Ocean Wave Simulation (FFT)

Members:

Srivatsan (220121067)

S.Dheeraj (220102099)

Shreyansh (220121062)

Srijan (220121066)

Overview

Generating Realistic Ocean Waves through Fast Fourier Transform

1. Introduction:

The project aims to employ the Fast Fourier Transform (FFT) to create accurate and realistic ocean wave simulations. This initiative is crucial for various sectors such as gaming, animation, virtual reality, and oceanographic research, where authentic wave behavior is essential. The complexity of ocean wave dynamics poses a challenge in creating lifelike simulations, necessitating the use of advanced mathematical models such as FFT.

2. Objective:

This project's primary objective is to leverage FFT to develop a mathematical model capable of generating realistic ocean wave patterns. The FFT will facilitate the synthesis of waves based on their frequency components, providing a method to simulate the intricate behavior of ocean waves.

3. Methodology:

The project will delve into the principles of the Fast Fourier Transform, focusing on its ability to break down time-domain ocean wave data into its constituent frequency components. The process involves initializing or defining wave parameters, applying FFT to transform the data into the frequency domain, and reconstructing wave patterns by manipulating these frequency components. The implementation will likely utilize

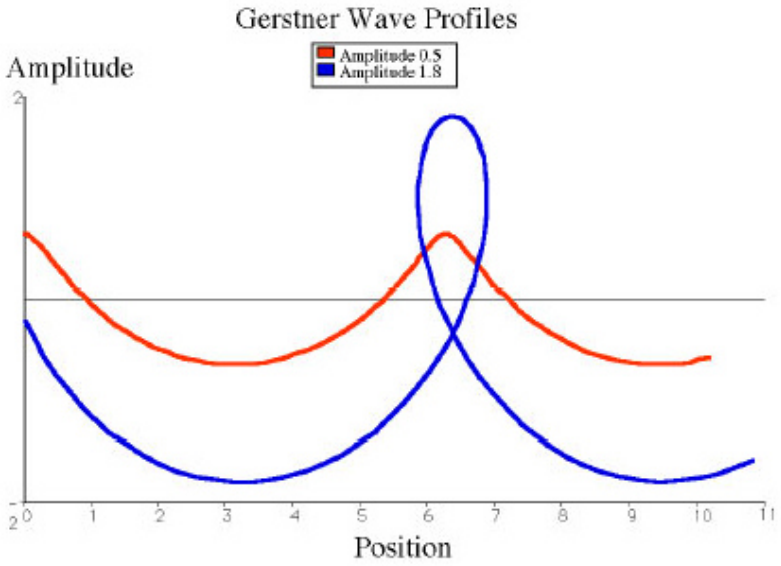
algorithms and programming languages tailored for efficient FFT operations and wave simulation.

4. Implementation:

We have implemented the waves by using trochoidal waves using Matlab. We have also tweaked various parameters of the wave to make it more realistic,for instance adding gravity,wind and adding more details on the wave.

5. Results and Discussion:

We have generated the ocean wave patterns, comparing them against real-world data or existing models.



Gerstner Waves :

Sine waves are simple, but they do not match the shape of real water waves. Big wind waves are realistically modeled by the Stokes wave function, but it's rather complex. Instead, Gernster waves are often used for real time animation of water surfaces. Gerstner waves are named after František Josef Gerstner, who discovered them. They're also known as trochoidal waves, named after their shape, or periodic surface gravity waves, which describes their physical nature.

The parametric equations for Gerstner waves are:

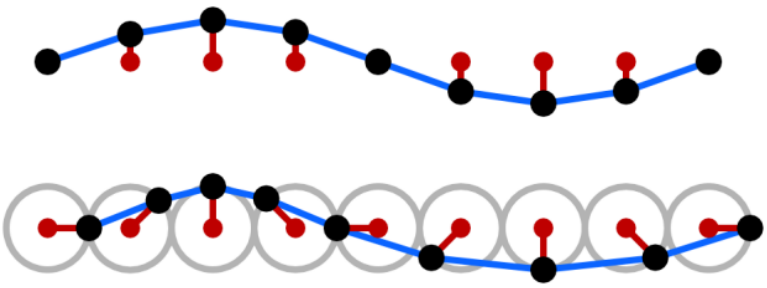
$x(t) = A * \cos(k * x - \omega * t)$

$y(t) = A * \sin(k * x - \omega * t)$

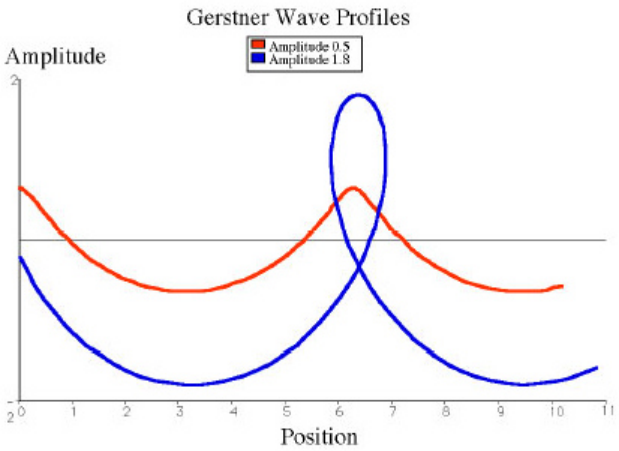
$z(t) = \zeta * \cos(k * x - \omega * t)$

Moving Back and Forth :

The fundamental observation is that while waves move across a water surface, the water itself doesn't move along with it. In the case of a sine wave, each surface point goes up and down, but doesn't move horizontally. But actual water isn't just the surface. There is more water underneath. When the surface water moves down, where does the water below it go? When the surface moves up, what fills the space below it? It turns out that the surface points not only move up and down, they move forward and backward too. Half the time they move along with the wave, but the other half they move in the opposite direction. The same is true for the water below the surface, but the deeper you go the less movement there is. Specifically, each surface point moves in a circle, orbiting a fixed anchor point. As the crest of a wave approaches, the point moves toward it. After the crest passes, it slides back, and then the next crest comes along. The result is that water bunches up in crests and spreads out in troughs, and the same will happen to our vertices.



Sine vs. Gerstner wave.



2D Trochoidal Wave :

We begin by attempting to plot a 2D Trochoidal wave on Matlab to understand its characteristic behavior.

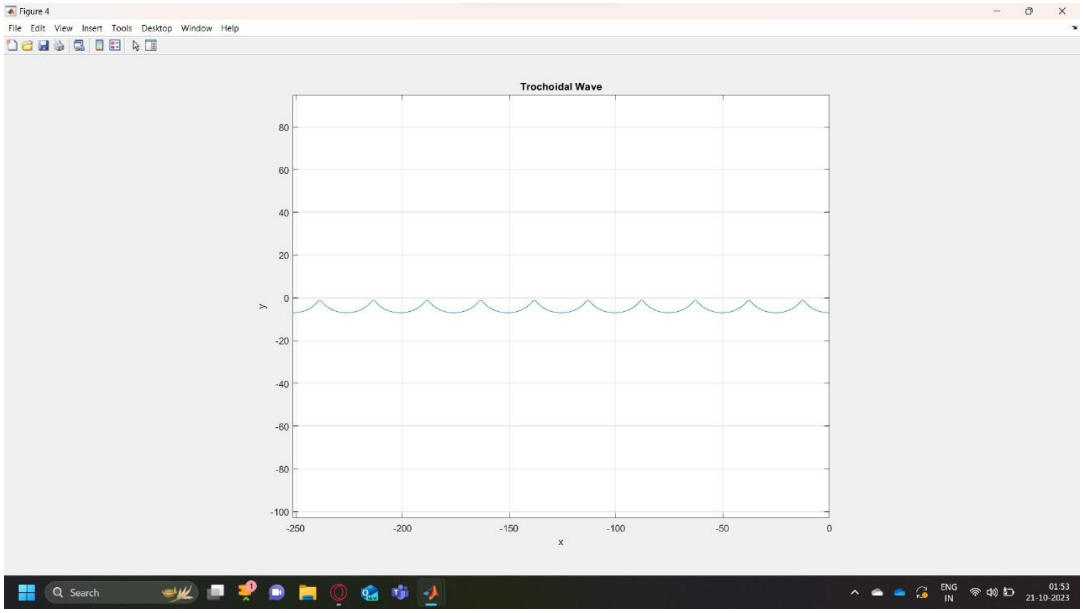
This is the respective Matlab code snippet.

