 2.2. By following pseudocode from [20] 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

Lastlly, our data needs to be permuted as you will see later our data is offseted:

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

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 [20]

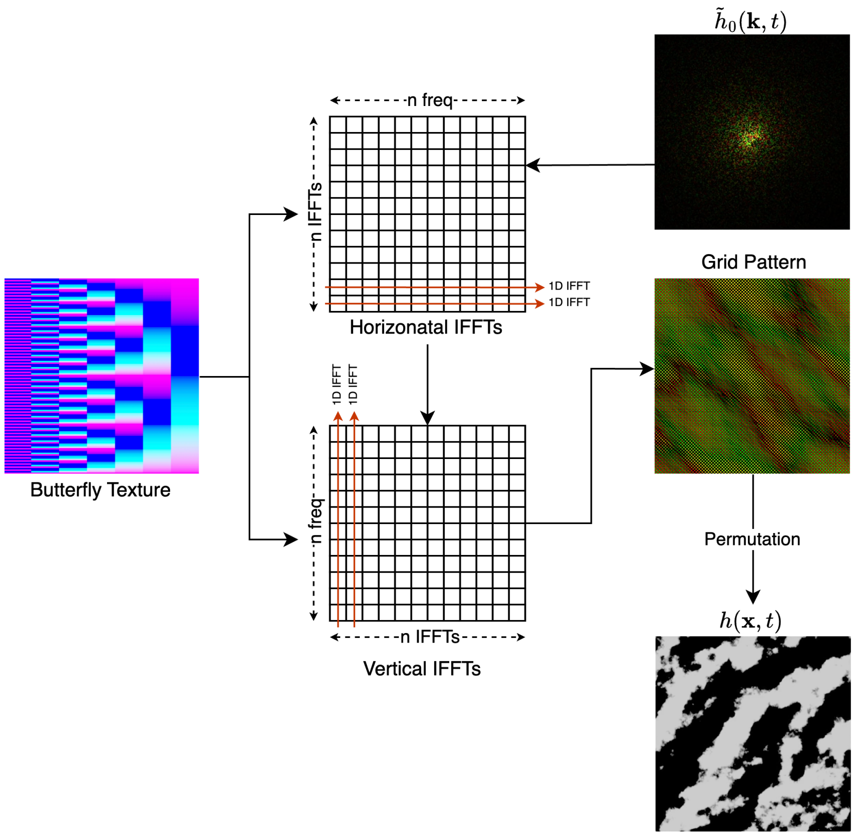

Figure 2.2: IFFT Algorithm

2.6 Ocean Geometry

2.6.1 Spectrum Generation

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	Magnetude of wave vector
l	Length scale of the ocean
ω	dispertion relationship (angular frequency)
U_{10}	Wind speed at 10m above the sea level
F	Fetch (Disntance from lee shore)
θ_{wind}	Wind angle
λ	Choppy factor

Table 2.1: Deffinition Table

Spectrums are responsable for whole look of the ocean. It's importatnt to pick the one that is based on real world data to make our ocean look realistic. Initialy, Tessendorf spectrum 1.8 was implemented as it was straight forward to implement. However, it didn't produce satisfying results, as waves didn't seem to transfer energy in convincing way, waves didn't seem to follow wave direction and it was hard to have artistic control over the ocean.

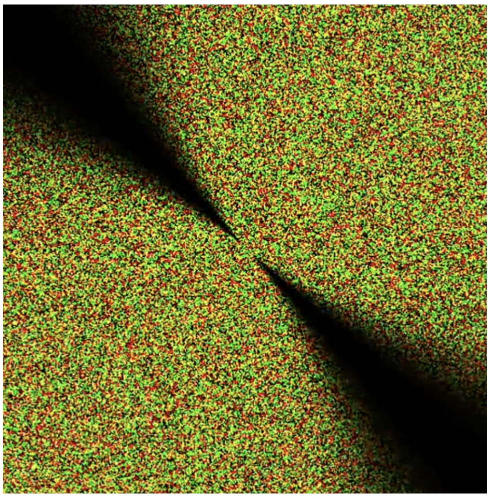

Figure 2.3: Jerry Tessendorf's Spectrum

Therefore, TMA spectrum 1.15 was 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.

This spectrum resulted in more relistic looking ocean with intuitive controls: fetch, wind speed, wind angle, depth. Moreover, it didn't require any efort to make the ocean look relistic and it just worked out of the box.

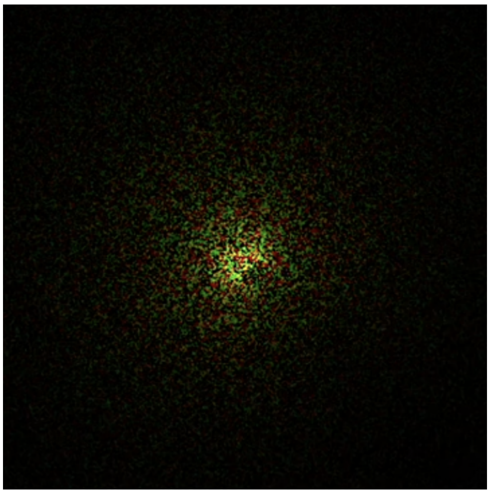

Figure 2.4: TMA spectrum, Frequency Domain
(100x for better visibility)

2.6.2 Height Map Generation

To generate height map, firstlly we need to have fourier amplitudes as shown in 1.5, where P_h is TMA Spectrum $S_{TMA}(\mathbf{k})$. For the next step we need to add fourier amplitude and it's complex conjugate to produce "produce waves towards and against the wave direction when propagating"[3]. In our luck we don't need to recalculate complex conjugate as fourier series are symetric, so we can just mirror the amplitudes:

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

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.

