THE COLLEGE OF
WOOSTER

# A Performance Analysis of the Sum of Sines and Fast Fourier Transform Algorithms for Real-Time Ocean Rendering

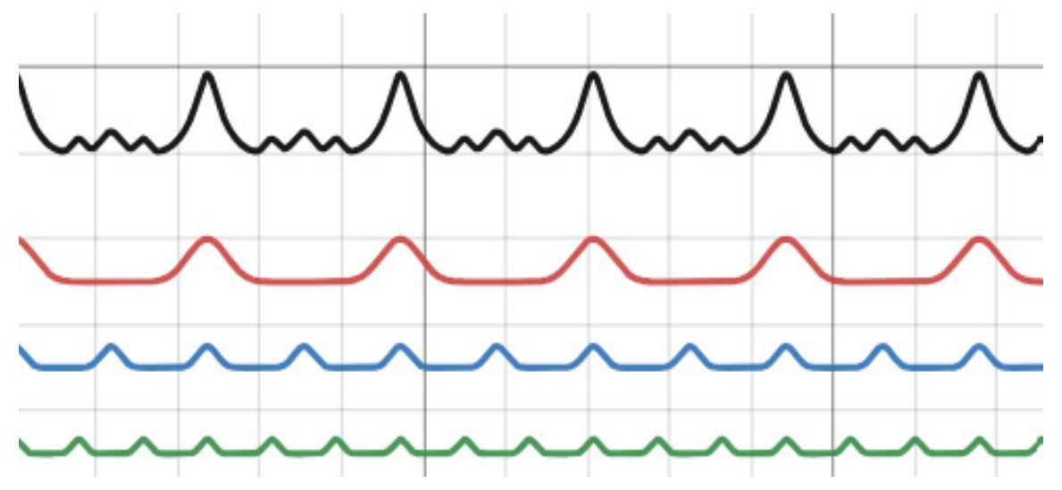
Kaiya Magnuson ‘24, Department of Mathematics and Computational Sciences
Advised by Dr. Asa’d As’ad

## Sum of Sines

This foundational algorithm adds several sine waves of varying amplitude and frequency to produce one highly detailed wave that is animated over time.

$$\underbrace{H(x,y,t)}_{\text{vertex height}} = \sum_{i=1}^{n}(\underbrace{A_i}_{\text{amplitude}} * sin(\underbrace{D_i}_{\text{direction}} \cdot (x,y) * \underbrace{w_i}_{\text{frequency}} + t * \underbrace{\rho_i}_{\text{phase shift}}))$$

$$\begin{aligned} H(x, y, t) = {} & \\ & A_1 sin(x * w_1 + t * p_1) + \\ & A_2 sin(x * w_2 + t * p_2) + \\ & A_3 sin(x * w_3 + t * p_3) + \end{aligned}$$