Editor - D:\MATLAB\Ocean_Simulator\Ocean.m *

Ocean.m *

```
1   R = 4;  % Radius of the larger circle
2   r = 1;  % Radius of the smaller circle
3   t = 0:0.01:20*pi;  % Parameter values
4
5   % Trochoidal equations
6   x = R * t - r * sin(t);
7   y = R - r * cos(t);
8
9   % Plot
10  figure;
11  plot(x, y);
12  title('Trochoidal Wave');
13  xlabel('x');
14  ylabel('y');
15  axis equal;
16  grid on;
17
```

The Plot looks something like this:



Preventing Loops :

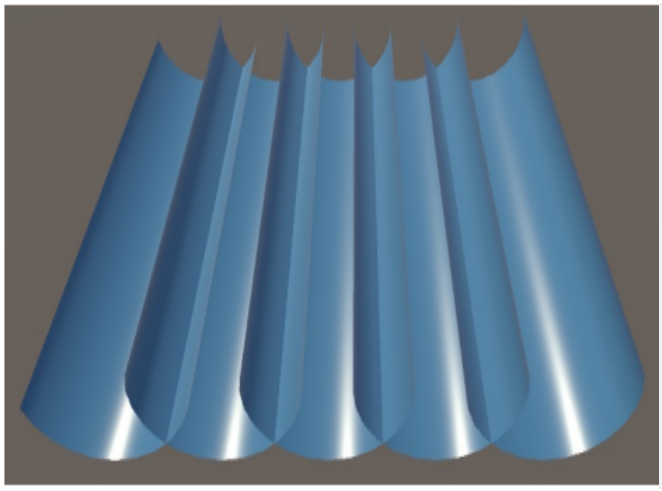
While the resulting waves might look fine, this isn't always the case. For example, reducing **R** to -4 while keeping **r** at 9 produces weird results. Wave loops, wavelength 20. **T x ′ = 1 T = [1 − ka sin f ka cos f]**

Because the amplitude is so large relative to the wavelength, the orbits of the surface points overshoot and form loops above the surface. If this was real water, then the waves would break and fall apart, but we cannot represent that with Gerstner waves.

We can see why this happens mathematically, by observing that can become negative when is larger than 1. When that happens, the tangent vector ends up pointing backward instead of forward. And when is 1 then we end up with a tangent vector that points straight up.

In reality, we don't get intact waves where the angle between both sides of the crests exceed 120°. Gerstner waves don't have this limit, but we don't want to go below 0°, because that's when we get surface loops.

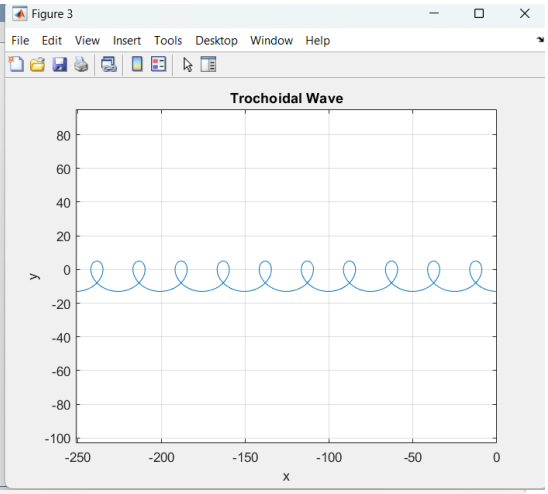
To get sharper waves, **|r|<|R|,** where **r** is as close as possible to **|R|.**



Editor - D:\MATLAB\Ocean_Simulator\Ocean.m

Ocean.m

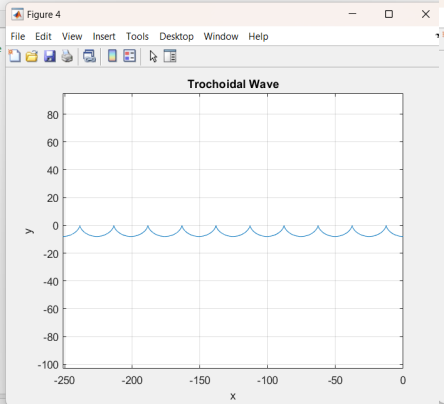
```
R = -4;  % Radius of the larger circle
r = 9;  % Radius of the smaller circle
t = 0:0.01:20*pi;  % Parameter values

% Trochoidal equations
x = R * t - r * sin(t);
y = R - r * cos(t);

% Plot
figure;
plot(x, y);
title('Trochoidal Wave');
xlabel('x');
ylabel('y');
axis equal;
grid on;
```

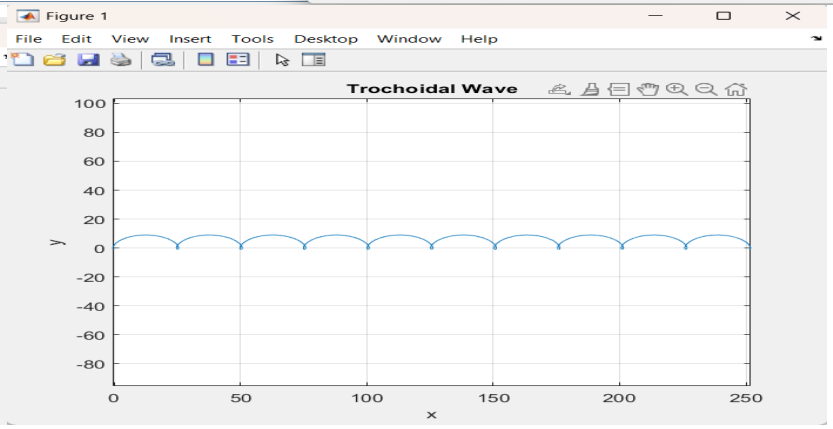


Editor - D:\MATLAB\Ocean_Simulator\Ocean.m

Ocean.m

```
R = -4;  % Radius of the larger circle
r = 3.999;  % Radius of the smaller circle
t = 0:0.01:20*pi;  % Parameter values

% Trochoidal equations
x = R * t - r * sin(t);
y = R - r * cos(t);

% Plot
figure;
plot(x, y);
title('Trochoidal Wave');
xlabel('x');
ylabel('y');
axis equal;
grid on;
```





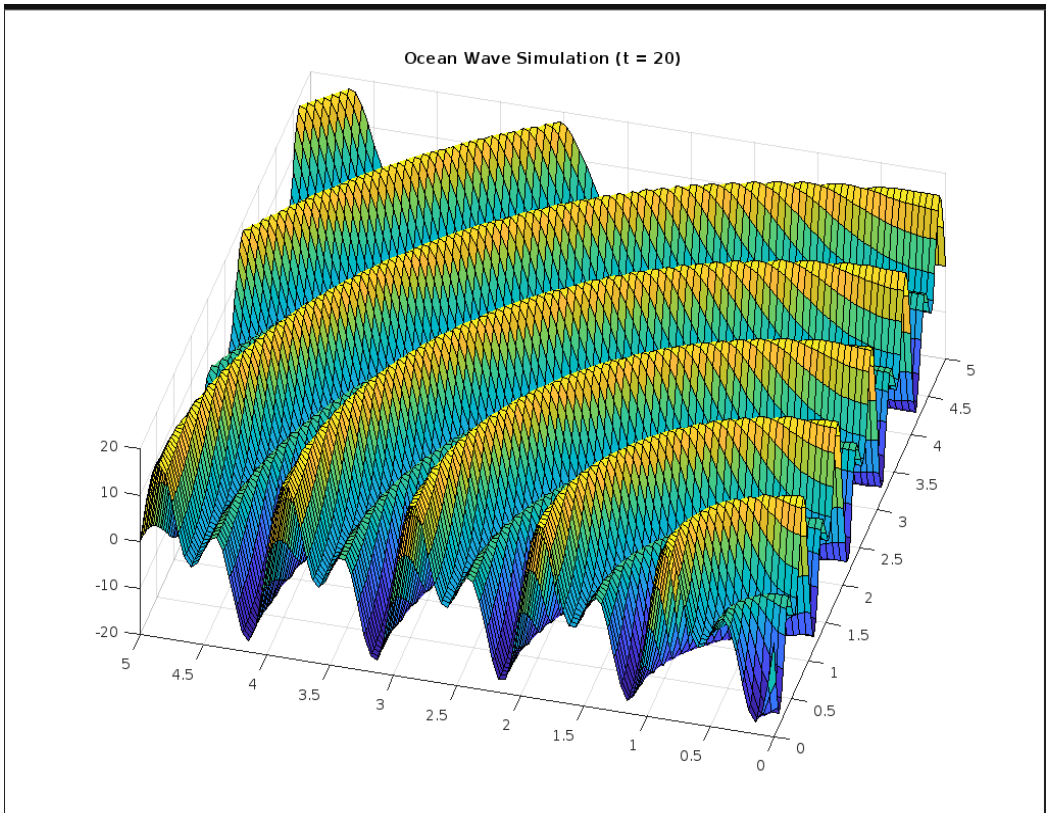
```
Lx = 5;     % Length of the domain in the x-direction
Ly = 5;     % Length of the domain in the y-direction
Nx = 100;    % Number of grid points in the x-direction
Ny = 100;    % Number of grid points in the y-direction
tmax = 20;   % Total simulation time
c = 0.3;     % Phase speed

% Create grid
x = linspace(0, Lx, Nx);
y = linspace(0, Ly, Ny);
[X, Y] = meshgrid(x, y);

% Initialize time
t = 0;

% Generate ocean wave height data
H = 10 * sin(2*pi*(c*t - sqrt(X.^2 + Y.^2))) + 10 * sin(4*pi*(c*t - sqrt(X.^2 + Y.^2)));

% Create a surface plot of the ocean wave
figure;
surf(x, y, H);
title('Ocean Wave Simulation');
xlabel('X');
ylabel('Y');
zlabel('Wave Height');
axis([0, Lx, 0, Ly, -0.4, 0.4]);
shading interp;

% Animation loop
for t = 0:0.1:tmax
    H = 10* sin(2*pi*(c*t - sqrt(X.^2 + Y.^2))) + 10 * sin(4*pi*(c*t - sqrt(X.^2 + Y.^2)));
    surf(x, y, H);
    title(['Ocean Wave Simulation (t = ', num2str(t), ')']);
    drawnow;
end
```