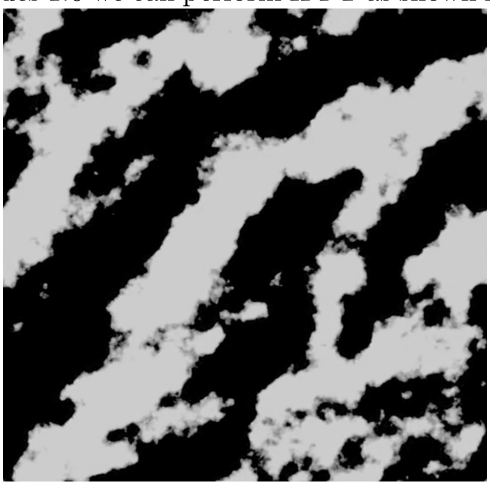

Figure 2.5: Height Map using S_{TMA}

2.6.3 Normal Map Generation

Latter we will need extra information about the ocean to calculate ocean shading. Therefore, we need to calculate normal map. Normal map is perpendicular direction to the surface of the ocean. To calculate normal map we need to calculate gradient, which is derivative of the height map. According to [21] the derivative is:

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

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

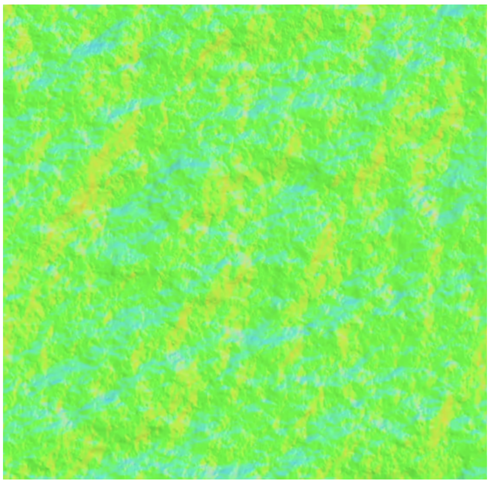

Figure 2.6: Normal Map using S_{TMA}

2.6.4 Choppy Waves

Currentlly, when rendering ocean it looks too smooth 2.7 and lacks choppiness.

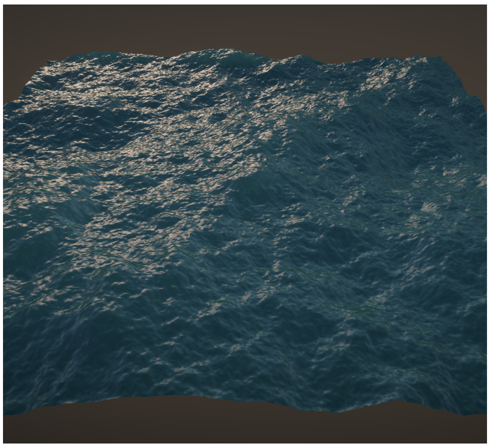

Figure 2.7: Ocean without Choppy Waves

To make waves choppy we need to introduce horizotal displacement. This will not only make waves more chopy but also make energy transfer between waves more relistic. Following formula from [21, J. Tessendorf] we can calculate horizontal displacement:

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

Using this horizotal displacement we can displace our verticies:

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

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

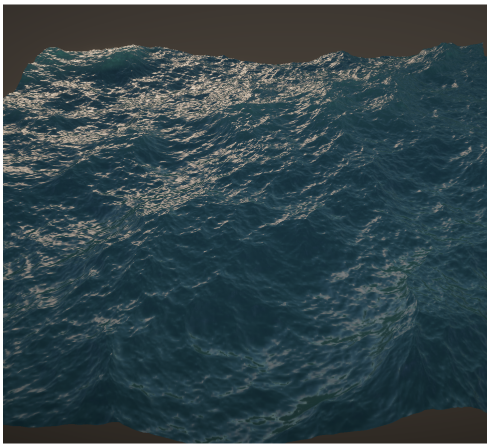

Figure 2.8: Ocean with Choppy Waves

2.7 Ocean Shading

Symbol	**Parameter**
L_a	Ambient Light
$L_s s$	Subsurface Scatter Light
L_s	Specular Light
L_r	Enviroment Reflection
N	Normal
D_s	Sun Direction
D_v	View Direction
C_a	Ambient Light Color
C_l	Light Color
C_b	Air Bubble Color
C_{ws}	Water Scattering Color
H	Ocean Height
F	Fresnel Effect
ρ_a	Air Bubble Density
k_a	Ambient Light Intensity
k_{ss_1}	Subsurface Scattering Intensity
k_{ss_2}	Subsurface Scattering Intensity

Table 2.2: Lighting Parameters

2.7.1 Lighting Model

Output

[MENTION PHON SHADING DISCCUSED IN BACKGROUND RESERCH]

Our lighting model, as depicted in Equation 2.11, comprises four integral components: ambient light, subsurface scattering, specular light, and environmental reflection. The culmination of these components results in the final output, as illustrated in Figure 2.9

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

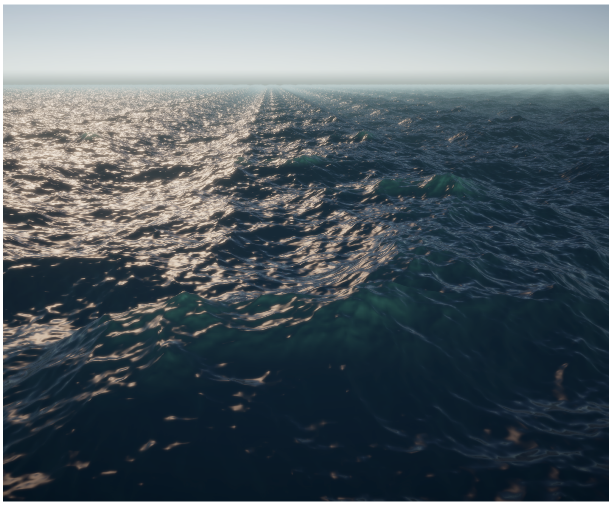

Figure 2.9: Finall Shader Output

Ambient Light

Ambient light, 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 [18]:

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

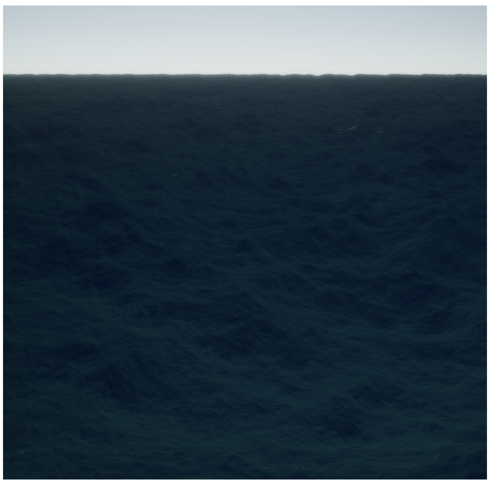

Figure 2.10: Ambient Light

Fresnel Effect

The Fresnel effect, a phenomenon where light is more reflective at grazing angles as shown in figure 2.11, is crucial for calculations involving specular and reflection:

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

Figure 2.11: Fresnel Effect

Subsurface Scattering

[SAY AS EARLIER MENTIONED]

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 [18] due to the majority of our light being trapped in the ocean, with only light at wave peaks able to escape.

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

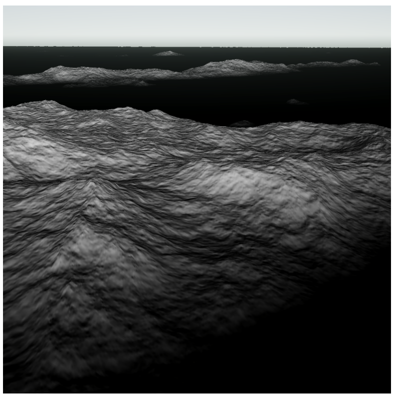

Figure 2.12: L_{ss_1}

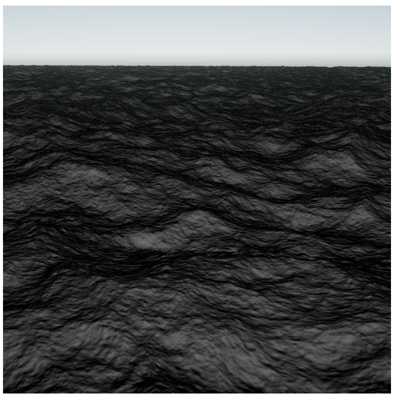

Figure 2.13: L_{ss_2}

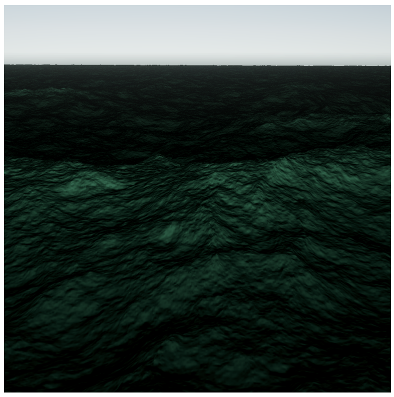

Figure 2.14: Subsurface Scattering

Specular Reflection

Specular reflection, the reflection of light from a light source on an object, is a characteristic of shiny objects like metals, plastic, or in our case, water. We employ a simple Phong scattering technique, as shown in Figure 2.15, and the formula takes the following form:

$$\begin{aligned} L_s &= C_l S C_s F \\ S &= [D_v, D_r]^{\text{Shininess}} \\ D_r &= \text{reflect}(\text{LightPos}, N) \end{aligned} \tag{2.15}$$

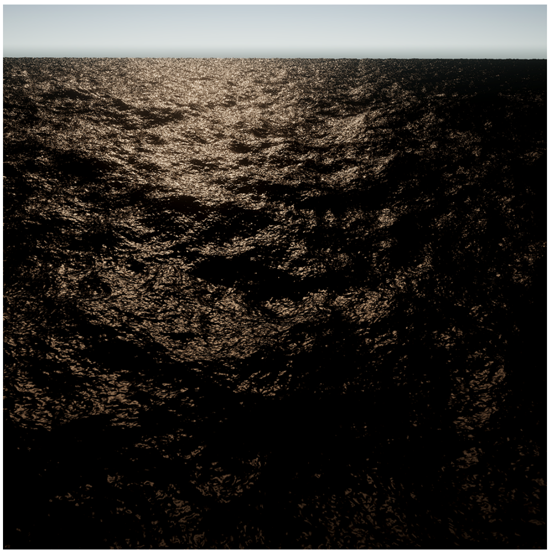

Figure 2.15: Specular Reflection

Enviroment Reflection

In our project, we only consider the reflection of the skybox on the water surface. As per Figure 2.16, we illustrate how skybox reflection operates on a calm ocean surface to better visualize environmental reflection.

$$\begin{aligned} L_r &= F k_r C_{\text{sky}} \\ C_{\text{sky}} &= \text{Sky}[\text{reflect} D_i, N] \end{aligned} \tag{2.16}$$

