School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

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. Therefore, using sum of sines or gerstner waves is not sufficient.

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 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 [1] in 1802. The initial known implementation of Gerstner waves in computer graphics was carried out by Fournier and Reeves [2] 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:

$$\mathbf{x} = \mathbf{x_0} - (\mathbf{k}/k)A\sin(\mathbf{k} \cdot \mathbf{x_0} - \omega t) \tag{1.1}$$

$$y = A\cos(\mathbf{k} \cdot \mathbf{x_0} - \omega t) \tag{1.2}$$

$$\omega = \sqrt{gk} \tag{1.3}$$

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.

$$k = 2\pi/\lambda \tag{1.4}$$

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:

$$\mathbf{x} = \mathbf{x_0} - \sum_{i=1}^{N}(\mathbf{k_i}/k_i)A_i\sin(\mathbf{k_i} \cdot \mathbf{x_0} - \omega_i t) \tag{1.5}$$

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 1.2, and underperforms in simulating stormy weather conditions.

Figure 1.2: Water Tiling in Pokemon Legends: Arceus

1.3 Particle Simulation and Machine Learning

Particle-based methods are commonly used for realistic water simulation, but they're computationally expensive. Recent advancements in Graphics Processing Unit (GPU) technology, coupled with innovative research such as that conducted by Libo Huang [3], are progressively rendering particle simulations more feasible for ocean simulation. Nonetheless, these techniques necessitate a high-performance GPU and are not yet suitable for widespread real-time applications.

Furthermore, there have been significant strides in the application of machine learning to expedite fluid simulations, as evidenced by the work of Dmitrii Kochkov [4]. These methodologies, however, are exceedingly complex, necessitating expertise in both computer graphics and machine learning. Additionally, they require substantial volumes of data for model training. Therefore, they will not be considered for this project.

1.4 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[5] 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:

$$\tilde{f}(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \tag{1.6}$$

, 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:

$$f(x) = \int_{-\infty}^{\infty} \tilde{f}(\omega) e^{i\omega x} d\omega \tag{1.7}$$

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

$$x_n = \sum_{k=0}^{N-1} \tilde{x}_k e^{-i2\pi kn/N} \tag{1.8}$$

and the Inverse Discrete Fourier Transform (IDFT):

$$\tilde{x}_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{i2\pi kn/N} \tag{1.9}$$

1.5 Fourier Transform Ocean

The challenge with Gerstner waves is that we need to generate multiple waves for each vertex to simulate an ocean. This can be computationally expensive as an ocean can have thousands of vertices. To address this problem, J. Tessendorf[6] proposed a method to generate 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:

$$\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}}(\xi_r + \xi_i)\sqrt{P_h(\mathbf{k})} \tag{1.10}$$

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"[7]:

$$\tilde{h}(\mathbf{k}, t) = \tilde{h}_0(\mathbf{k})e^{i\omega(k)t} + \tilde{h}_0^*(-\mathbf{k})e^{-i\omega(k)t} \tag{1.11}$$

By performing the Inverse Discrete Fourier Transform (IDFT):

$$h(\mathbf{x}) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t)e^{i\mathbf{k}\cdot\mathbf{x}} \tag{1.12}$$

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

Figure 1.3: Height Map using J. Tessendorf's Spectrum

1.6 Ocean Spectrums

1.6.1 Jerry Tessendorf's Ocean Spectrum

The main thing that defines how ocean will looks is the spectrum. Inside Jerry Tessendorf's [6] paper, he proposed modified Phillips spectrum 3.2:

$$P_h(\mathbf{k}) = A\frac{e^{-1/(kL)^2}}{k^4}|\hat{\mathbf{k}} \cdot \mathbf{w}|^6 \tag{1.13}$$

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 $e^{-k^2l^2}$.