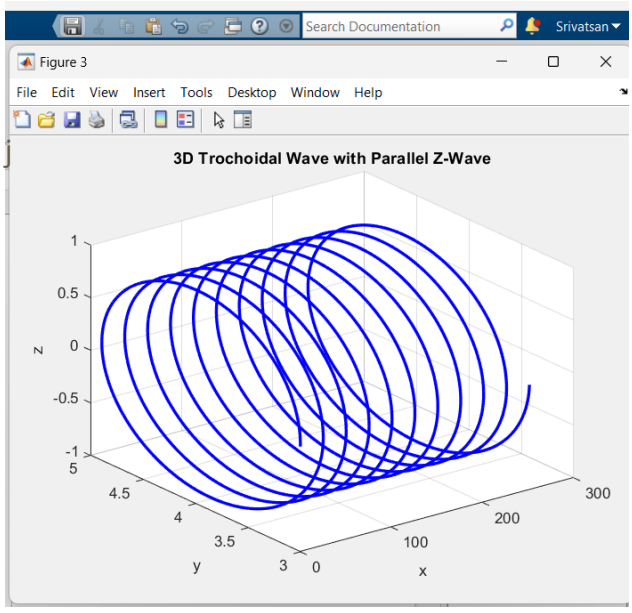
This is a matlab code for generating an animated ripple wave.

We took the model for reference and rewrote the equations to achieve a gerstner wave but the result was like a combined sinusoidal plot.

A snip of animation is attached below :

We have received the respective results, we were not able to make a plot of a 3D trochoidal wave like imagined but were able to achieve a form of combination of sine waves to achieve this.





A Direct attempt to have a 3D view of trochoidal wave looked something like this.

Cooley Tukey FFT Algorithm :

The Cooley-Tukey FFT-algorithm for computing the Discrete Fourier Transformation (DFT). The DFT of a discrete signal f with N samples :

$$\mathbf{f} = (\mathbf{f}[0], \mathbf{f}[1], ..., \mathbf{f}[N-1])$$

is defined by the N-tuple :

$$\mathbf{F} = (\mathbf{F}[0], \mathbf{F}[1], ..., \mathbf{F}[N-1]) \ ,$$

Where,

$$\mathbf{F}[k] = \sum_{n=0}^{N-1} \mathbf{f}[n] e^{\frac{-2\pi i k n}{N}} \quad , n = 0, 1, ..., N-1 \ .$$

Computing F of a discrete signal f with N samples in the naive way would require N complex multiplications and N − 1 complex additions for each element of F, which total time is proportional to O(n 2). The FFT algorithm reduces the complexity to O(n log n). For large N this leads to a massive performance advantage. One limitation of the FFT is, that the DFT must be of even order and for most efficiency N must be a power of 2. An option to execute FFT's on signal data without N2 samples is by adding zeros until the closest power of 2 is reached. This technique is known as "Zero Padding".

Four-Point DFT :

The basic concept of the FFT is to exploit the structure of the DFT by smartly rearranging the products. Before the general procedure of the FFT algorithm will be presented in chapter 2.2, the computation of the DFT with N = 4 samples is demonstrated.

Since,

$$e^{-i\pi/2} = -i \ ,$$

F[k] can be simplified for N = 4 to,

$$\mathbf{F}[k] = \sum_{n=0}^{3} = (-i)^{kn} \mathbf{f}[n] \ ,$$

which leads to

Hence, F^T is giv

$$\mathbf{F}[k] = \mathbf{f}[0] + (-i)^k\,\mathbf{f}[1] + (-1)^k\,\mathbf{f}[2] + i^k\,\mathbf{f}[3]\ .$$

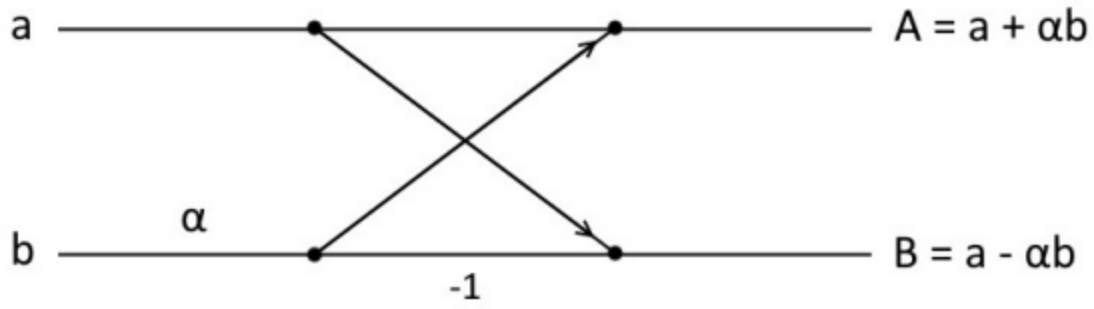
$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \mathbf{f}^\top = \begin{bmatrix} \mathbf{f}[0] + \mathbf{f}[1] + \mathbf{f}[2] + \mathbf{f}[3] \\ \mathbf{f}[0] - i\mathbf{f}[1] - \mathbf{f}[2] + i\mathbf{f}[3] \\ \mathbf{f}[0] - \mathbf{f}[1] + \mathbf{f}[2] - \mathbf{f}[3] \\ \mathbf{f}[0] + i\mathbf{f}[1] - \mathbf{f}[2] - i\mathbf{f}[3] \end{bmatrix}$$

After some rearranging and reduction the summands of F[k] to

$$\begin{bmatrix} \mathbf{f}[0] + \mathbf{f}[2] + \mathbf{f}[1] + \mathbf{f}[3] \\ \mathbf{f}[0] - \mathbf{f}[2] - i\mathbf{f}[1] + i\mathbf{f}[3] \\ \mathbf{f}[0] + \mathbf{f}[2] - \mathbf{f}[1] - \mathbf{f}[3] \\ \mathbf{f}[0] - \mathbf{f}[2] + i\mathbf{f}[1] - i\mathbf{f}[3] \end{bmatrix} = \begin{bmatrix} (\mathbf{f}[0] + \mathbf{f}[2]) + (\mathbf{f}[1] + \mathbf{f}[3]) \\ (\mathbf{f}[0] - \mathbf{f}[2]) - i(\mathbf{f}[1] - \mathbf{f}[3]) \\ (\mathbf{f}[0] + \mathbf{f}[2]) - (\mathbf{f}[1] + \mathbf{f}[3]) \\ (\mathbf{f}[0] - \mathbf{f}[2]) + i(\mathbf{f}[1] - \mathbf{f}[3]) \end{bmatrix}$$