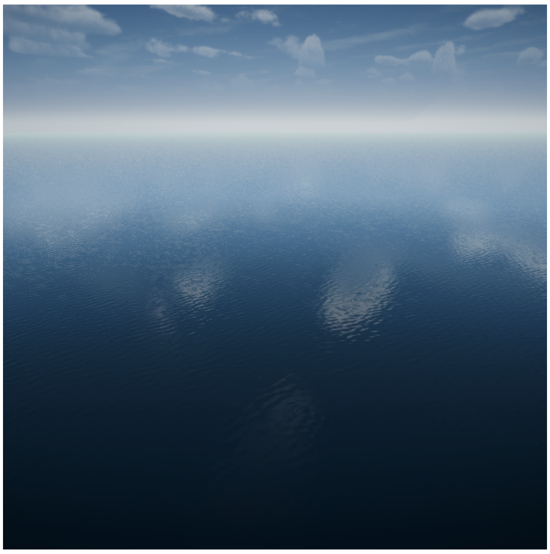

Figure 2.16: Skybox reflection on calm ocean

2.7.2 Foam

Foam Generation

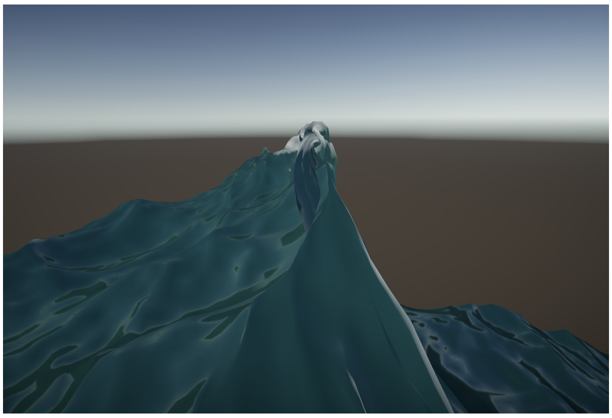

Figure 2.17: Wave Curl At Wave Peek

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 2.17. In essence, our horizontal transformation undergoes an inversion.

As proposed by Tessendorf [2], 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:

$$\mathrm{Det}(\mathbf{x}) = J_{xx} + J_{yy} - J_{xy}J_{yx} \tag{2.17}$$

where,

$$J(\mathbf{x}) = \begin{bmatrix} 1 + \lambda \frac{\partial D_x(\mathbf{x})}{\partial x} & 1 + \lambda \frac{\partial D_x(\mathbf{x})}{\partial y} \\ 1 + \lambda \frac{\partial D_y(\mathbf{x})}{\partial x} & 1 + \lambda \frac{\partial D_y(\mathbf{x})}{\partial y} \end{bmatrix} \tag{2.18}$$

in this case $J_{xy} = J_{yx}$.

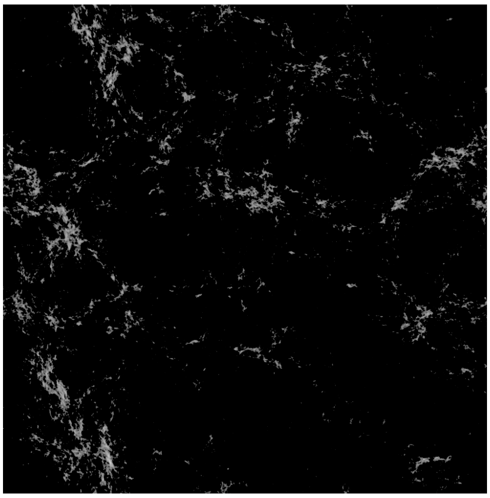

Figure 2.18: Foam Texture

Foam Accumilation

Curretlly the foam apears and diasapears quicly, however in real life the foam accumilates and disapears over time. We can introduce foam accumilation by compering previous and current foam values:

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

Listing 2.4: Foam Accumilation

This results in pleasing foam accumilation:

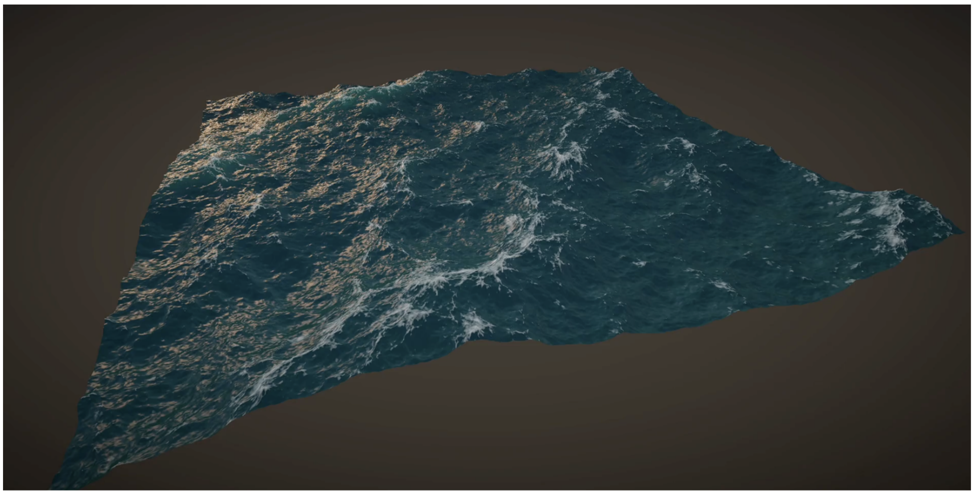

Figure 2.19: Ocean With Foam

2.8 Multiple Cascades

At this stage our ocean with texture 512x512 simulates 262,144 distinc waves however the tilling is still noticible 2.20.