1.6.2 JONSWAP Spectrum

Regarding all the issues that we experiance with "Phillips Spectrum", Horvath Christopher [7] proposed to use JONSWAP spectrum. JONSWAP stands for "Joint North Sea Wave Project" and was developed by Hasselmann [8, et al. in 1973]. This spectrum is improved version of Pierson-Moskowitz Spectrum [9] 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:

$$
\begin{aligned}
S_{\text{JONSWAP}}(\omega) &= \frac{\alpha g^2}{\omega^5} \exp\left(-\frac{5}{4}\left(\frac{\omega_p}{\omega}\right)^4\right)\gamma^r \\
r &= \exp\left(-\frac{(\omega-\omega_p)^2}{2\sigma^2\omega_p^2}\right) \\
\alpha &= 0.076\left(\frac{U_{10}^2}{Fg}\right)^{0.22} \\
\omega_p &= 22\left(\frac{g^2}{U_{10}F}\right)^{1/3} \\
\gamma &= 3.3 \\
\sigma &= \begin{cases} 0.07 & \text{if } \omega \leq \omega_p \\ 0.09 & \text{if } \omega > \omega_p \end{cases}
\end{aligned}
\tag{1.14}
$$

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.

1.6.3 The Texel MARSEN ARSLOE (TMA) Spectrum

The biggest problem with with JONSWAP spectrum is that it only works with deep waters. Therefore, Horvath Christopher [7] proposed to use TMA correction for deep water spectrum. TMA was developed by Steven A. Hughes [10], based on the observations made by Kitaigordskii [11]. this is known as Kitaigordskii depth Attenuation function and it takes form:

$$
\begin{aligned}
\Phi(\omega_h) &= \left[\frac{(k(\omega,k))^{-3}\frac{\partial k(\omega,h)}{\partial\omega}}{(k(\omega,\infty))^{-3}\frac{\partial k(\omega,\infty)}{\partial\omega}}\right] \\
\omega_h &= \omega\sqrt{h/g}
\end{aligned}
\tag{1.15}
$$

where h is the depth of the water. At first glance this function seems to be difficult, however looking at 1.4 we can see that this can be easilly approximated.

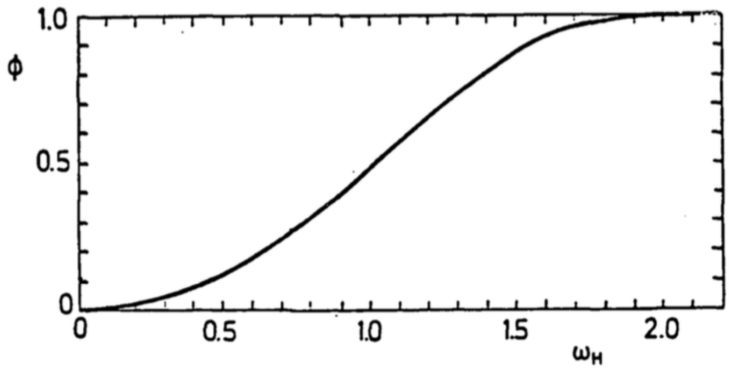

Figure 1.4: $\Phi(\omega, h)$ as a function of ω_h [10]

Approximation given by Thomson and Vincent [12] is:

$$\Phi(\omega, h) \approx \begin{cases} \frac{1}{2}\omega_h^2 & \text{if } \omega_h \leq 1 \\ 1 - \frac{1}{2}(2 - \omega_h)^2 & \text{if } \omega_h > 1 \end{cases} \tag{1.16}$$

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

$$S_{\text{TMA}}(\omega, h) = S_{\text{JONSWAP}}(\omega)\Phi(\omega, h) \tag{1.17}$$

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 [6] paper, we need to use $\mathbf{k}$ as input. To solve this problem Horvath Christopher [7] proposes to use Donelan-Banner Directional Spreading [13]:

$$D(\omega, \theta) = \frac{\beta_s}{2\tanh(\beta_s \pi)}\text{sech}(\beta_s \theta)^2 \tag{1.18}$$

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:

$$D_{\text{TMA}}(\omega, \theta) = S_{\text{TMA}}(\omega) \cdot D(\omega, \theta) \tag{1.19}$$

TMA transformation

Lastlly, to make this spectrum usable we need to transform it from ω to $\mathbf{k}$ and according to Horvath Christopher [7] we can transform it like this:

$$S_{\text{TMA}}(\mathbf{k}) = 2S_{\text{TMA}}(\omega, h) \cdot \frac{d\omega}{dk}/k \cdot \Delta k_x \cdot \Delta k_y \tag{1.20}$$

where in our case,

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

1.7 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 1.9 has time complexity of $O(n^2)$, and it's clearlly that this algorithm is not sufficient to simulate heiger resolution ocean surfaces in real time. In 1805 Gauss Carl Friedrich [14] invented FFT algorithm $O(n \log n)$ and in 1965 Cooley James W. and Tukey John W. [15] 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

$$k = i * N/2^{\text{stage}+1} \bmod N \tag{1.21}$$

   where i is the index of the sample, stage is the current stage and N is the number of samples.