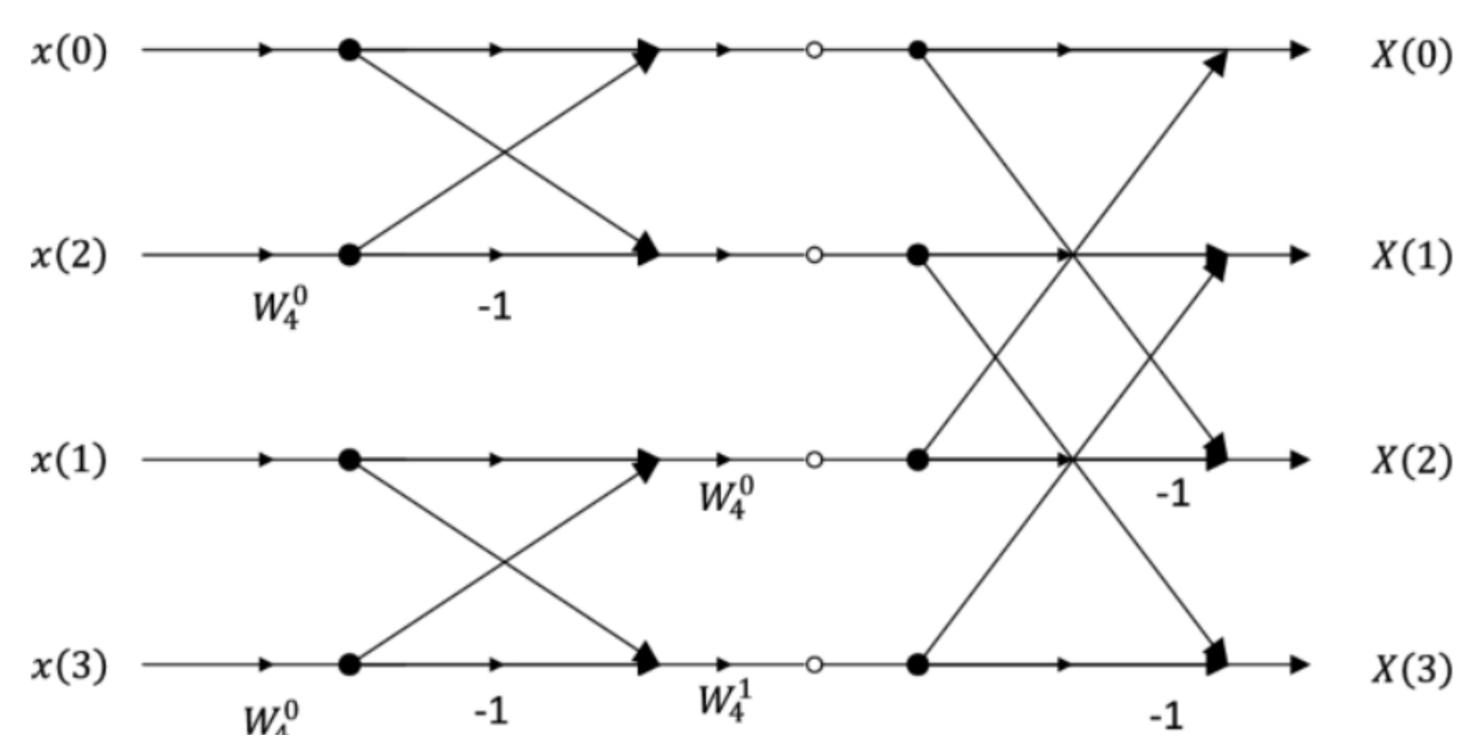


Algorithmic Complexity: *O(m*n)*, *m*=number of mesh vertices, *n*=number of waves

## Fast Fourier Transform

The Cooley-Tukey FFT converts frequency data from the Phillips spectrum to a sequence of wave amplitude data over time. Its efficiency makes it the most popular ocean modelling algorithm.

$$\underbrace{X(k)}_{\text{frequency component}} = \sum_{n_2=0}^{N_2-1}(\sum_{n_1=0}^{N_1-1} \underbrace{x(N_2 n_1 + n_2)}_{\text{time series component}} \underbrace{W_{N_1}^{n_1 k_1} * W_N^{n_2 k_1}}_{\text{twiddle factors}}) * W_{N_2}^{n_2 k_2})$$



Algorithmic Complexity: *O(n log(n))*, *n*=number of waves

*Selected References: Jerry Tessendorf, “Simulating ocean water,” 2001. Nigel Ang et al., “The technical art of Sea of Thieves,” 2018. Marc Droske et al., “Path tracing in production: The way of water,” 2023. Baoquan Liu et al., “Building a real-time system on GPUs...,” 2023.*

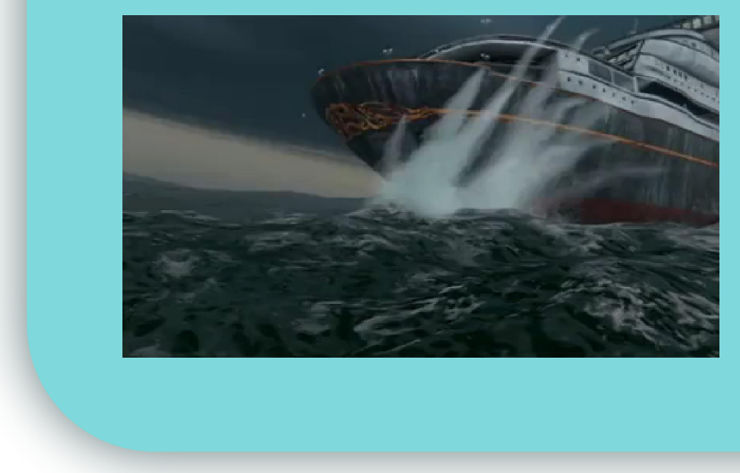
## Research Question:

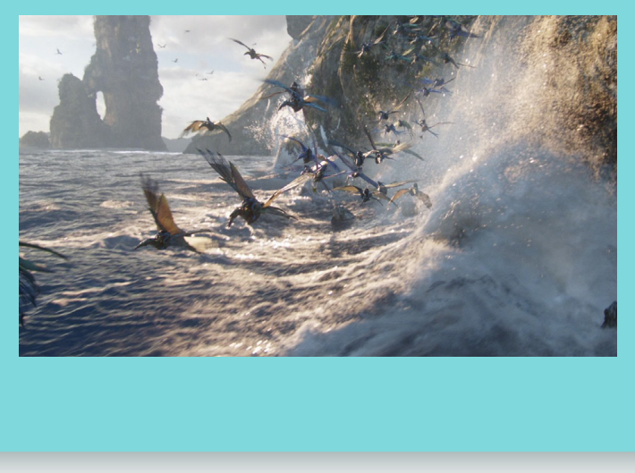
Which ocean modelling algorithms have the most significant impact on ocean shader performance, as measured by:

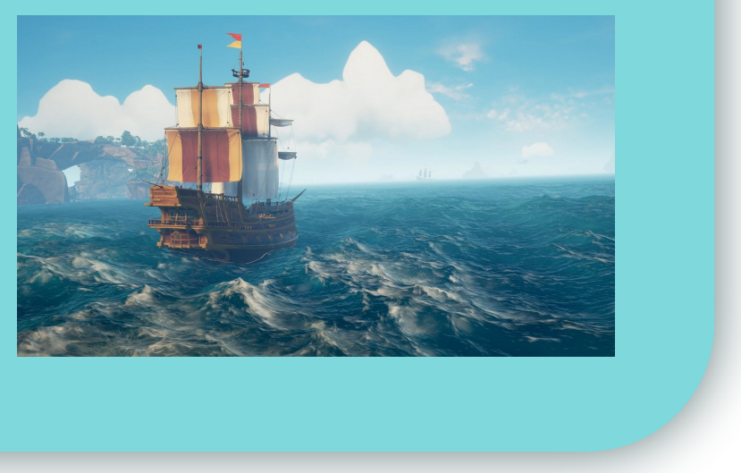
- frame rate (FPS)
- render time
- current memory usage

## Why is Water Rendering Important?

Real-time oceans establish atmosphere in video games and media. Oceans are often the most resource-intensive stage of rendering and professional shaders are rarely published.







## Objectives:

- Develop OpenGL vertex and fragment shaders
- Profile FPS, draw time, and memory usage
- Set guidelines for choosing ocean models