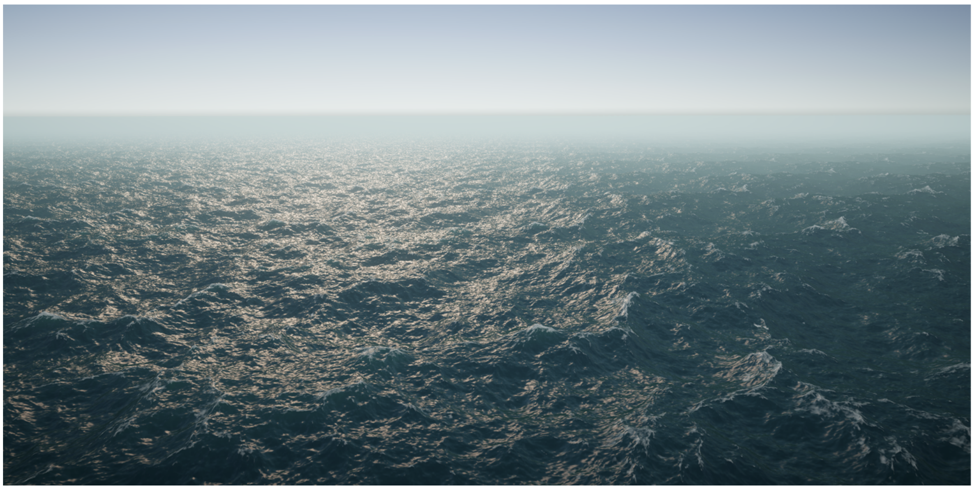

Figure 2.20: Ocean with Tilling, using 512x512 texture

To counter this we could increase our simulation texture however even FFT becomes expensive really fast. Another approuch is to simulate multiple cascades for diffrent wave lengths k 2.21 and diffrent l length scales. We can split our simulation into 3 parts, big waves, medium waves and small waves.

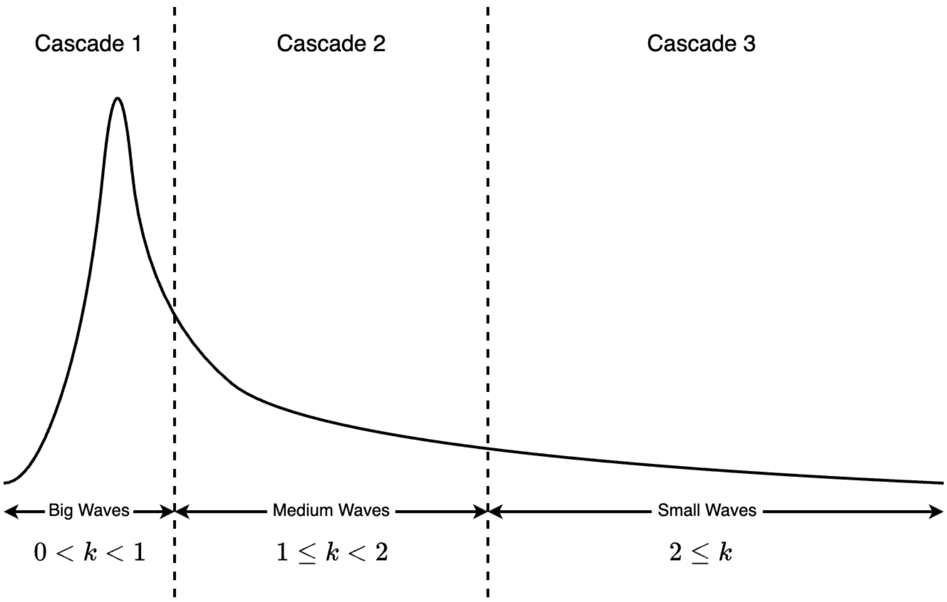

Figure 2.21: Cascades

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

1. We chose bigger l for bigger waves as we are more likly to notice tilling with big waves
2. We chose smaller l for smaller waves as we are less likly to notice tilling with small waves

This results ocean that has less tilling and more detail 2.22.

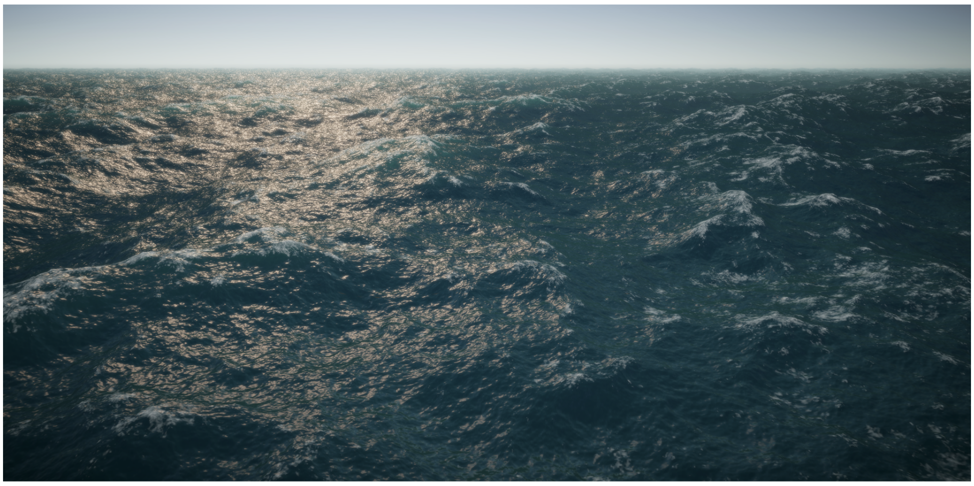

Figure 2.22: Ocean with Cascades, using 3x(512x512) textures

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) xtextures as shown in figure 2.23.