3. Perform butterfly operations 1.5, where x and y are complex numbers and W is the twiddle factor

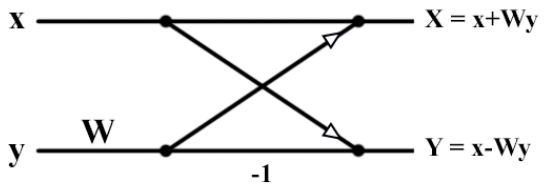

Figure 1.5: Butterfly Diagram

4. The butterfly operations are performed in stages 1.6. 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.

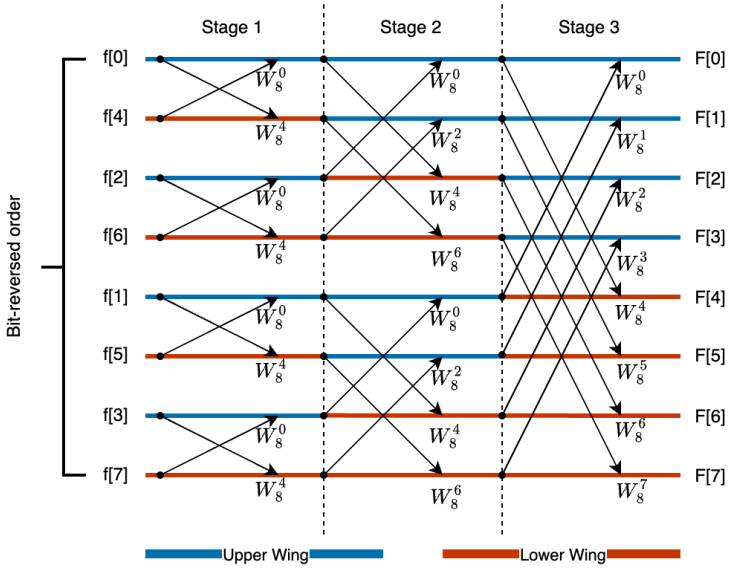

Figure 1.6: 8-point FFT Butterfly Diagram

Chapter 2

Methods

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 IFFT

3.1.1 Butterfly Texture 3.1

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 [16, 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 1.5.

In butterfly texture coordinate x represents a stage 1.6, 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

$$\begin{aligned} y_t &= y_{\text{current}} \\ y_b &= y_{\text{current}} + 2^{\text{stage}} \end{aligned} \tag{3.1}$$

- If the butterfly wing is bottom half

$$\begin{aligned} y_t &= y_{\text{current}} - 2^{\text{stage}} \\ y_b &= y_{\text{current}} \end{aligned} \tag{3.2}$$

We can determine if the wing is upper or lower half:

$$\text{wing} = y_{\text{current}} \bmod 2^{(\text{stage}+1)} \tag{3.3}$$

Lastlly we need to calculate twiddle factors:

$$\begin{aligned} k &= (y_{\text{current}} \cdot n/2^{\text{stage}+1}) \bmod n \\ W &= \exp(-2\pi i k/n) \end{aligned} \tag{3.4}$$

3.1.2 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" 3.1. By following pseudocode from [16] we can perform IFFT in 2D:

```
float4 butterflyData = ButterflyTexture[float2(Stage, id.x)];
const float2 twiddle = butterflyData.xy;
// fetch top butterfly input sample
topSignal = PingPong0[float2(butterflyData.z, id.y)].xy;
// fetch bottom butterfly input sample
bottomSignal = PingPong0[float2(butterflyData.w, id.y)].xy;
// perform butterfly operation
h = topSignal + ComplexMult(twiddle, bottomSignal);
```