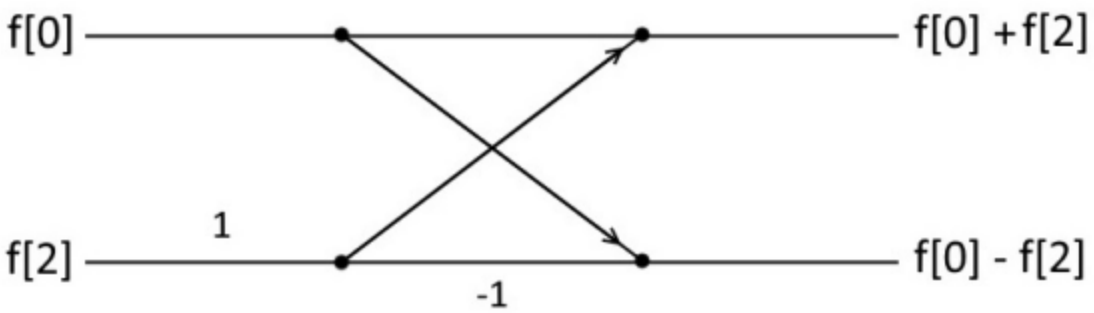
The FFT exploits the possibility to save operations by rearranging the components and reusing intermediate results of the DFT. The next section presents the execution of the FFT for N=4.

A two-point DFT consisting of two Complex multiplies and adds can be sketched with the following two-point butterfly diagram:

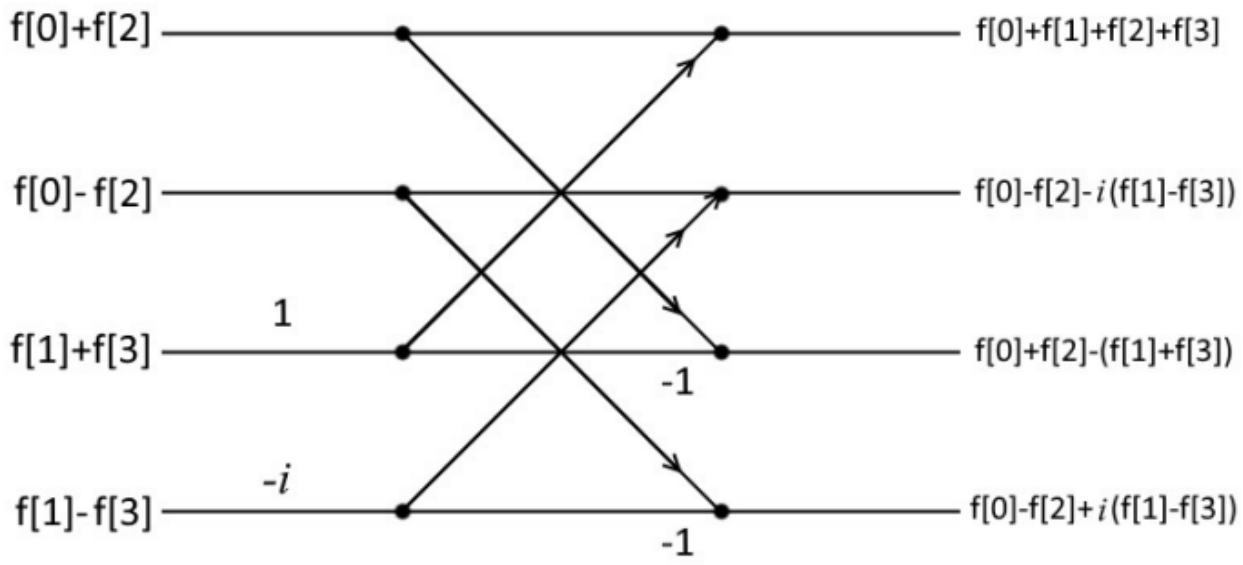


A two-point butterfly diagram consists of a single butterfly operation. A butterfly operation takes two complex numbers a and b as input and a fixed complex number a and outputs the complex values A and B.

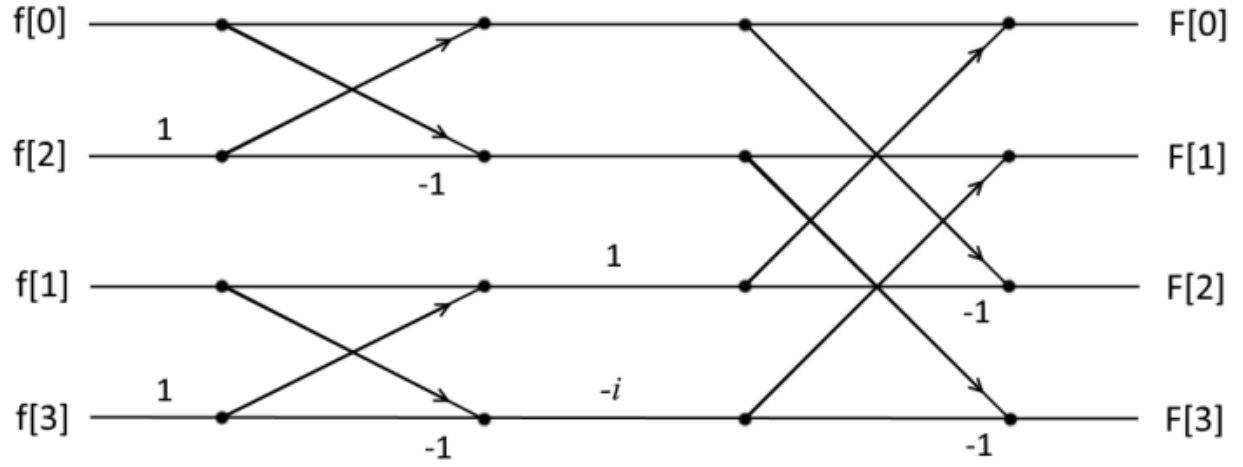
Therefore, the subexpressions f[0] + f[2] and f[0] − f[2] can be depicted as a butterfly operation as follows:



With divide and conquer the output of the previous two butterfly operations can be used for two further butterfly operations to obtain F ^T:



Solving the DFT for N=4 with butterfly operations reduces the computation cost to O(n log n), since eight complex multiplies and adds are executed (two for each sample f[k]).



From 4-Point to N-Point FFT :

The strategy of the FFT is that each component of the DFT is not computed separately, which would result in unnecessary repetitions of a (relative to N) large number of computations. Instead, the FFT does it's computation in stages. Each stage takes N complex numbers from the previous stage's output (resp. f in the first stage) and executes N/2 butterfly operations. The output of these butterfly operations are N complex numbers as input for the next stage (resp. F in the last stage). Since one stage involves N/2 butterfly operations with two complex additions, two complex multiplications and one sign inversion (resp. multiplication by -1) for each butterfly operation, one stage can be executed in O(n) time.

FFT's can be well illustrated with butterfly diagrams. It is obvious that the four-point FFT consists of two stages (log2 N for N = 4) with 2 butterfly operations for each each stage (N/2 for N = 4). For a general formulation of the FFT the simplification **e −iπ/2 = −i** is not helpful. Therefore, a new notation is introduced, the twiddle factor,

$$W_N = e^{-i\frac{2\pi}{N}}$$

The matrix representation of the four-point DFT by using the twiddle factor is given by

$$\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \mathbf{f}^\top$$

and with rearrangement of the components of f^T,

$$\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^2 & W_4^1 & W_4^3 \\ W_4^0 & W_4^4 & W_4^2 & W_4^6 \\ W_4^0 & W_4^6 & W_4^3 & W_4^9 \end{bmatrix} \begin{bmatrix} \mathbf{f}[0] \\ \mathbf{f}[2] \\ \mathbf{f}[1] \\ \mathbf{f}[3] \end{bmatrix}$$

Due to the Nth roots of unity of Wk N for k=0,...,N-1 the matrix representation can be simpliefied to

$$\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^2 & W_4^1 & W_4^3 \\ W_4^0 & W_4^0 & W_4^2 & W_4^2 \\ W_4^0 & W_4^2 & W_4^3 & W_4^1 \end{bmatrix} \begin{bmatrix} \mathbf{f}[0] \\ \mathbf{f}[2] \\ \mathbf{f}[1] \\ \mathbf{f}[3] \end{bmatrix}$$