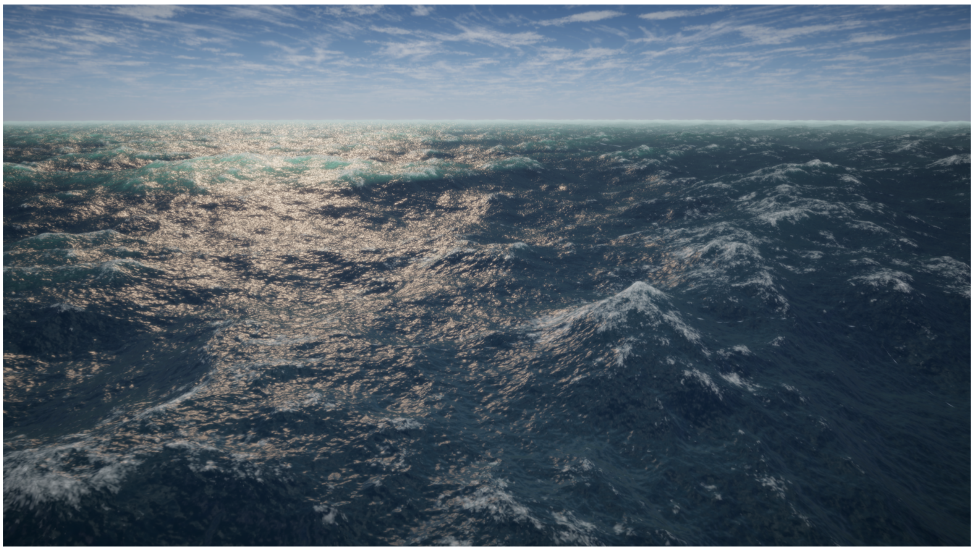

Figure 2.23: Ocean with Cascades, using 3x(256x256) textures

Chapter 3

Results

<Results, evaluation (including user evaluation) *etc.* should be described in one or more chapters. See the 'Results and Discussion' criterion in the mark scheme for the sorts of material that may be included here.>

3.1 Performance

The table below 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
M1	512x512	3	43.50ms
M1	1024x1024	3	128.25ms
Nvidia GeForce RTX 4070	256x256	3	1.65ms
Nvidia GeForce RTX 4070	512x512	3	3.29ms
Nvidia GeForce RTX 4070	1024x1024	3	10.70ms

Table 3.1: Performance on high and and low end GPU

The data reveals that for a texture size of 256x256, both GPUs are capable of rendering the ocean in real-time, demonstrating the broad applicability of our technique. However, as the texture size increases, the performance on the M1 GPU deteriorates significantly, indicating that our technique becomes computationally expensive for weaker GPUs at higher texture sizes.

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.

3.2 Comparison

3.2.1 Spectrums

The main diffrence comes to what kind of spectrum you use for FFT based oceans. The proposed "Phillips" spectrum 3.1 by [2, J. Tessendorf] has issues with energy transformation and following the wind direction. The proposed TMA spectrum 3.2 handles energy transformation way more relistic and does not have any issues following the wind direction. This is mostly because that TMA is based on empirical data. You can see clear diffrence between "Phillips" spectrum and TMA spectrum:

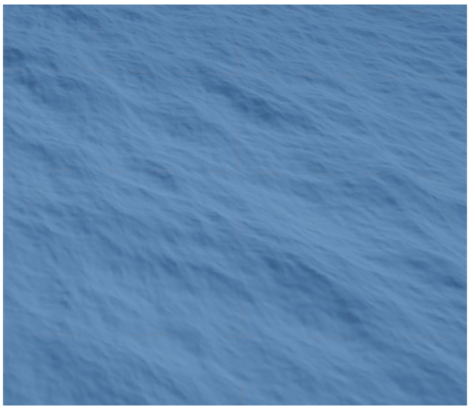

Figure 3.1: Height Map with P_h

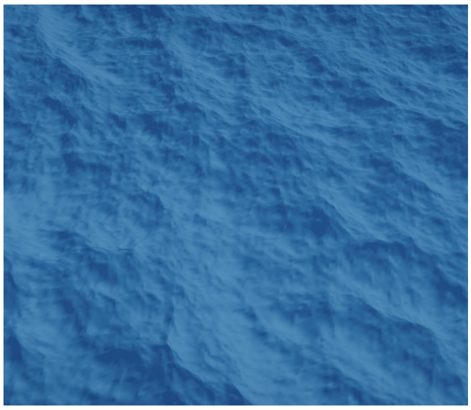

Figure 3.2: Height Map using TMA Spectrum

3.2.2 DFT and FFT

The computational complexity of the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) presents a significant contrast. With DFT operating at O(n2) and FFT at a more efficient O(nlog(n)), the difference becomes strikingly apparent when comparing Tables 3.1 and 3.2. This efficiency of FFT proves to be crucial for this project, enabling real-time rendering.

It should be noted that in the table bellow the mesurments were done only for high end gpu as M1 din't have enough power to run the algorithm.

GPU	Texture Size	Cascade Count	Time
Nvidia GeForce RTX 4070	256x256	1	18.65ms
Nvidia GeForce RTX 4070	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 underperformance 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 chopier waves, and accurately simulates energy transfers, contributing to a more authentic representation of oceanic conditions.

Figure 3.3: Sum Of Sines Calm Ocean

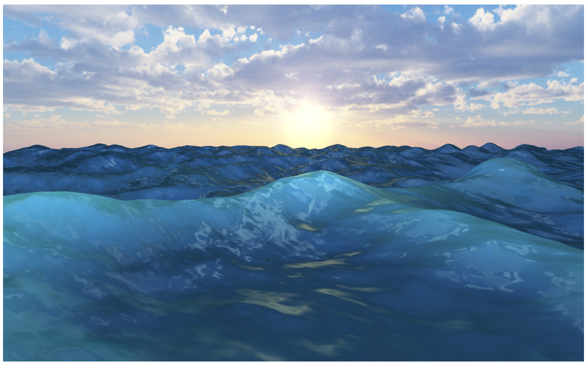

Figure 3.4: Sum Of Sines Stormy Ocean

3.2.4 Real world oceans

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

Spectrum	**Size**	$\mathbf{l_1}$	$\mathbf{l_2}$	$\mathbf{l_3}$	λ	$\mathbf{U_{10}}$	**Fetch**	**Depth**
256x256	TMA	256	100	10	0.9	0.5 m/s	1000 km	500 m