Listing 3.1: 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:

```
float4 butterflyData = ButterflyTexture[float2(Stage, id.y)];
const float2 twiddle = butterflyData.xy;
// fetch top butterfly input sample
topSignal = PingPong0[float2(id.x, butterflyData.z)].xy;
// fetch bottom butterfly input sample
bottomSignal = PingPong0[float2(id.x, butterflyData.w)].xy;
// perform butterfly operation
h = topSignal + ComplexMult(twiddle, bottomSignal);
```

Listing 3.2: Vertical Butterfly Operation

Permutation

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

$$[\text{freq}(-N/2),\ ...,\ \text{freq}(-1),\ \text{freq}(0),\ \text{freq}(1),\ ...,\ \text{freq}(N/2-1)] \tag{3.5}$$

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

$$[\text{freq}(0),\ \text{freq}(1),\ ...,\ \text{freq}(N-1)] \tag{3.6}$$

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

```
float perms[] = {-1, 1};
uint index = int((id.x + id.y) % 2);
float perm = perms[index];
float h = perm * PingPong1[id.xy].x;
```

```
    PingPong0[id.xy] = float4(h, h, h, 1);
```

Listing 3.3: Data Permutation [16]

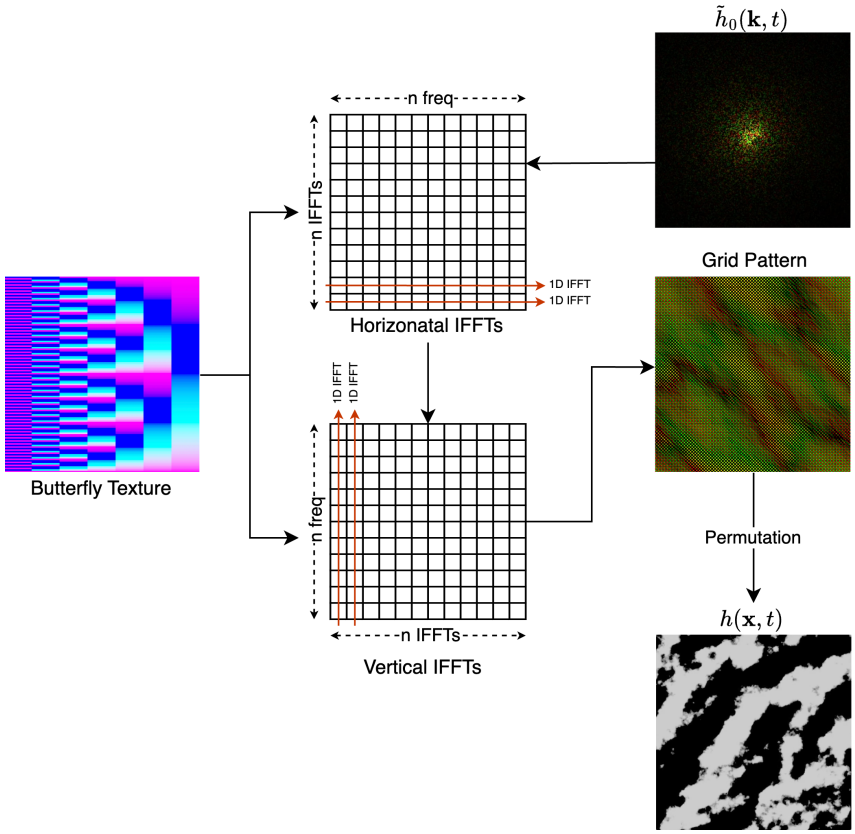

Figure 3.1: IFFT Algorithm

3.2 Ocean Geometry

3.2.1 Spectrum Generation

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 1.13 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.

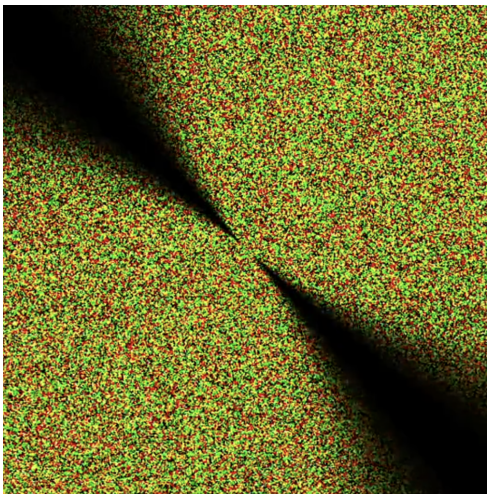

Figure 3.2: Jerry Tessendorf's Spectrum

Therefore, TMA spectrum 1.20 was implemented. 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.