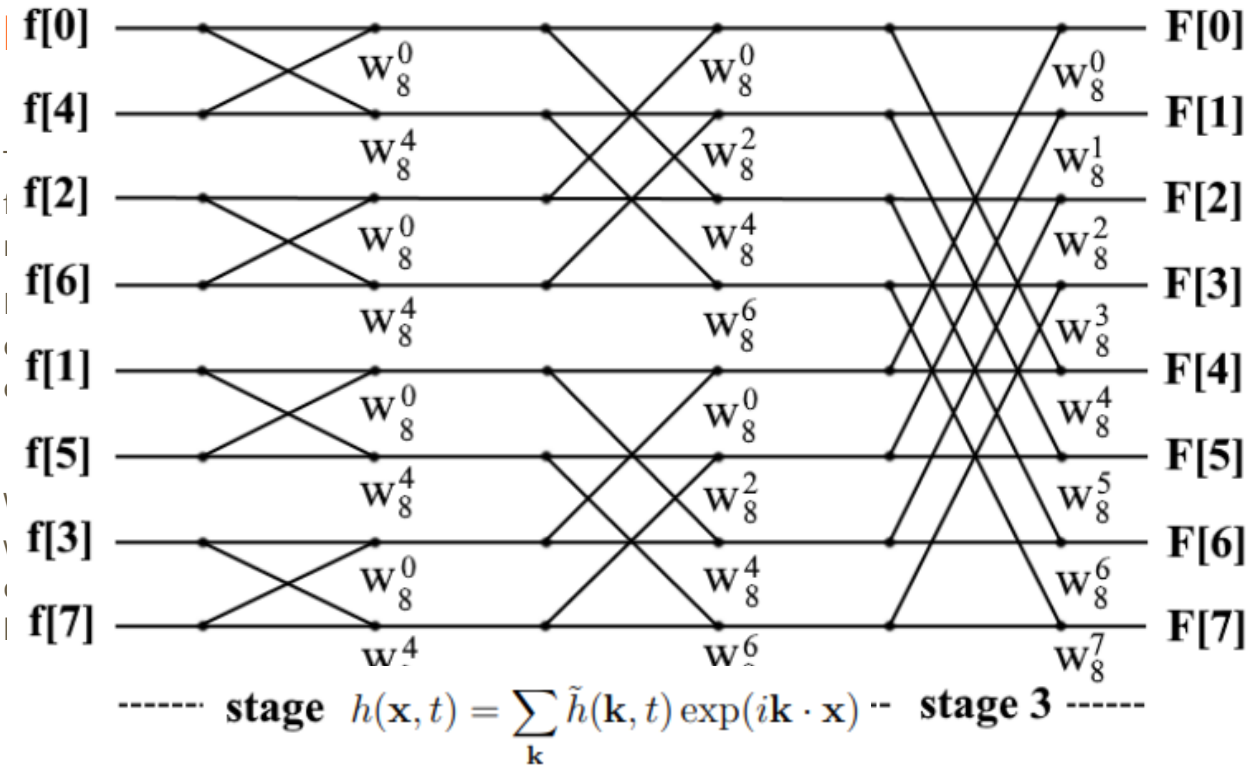
since,

$$e^{i\frac{2\pi k}{N}} = e^{i\frac{2\pi N+k}{N}}$$

Thus,

$$W_N^k = W_N^{k+mN} \quad \forall m \in \mathbb{Z}$$

The number of butterfly stages can be generically determined to log2 n stages, since the butterfly-span is doubled after each stage to a maximum span of N/2 in the last stage. Thus, the total execution time of the FFT with log2 n stages and O(n) for each stage is O(n log n). Due to the reason that the butterflyspan is doubled after each stage, the Cooley-Tukey FFT algorithm is a divide and conquer algorithm. Since the algorithm can only be applied on DFT's with N as a power of two, it is also called Radix-2 FFT algorithm.



The wavevector k is a two-dimensional horizontal vector, which points in the direction of travel of the wave. The components kx and kz of **k = (kx, kz)** as,

$$k_x = (2\pi n/L) \quad \text{and} \quad k_z = (2\pi m/L)$$

where n and m have bounds: **− N/2 ≤ n, m < N/2**

L is the horizontal dimension of the patch. The height amplitude Fourier components h˜(k, t) are generated by gaussian random numbers and a spatial spectrum. Oceanographic research showed with reference to statistical analysis, photographic and radar measurements of the ocean surface that the wave height amplitudes h˜(k, t) are approximately independent gaussian fluctuations with the Phillips spectrum Ph(k) by:

$$L = V^2/g$$

V is the wind speed, w is the direction of the wind and g = 9.8m/sec2 , the gravitational constant of the earth. The factor | ˆk · wˆ | 2 removes waves that are directed perpendicular to the wind direction vector.

For suppressing very small waves with l << L the Phillips spectrum can be multiplied with the factor exp(−k 2 l 2).

The Fourier amplitudes are denoted as :

where **ξr** and **ξi** are inde $\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P_h(\mathbf{k}}$ stribution with mean 0 and standard deviation

$$w^2(k) = gk$$

is integrated into the equation of the Fourier amplitudes. The dispersion relation is the relationship between frequencies and magnitude of the water wave vectors **ki** , where g is the gravitational constant. With the dispersion relation, the Fourier amplitudes at time t are denoted by

$$\tilde{h}(\mathbf{k}, t) = \tilde{h}_0(\mathbf{k}) \exp(iw(k)t) + \tilde{h}_0^*(-\mathbf{k}) \exp(-iw(k)t)$$

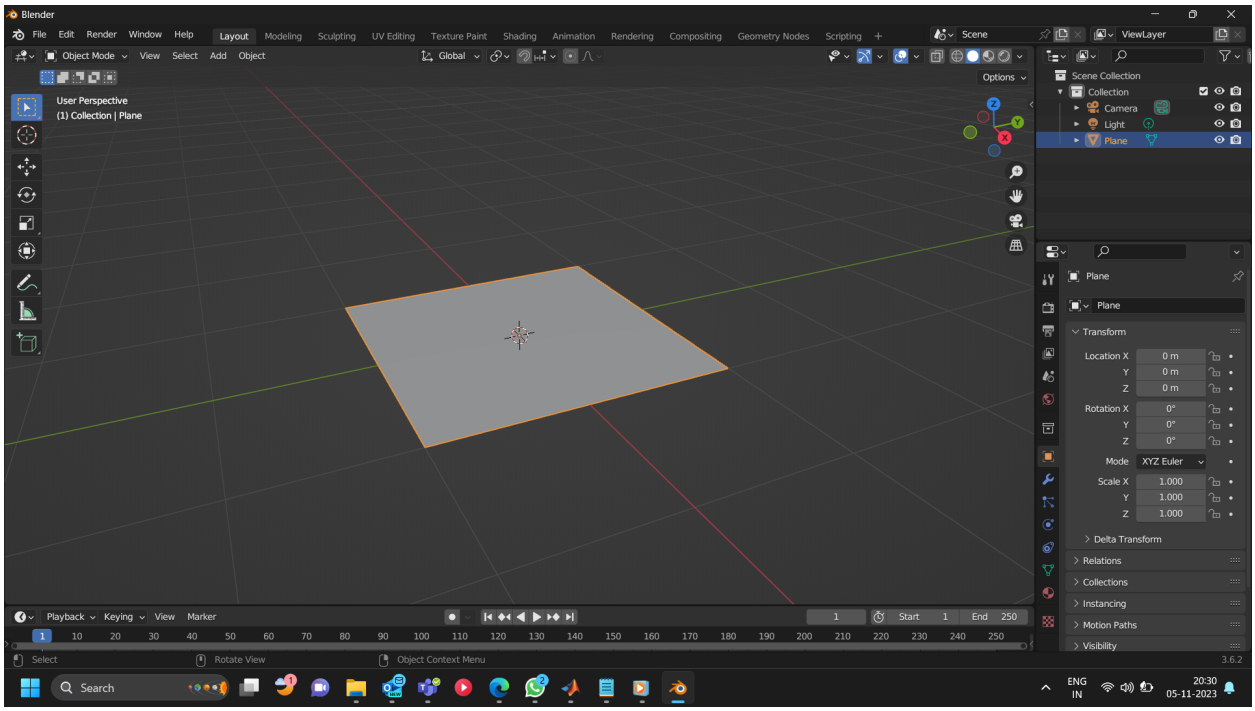
During our research we came across multiple coefficients, we have used the one above for our simulation, but the one provided below in reference to another paper is given below which also seemed to provide very promising results.