Table 3.3: Calm Ocean Parameters

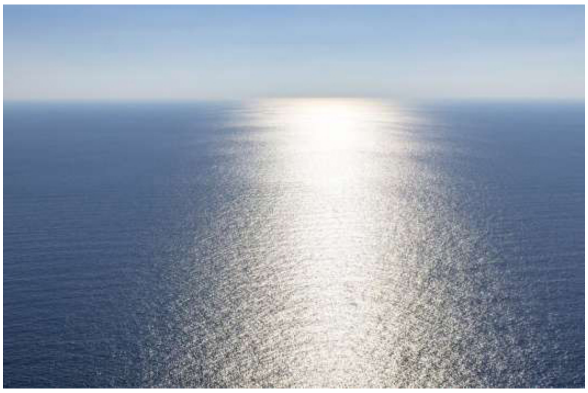

Figure 3.5: Calm Atlantic Ocean
Credits: CC0 Public Domain

Figure 3.6: FFT Calm Ocean

When it comes to stormy ocean where huge waves is expected the general shape of an ocean is still relistic and can hadle the big waves without any trouble, and because of diffrent cascades the tilling is bearlly noticible as shown in the following figures 3.7 and 3.8.

Spectrum	**Size**	$\mathbf{l_1}$	$\mathbf{l_2}$	$\mathbf{l_3}$	λ	$\mathbf{U_{10}}$	**Fetch**	**Depth**
256x256	TMA	700	256	70	0.9	21 m/s	1000 km	500 m

Table 3.4: Stormy Ocean Parameters

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

Figure 3.7: Calm Atlantic Ocean
Credits: CC0 Public Domain

Figure 3.8: Height Map using
TMA Spectrum

3.3 Diffrent Outputs

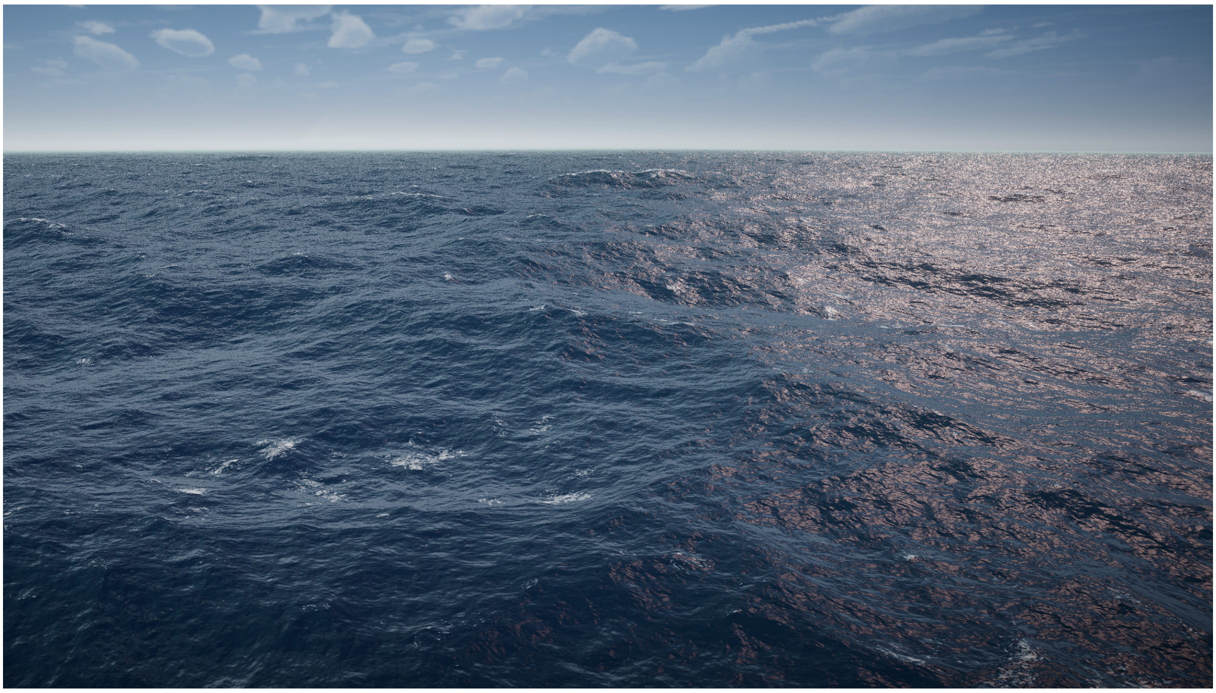

Figure 3.9: Calm ocean with blue sky

Spectrum	**Size**	$\mathbf{l_1}$	$\mathbf{l_2}$	$\mathbf{l_3}$	λ	$\mathbf{U_{10}}$	**Fetch**	**Depth**
256x256	TMA	400	100	10	2	0.5 m/s	1000 km	500 m

Table 3.5: Calm Ocean Parameters

Figure 3.10: Stormy Ocean with cloudy enviroment

Spectrum	**Size**	$\mathbf{l_1}$	$\mathbf{l_2}$	$\mathbf{l_3}$	λ	$\mathbf{U_{10}}$	**Fetch**	**Depth**
256x256	TMA	800	256	10	0.95	31 m/s	1000 km	500 m