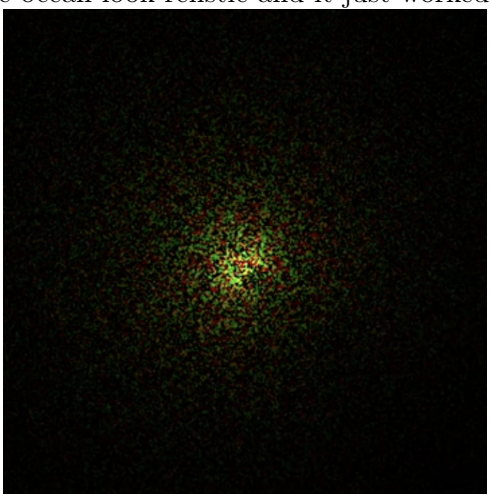

Figure 3.3: TMA spectrum, Frequency Domain
(100x for better visibility)

3.2.2 Height Map Generation

To generate height map, firstlly we need to have fourier amplitudes as shown in 1.10, 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"[7]. In our luck we don't need to recalculate complex conjugate as fourier series are symetric, so we can just mirror the amplitudes:

$$\tilde{h}_0^* = T_{h_0}(x^*, y^*) \tag{3.7}$$

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 1.11 we can perform IFFT as shown in 3.1 to produce height map.

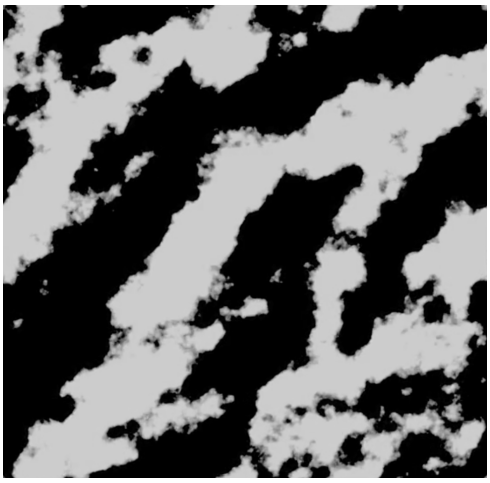

Figure 3.4: Height Map using S_{TMA}

3.2.3 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 [17] the derivative is:

$$\epsilon(\mathbf{x}, t) = i\mathbf{k}\tilde{h}(\mathbf{k}, t) \tag{3.8}$$

Then we need to perform IFFT 3.1 to get normal map in the time domain.

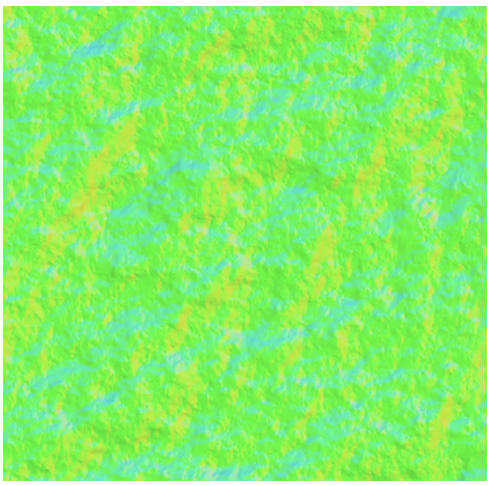

Figure 3.5: Normal Map using S_{TMA}

3.2.4 Choppy Waves

Currentlly, when rendering ocean it looks too smooth 3.6 and lacks choppiness.