$$h_0 = \frac{1}{\sqrt{2}}(\xi_1 + i\xi_2)\sqrt{2S(\omega)D(\theta,\omega)\frac{d\omega(k)}{dk}\frac{1}{k}\Delta k_x \Delta k_z}$$

Blender :

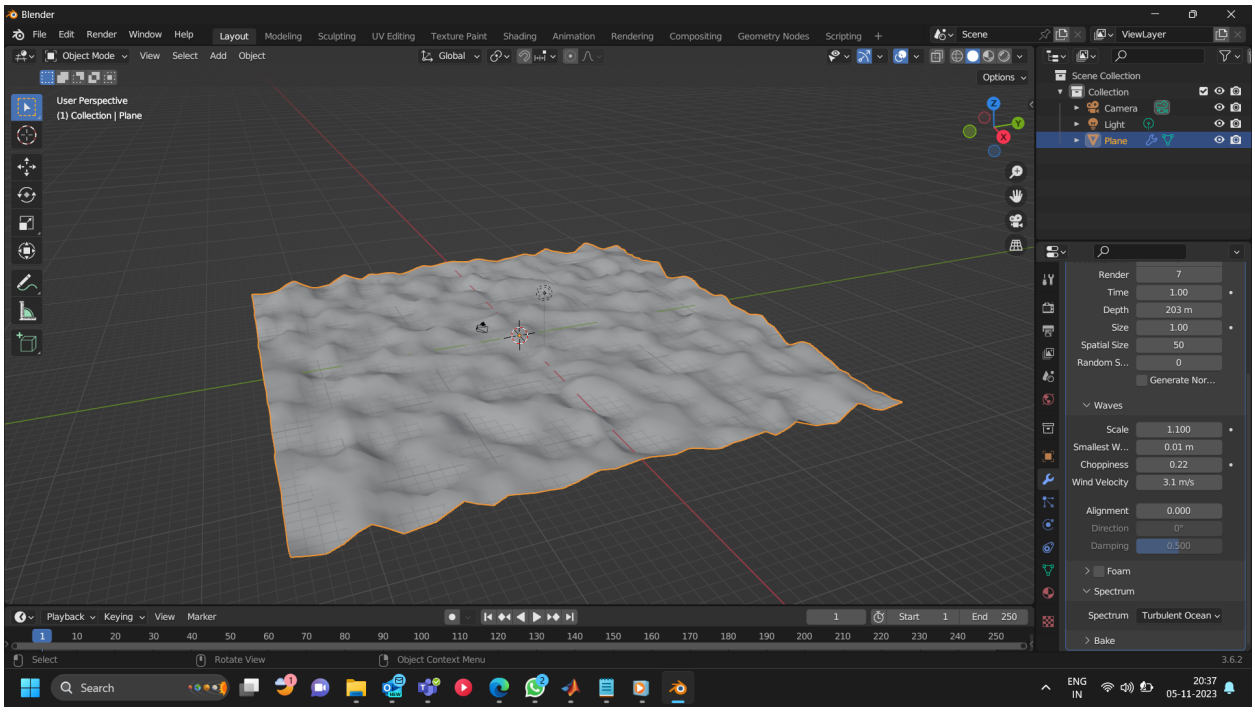
We have used blender to render the 3D simulations due to its versatility and some very useful modifiers it packs.

In this page we have created a plane to start of with, we will call this a tile.

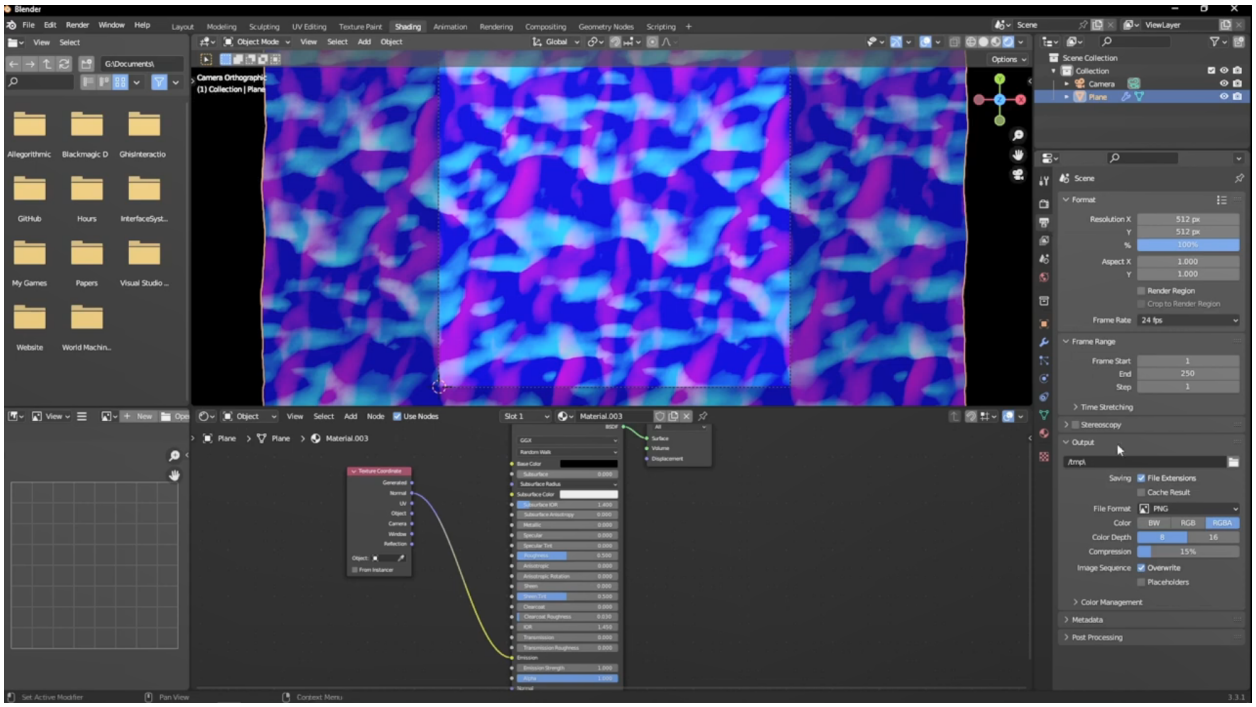


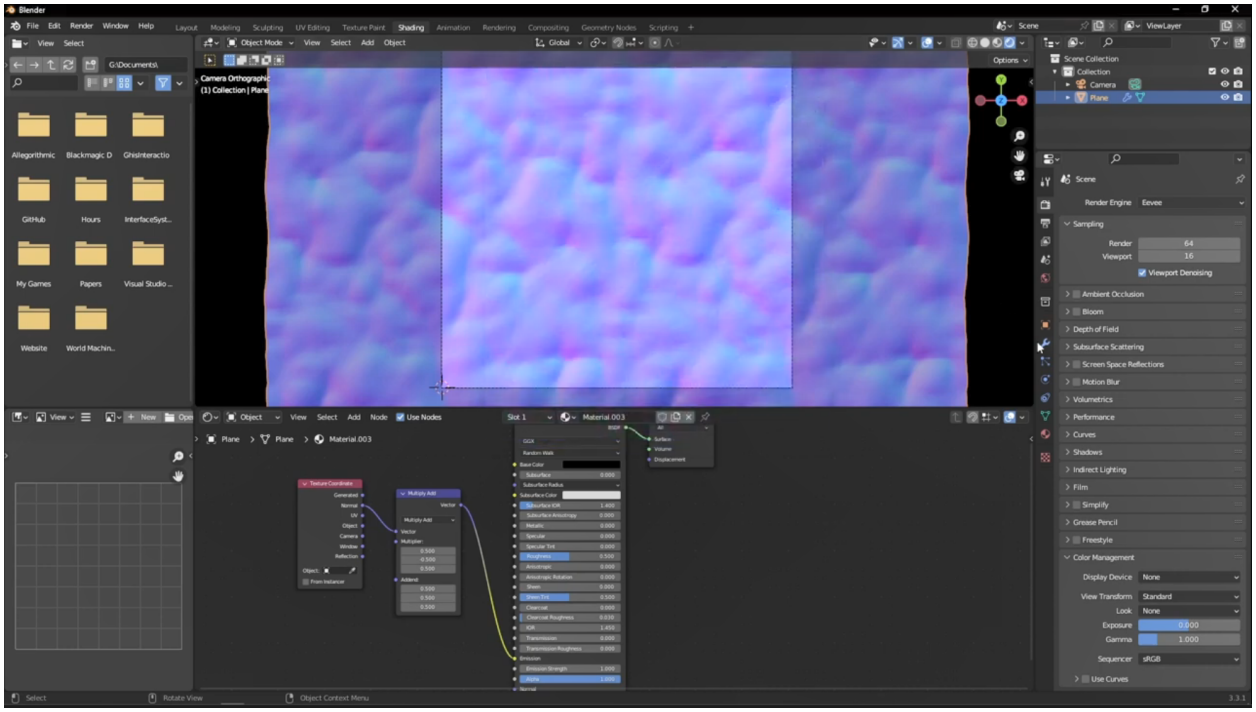
We have imported an ocean modifier here and used geometry nodes, to assign these functions : `x = R * t - r * sin(t);`