Table 3.6: Stormy Ocean Parameters

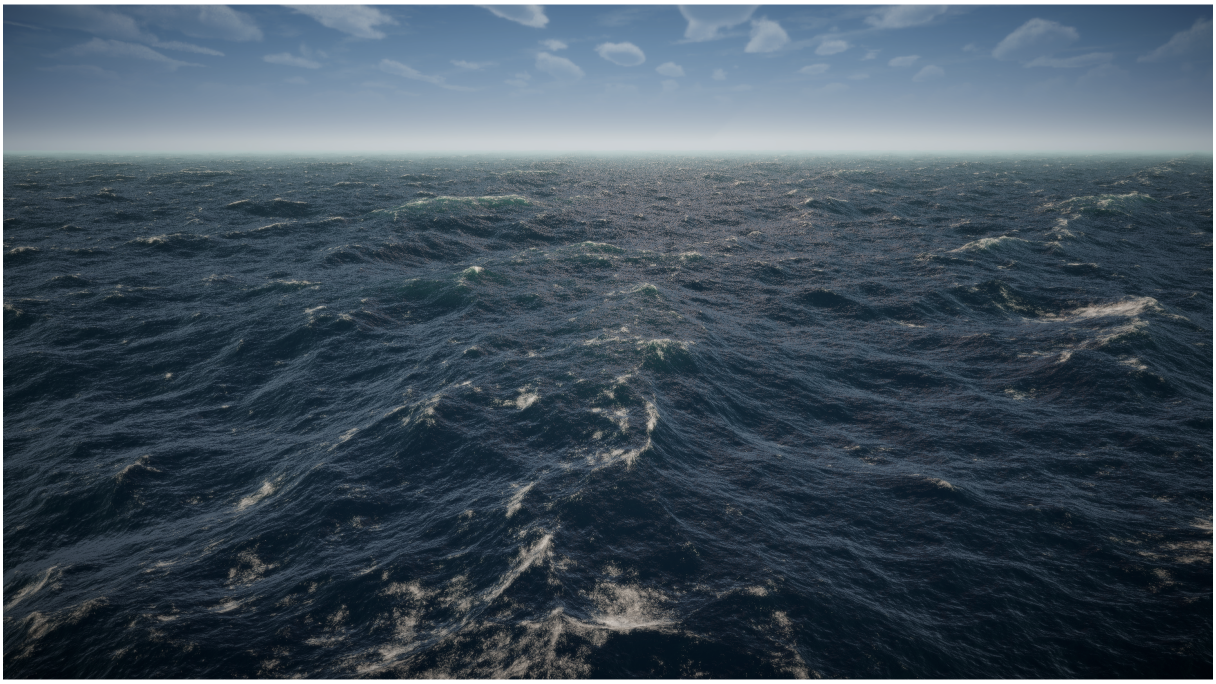

Figure 3.11: Ocean with Medium Waves

Spectrum	**Size**	$\mathbf{l_1}$	$\mathbf{l_2}$	$\mathbf{l_3}$	λ	$\mathbf{U_{10}}$	**Fetch**	**Depth**
256x256	TMA	600	256	10	1.3	12 m/s	1000 km	500 m

Table 3.7: Medium Waves Parameters

3.4 Known Problems

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 [21]. 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” [22].

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

The Fast Fourier Transform (FFT) has been instrumental in ocean rendering, providing a remarkably efficient and persuasive approximation. This efficiency is primarily attributable to the use of the JONSWAP spectrum, which is grounded in empirical data. The substantial enhancements in speed are largely credited to the FFT algorithm, without which the use of the Discrete Fourier Transform (DFT) algorithm would render real-time graphics unfeasible.

In our research, we found that the key to creating a realistic ocean lies in the selection of an empirically based spectrum, for which we employed the TMA spectrum. However, despite its efficiency, FFT-based ocean rendering is not yet sufficient to produce an ocean without tiling. To overcome this, we rendered multiple cascades for different wavelengths and superimposed them, effectively rendering the tiling inconspicuous. This approach not only removes tiling, but also enhances the quality of our ocean and reduces the computation expense as we can render multiple smaller textures.

One of the most significant challenges with FFT-based ocean rendering is its limited interactivity with surroundings. As suggested by Jerry Tessendorf in 2001[tessendorf2001], a hybrid approach could be a potential solution to this problem.

For ocean shading, we utilized a modified version of Phong shading, which yielded satisfactory results. However, to further enhance the realism of the ocean, it would be prudent to consider transitioning to Physically-Based Rendering (PBR). This transition could potentially provide a more convincing representation of the ocean.

4.2 Ideas for future work

In the pursuit of further enhancing the realism and interactivity of our ocean rendering, several advancements are proposed for future exploration.

Firstly, the current model lacks the representation of foam spray, a characteristic feature observed during significant wave crashes. To address this, we propose the integration of a GPU-based particle system, which could potentially simulate the foam spray effect, thereby adding to the visual complexity and realism of the ocean surface.

Secondly, the incorporation of buoyancy is another aspect worth considering. This would allow objects to interact with the ocean in a more realistic manner, adhering to the principles of fluid dynamics.

However, for more complex interactions, such as the ocean's reaction with the shoreline by reducing wave amplitude, or with ships by creating wakes, a hybrid approach is recommended. This approach would combine the strengths of different methods to achieve a more comprehensive and realistic simulation.

Lastly, to further enhance the visual fidelity of the ocean, we suggest transitioning from our customized Phong shading to Physically-Based Rendering (PBR). This transition could potentially provide a more convincing and physically accurate representation of the ocean surface, thereby contributing to the overall realism of the scene.

References

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

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

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

[4] 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.

[5] 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.

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

[7] 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.

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

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

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

[11] 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.

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

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

[14] Alain Fournier and William T. Reeves. A simple model of ocean waves. *SIGGRAPH Comput. Graph.*, 20(4):75–84, 1986.

[15] 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.

[16] 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.

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

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

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

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

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

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

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. However, this decision should be made with careful consideration of potential implications, including the quality assurance of the code and the readiness to handle community feedback and contributions.

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

<This appendix should provide a brief record of materials used in the solution that are not the student's own work. Such materials might be pieces of codes made available from a research group/company or from the internet, datasets prepared by external users or any preliminary materials/drafts/notes provided by a supervisor. It should be clear what was used as ready-made components and what was developed as part of the project. This appendix should be included even if no external materials were used, in which case a statement to that effect is all that is required.>

B.1 Skyboxs

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`.