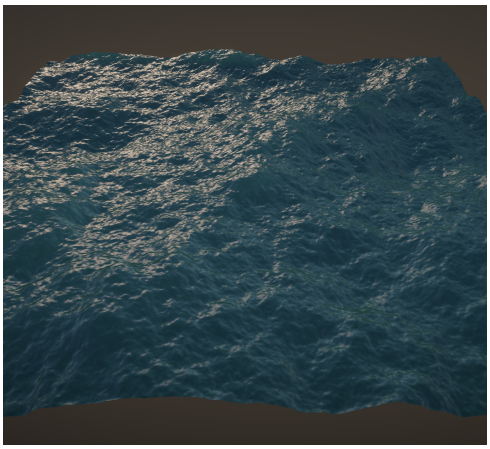

Figure 3.6: Ocean without Choppy Waves

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 [17, J. Tessendorf] we can calculate horizontal displacement:

$$\mathbf{D}_{\text{hori}}(\mathbf{x}, t) = -i\frac{\mathbf{k}}{k}h(\tilde{\mathbf{k}}, t) \tag{3.9}$$

Using this horizotal displacement we can displace our verticies:

$$\mathbf{x} + \lambda\mathbf{D}_{\text{hori}}(\mathbf{x}, t) \tag{3.10}$$

, where λ is "choppy factor". Once again we need to perform IFFT 3.1 to get horizontal displacement in the time domain.

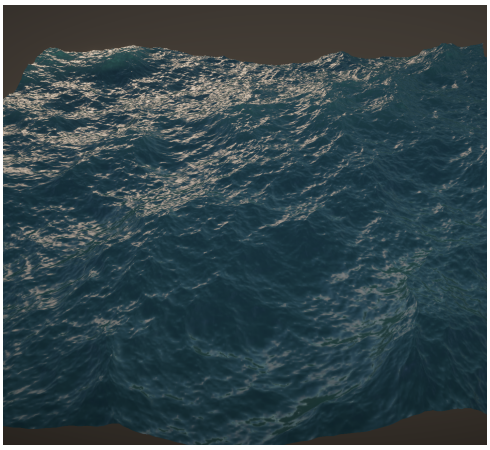

Figure 3.7: Ocean with Choppy Waves

3.3 Foam

3.3.1 Foam Generation

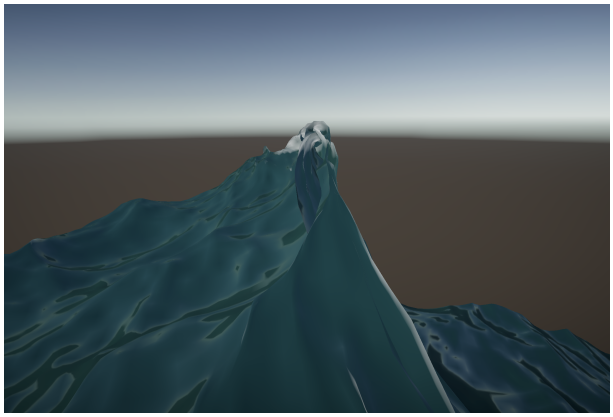

Figure 3.8: Wave Curl At Wave Peek

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

As proposed by Tessendorf [6], 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:

$$\mathrm{Det}(\mathbf{x}) = J_{xx} + J_{yy} - J_{xy}J_{yx} \tag{3.11}$$

where,

$$J(\mathbf{x}) = \begin{bmatrix} 1 + \lambda\frac{\partial D_x(\mathbf{x})}{\partial x} & 1 + \lambda\frac{\partial D_x(\mathbf{x})}{\partial y} \\ 1 + \lambda\frac{\partial D_y(\mathbf{x})}{\partial x} & 1 + \lambda\frac{\partial D_y(\mathbf{x})}{\partial y} \end{bmatrix} \tag{3.12}$$

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

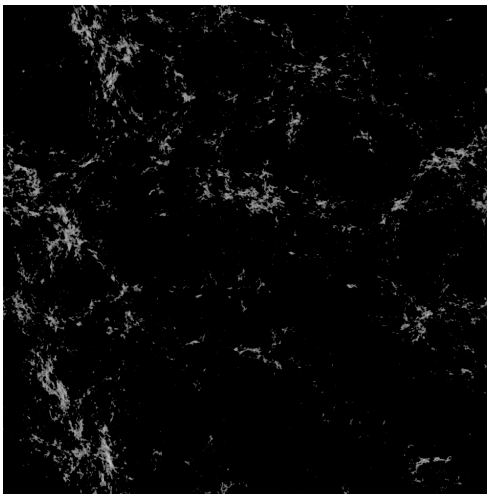

Figure 3.9: Foam Texture

3.3.2 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 3.4: Foam Accumilation

This results in pleasing foam accumilation:

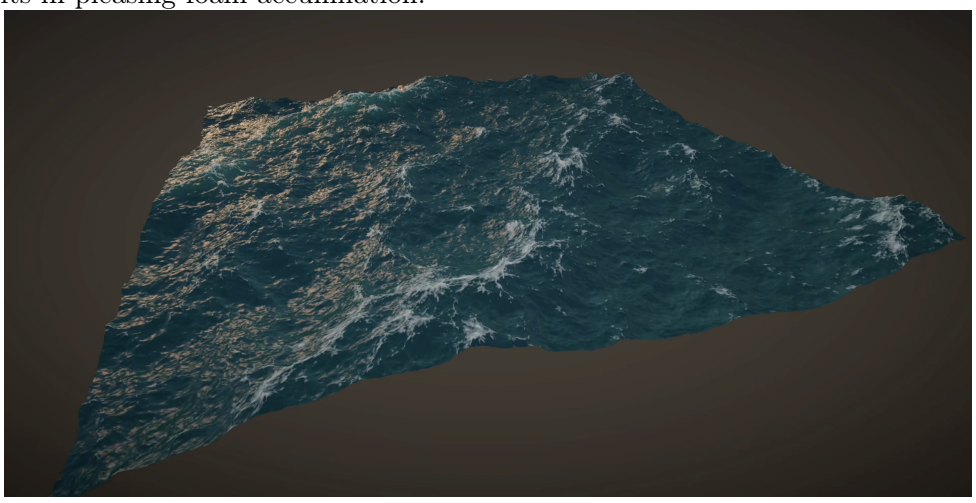

Figure 3.10: Ocean With Foam

3.4 Multiple Cascades

At this stage our ocean with texture 512x512 simulates 262,144 distinc waves however the tilling is still noticible 3.11.

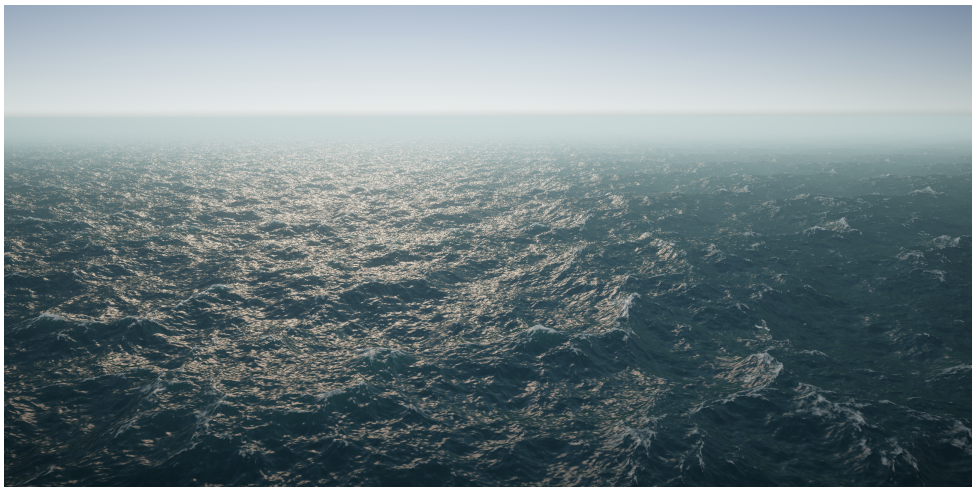

Figure 3.11: Ocean with Tilling

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 3.12. We can split our simulation into 3 parts, big waves, medium waves and small waves.