`y = R - r * cos(t);`

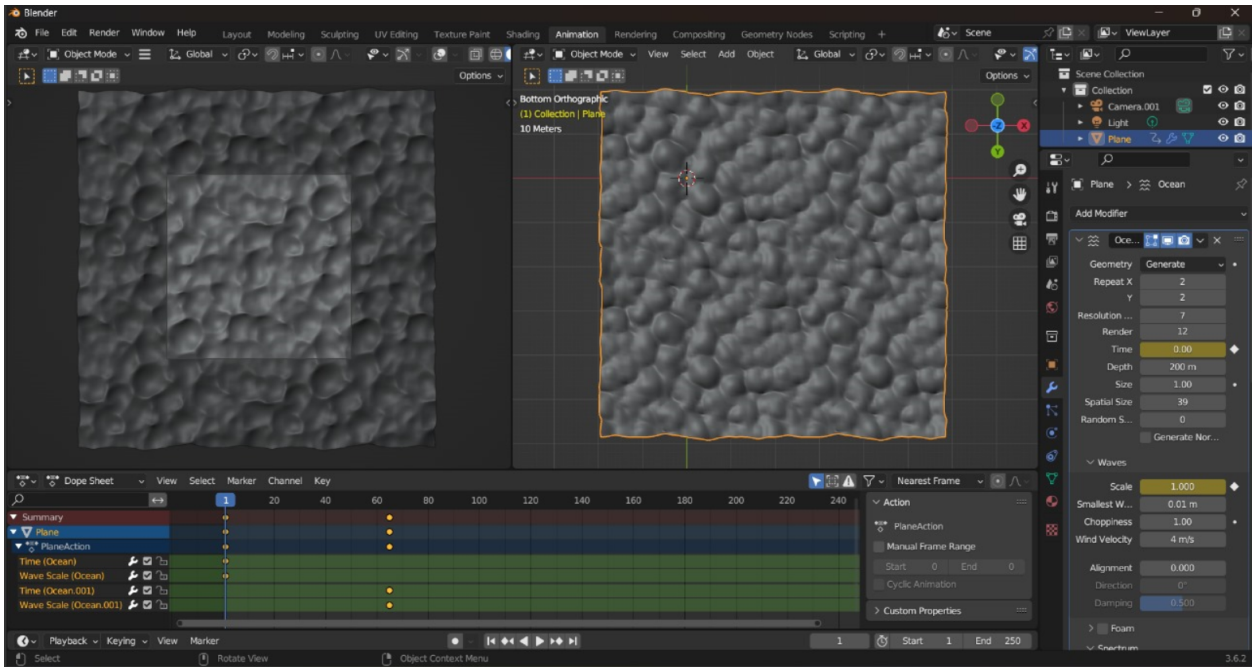


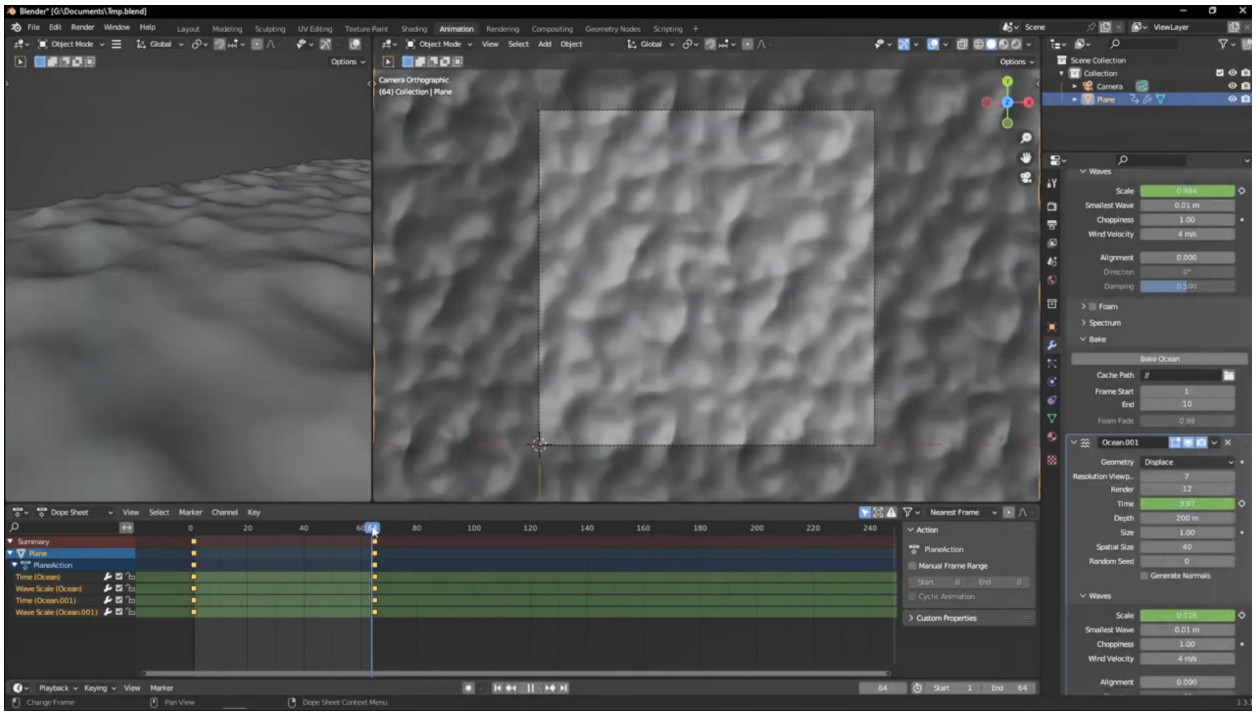
In this page we are working in the shading tab to assign vectors for respective tiles to provide smoother repetition upon adding array of multiple tiles.





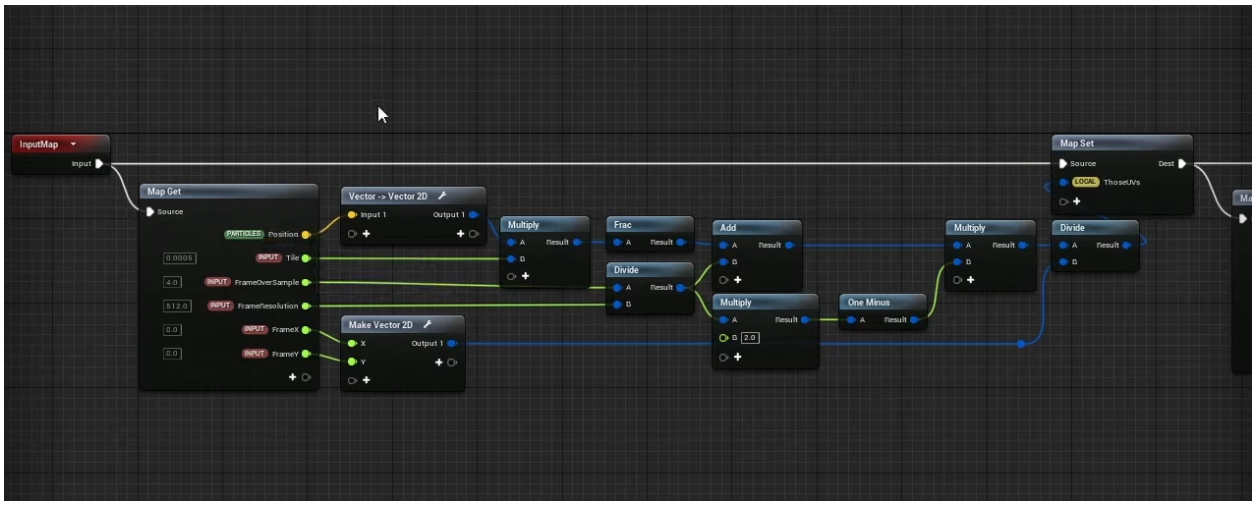
The left side frame is the top view of the ocean waves and the right side is the bottom view (-Z). It can be seen that the top view carries a distinct sharpness like in 2D trochoidal waves and also Gerstner waves. The bottom view shows the waves smoother portions like in a trochoidal wave. With this we have successfully managed to acquire 3D Gerstner waves with some noise (Noise covered above).