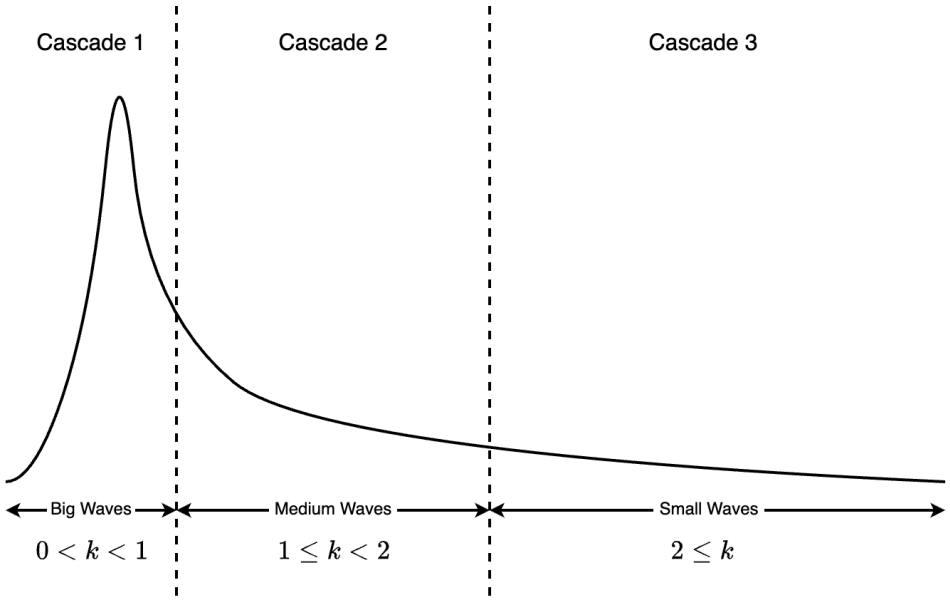

Figure 3.12: Cascades

This results ocean that has less tilling and more detail 3.13.

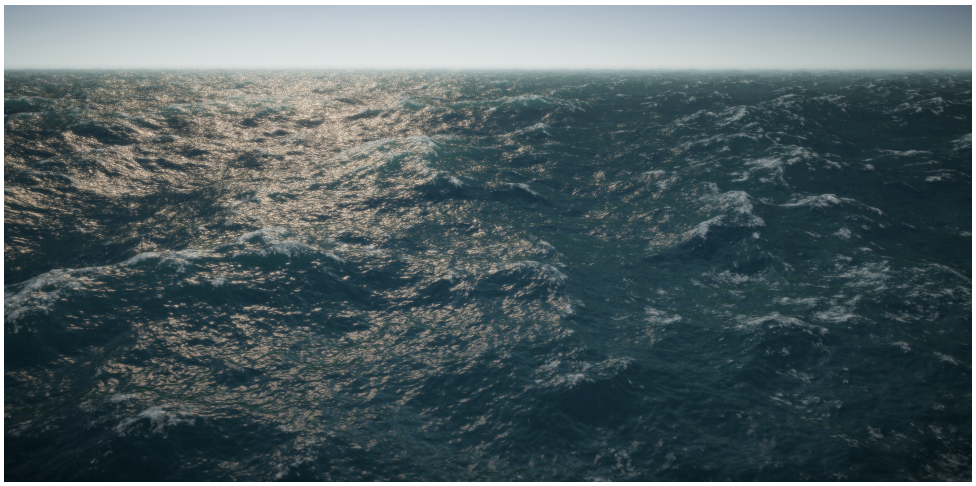

Figure 3.13: Ocean with Cascades

3.5 Performance

3.6 Compereson

Chapter 4

Discussion

<Everything that comes under the 'Results and Discussion' criterion in the mark scheme that has not been addressed in an earlier chapter should be included in this final chapter. The following section headings are suggestions only.>

4.1 Conclusions

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

4.2 Ideas for future work

This technique produces realistic oceans in a cost-efficient way suitable for real-time rendering. However, it does not support interactive waves. For that, J. Tessendorf proposed a method called "iWaves" [17] and a later upgraded version "eWaves" [18].

References

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

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

[3] 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.

[4] 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.

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

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

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

[8] 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.

[9] 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.

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

[11] 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.

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

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

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

[15] 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.

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

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

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

Appendix A

Self-appraisal

<This appendix should contain everything covered by the 'self-appraisal' criterion in the mark scheme. Although there is no length limit for this section, 2—4 pages will normally be sufficient. The format of this section is not prescribed, but you may like to organise your discussion into the following sections and subsections.>

A.1 Critical self-evaluation

A.2 Personal reflection and lessons learned

A.3 Legal, social, ethical and professional issues

<Refer to each of these issues in turn. If one or more is not relevant to your project, you should still explain *why* you think it was not relevant.>

A.3.1 Legal issues

A.3.2 Social issues

A.3.3 Ethical issues

A.3.4 Professional issues

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.>