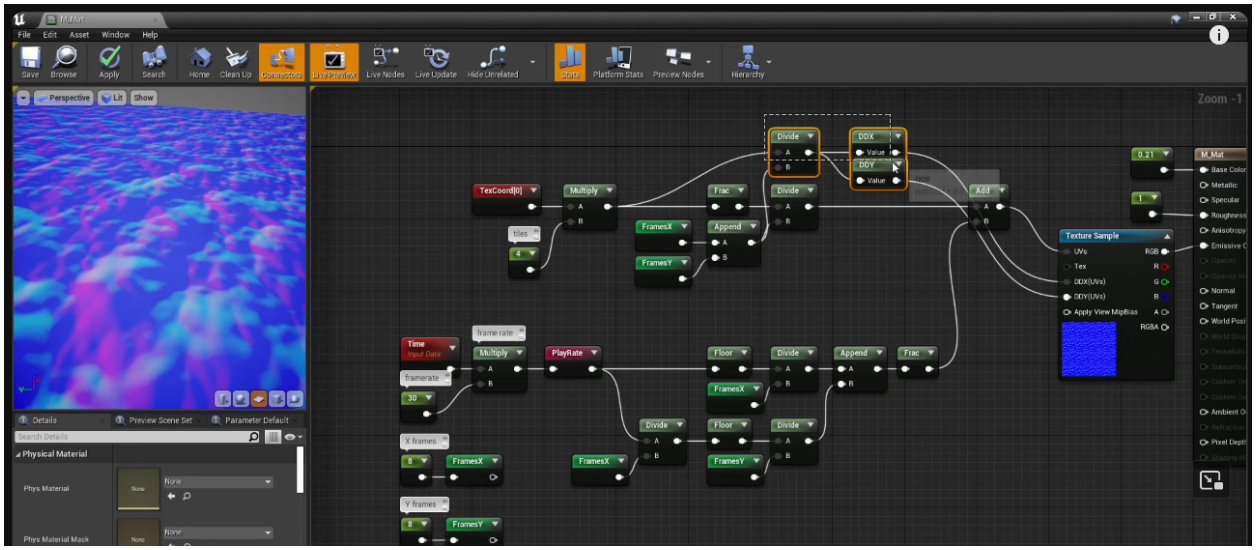


Flip book addon :

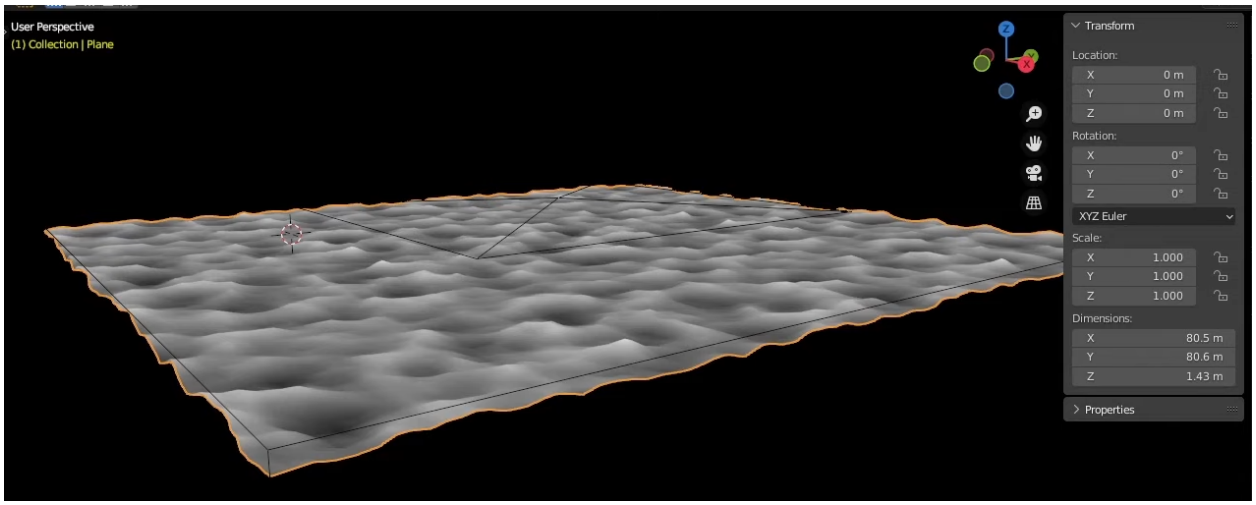
Flipbook Addon is a Blender add on that helps you create realistic flipbooks. It gets updated on your requirements. The flip book addon is associated with graphic design software, e-publishing tools, or even website development platforms that enable the creation of interactive, flipbook-style content. It could provide functionalities for creating, customizing, and sharing digital flip books, adding features like page-flipping effects, multimedia integration, and user-friendly interfaces.



We have generated a plane in the flip book add on. We have selected a part of the plane, say a tile in it. Then normals are set to the random surfaces ,now using an algorithm a noise coefficient for the tile is generated and added a texture to it. Through the Z dimension the height of the surfaces are decided using Sampling. This gives the model of a 3D wave . The Tiles are multiplied to match the edges and to generate a long distant view of the ocean. Animations are added to the file to generate the realistic movement of the wave. **This is essentially where we have run the FFT algorithms**



Randomness and a texture is added to the plane using this filp book algorithm.



Heights of the surface defined using Sampling (Crust and Trough with varying magnitudes aligned continuously) is represented above.

Applications:

Oceanography and Climate Studies:

Oceanographic research often involves studying wave patterns and their impact on the environment. Realistic wave simulations can aid in better understanding ocean dynamics, contributing to research on climate change, coastal erosion, and other environmental studies.

Developing Tsunami simulator:

Understanding wave mechanics, seismic activity, coastal geography, seafloor topography, energy transfer, and using computer modeling helps develop realistic simulations and early warning systems for tsunami preparedness and response.

Understanding ocean pattern:

Studying ocean waves is crucial for understanding ocean patterns as waves provide insights into how disturbances and energy propagate in the ocean, influencing long-term behaviors and variations in ocean conditions.

Predicting tide's nature:

Tides are influenced by lunar and solar gravitational forces, and the interplay between these forces and ocean waves helps in calculating and forecasting tidal patterns. Understanding coastal topography, harmonic analysis, and using computer models based on wave data is key to accurate tide prediction.

Analyzing behavior of water bodies in celestial object:

Studying ocean waves helps analyze the behavior of water bodies on celestial objects by providing insights into the presence of liquid water, environmental conditions, geological processes, climate, habitability, and aiding in the study of exoplanets with potential conditions for life.

Gaming and Animation:

In the gaming and animation industries, realistic ocean waves significantly enhance the visual quality of games, simulations, and animated scenes. Accurate wave simulations create immersive environments, improving the overall gaming experience and visual appeal of animated content.

Scientific Research:

Ocean waves play a role in the transport of heat and nutrients in the oceans, affecting global climate systems and marine life. Scientists study waves to better comprehend these complex interactions.

Defense and Security:

Nations with coastlines may study ocean waves for defense and security purposes, such as protecting naval assets and monitoring maritime traffic.

Coastal Management:

Coastal communities need to study ocean waves to protect their shorelines from erosion, storm surges, and flooding. This information is vital for designing effective coastal defenses and infrastructure.

Conclusion:

The project's accomplishments in successfully implementing FFT for simulating ocean waves is summarized, highlighting its potential implications across various industries. It also addresses the potential future enhancements or research directions for more precise and efficient ocean wave simulations.Its outcomes present opportunities for innovation, impacting diverse sectors and furthering our understanding of the intricate nature of oceanic behavior.The project not only shows the potential for technological innovation but also underlines the significance of understanding natural phenomena for practical applications and furthering scientific research.

Continued research and innovation in FFT technology promise further advancements in our comprehension and utilization of ocean wave data.

Our Rendered Results :