## Project Tools:

GODOT
Game engine
OpenGL
unity
C#

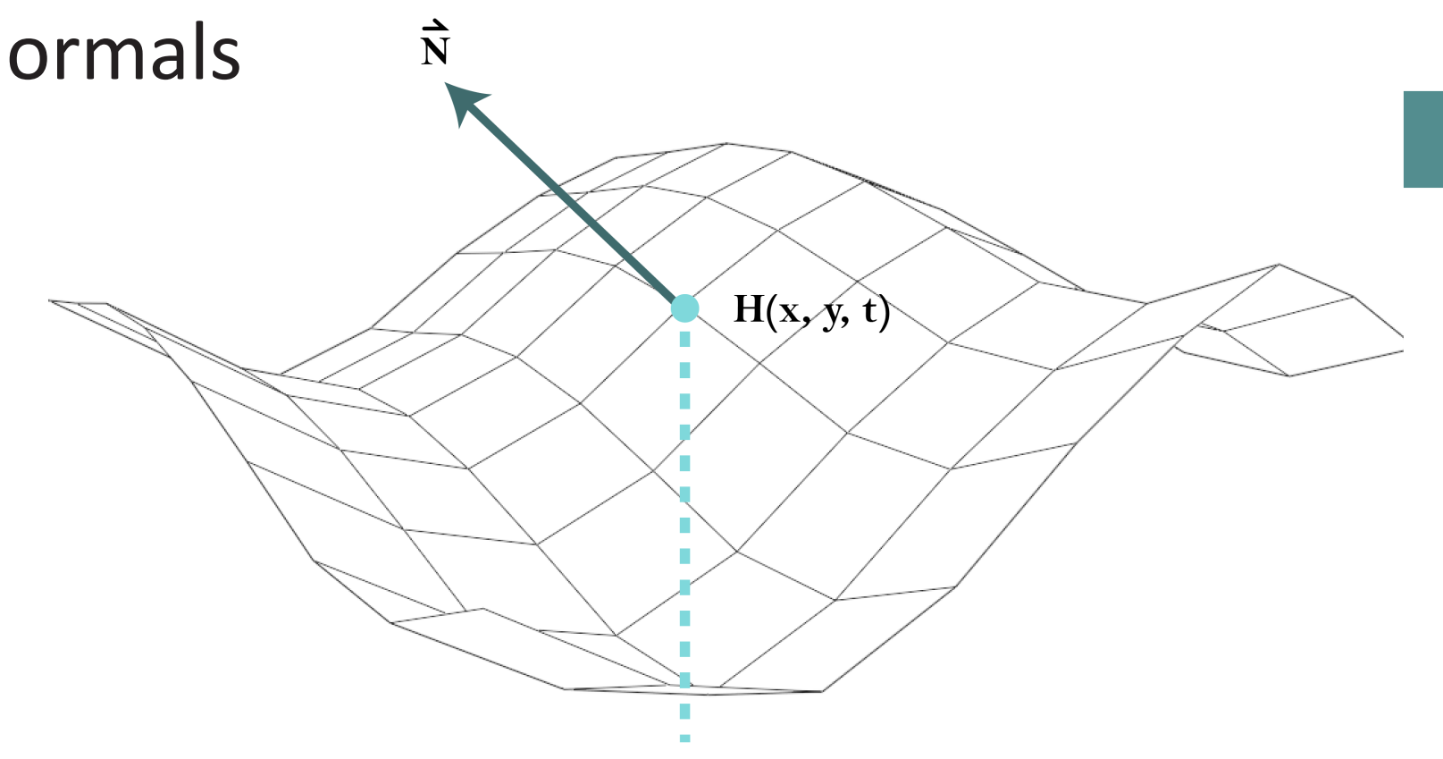
## Challenges & Limitations:

- Choosing compatible graphics APIs
- Identifying & developing realistic wave spectra
- Modifying open-source FFT compute shaders
- Constructive wave interference at high wave counts
- Limited configurations tested ➞ possibility of unidentified trends
- Profiling GPU is less powerful than the research standard

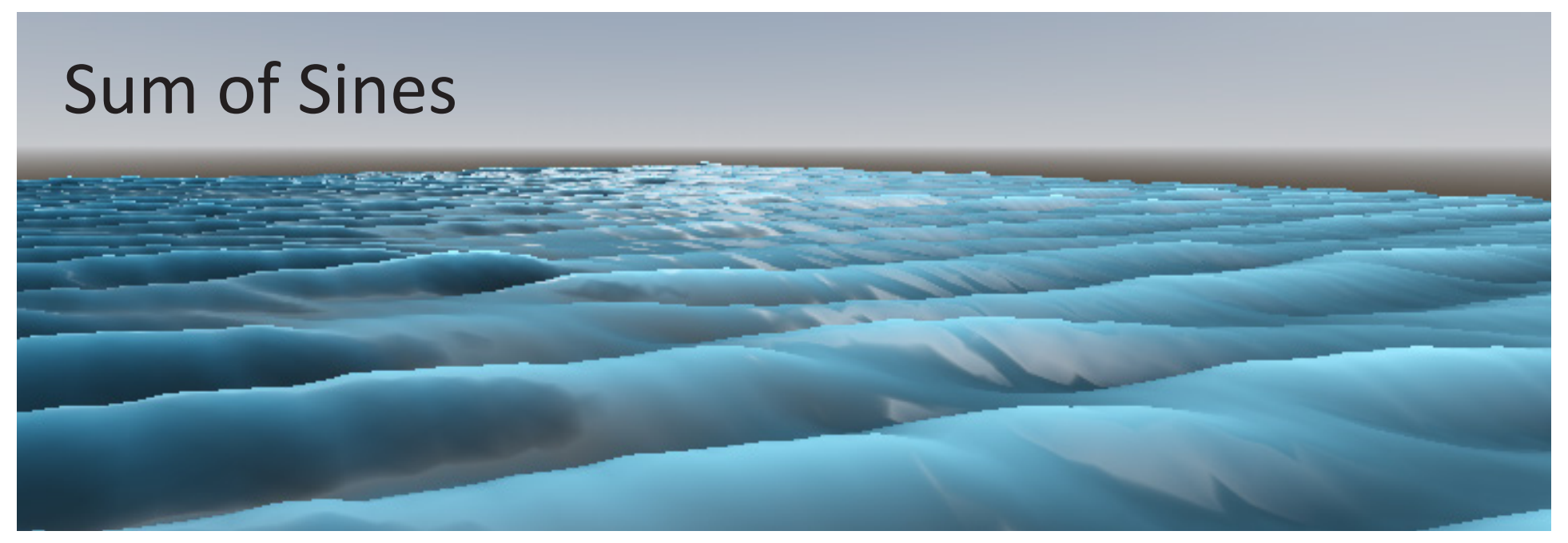
## Graphics Pipeline:

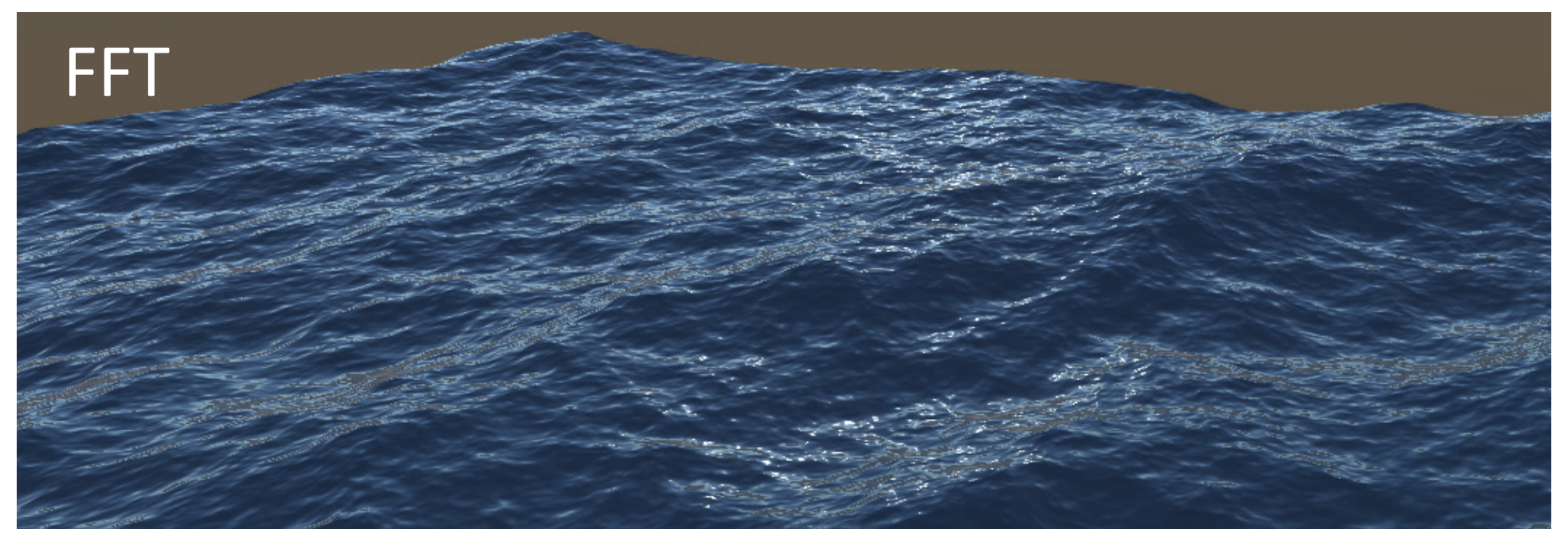
Vertex Shader:
*for each point on a mesh:*

- Calculate height
- Calculate normals



Fragment Shader:
*applies pixel color*



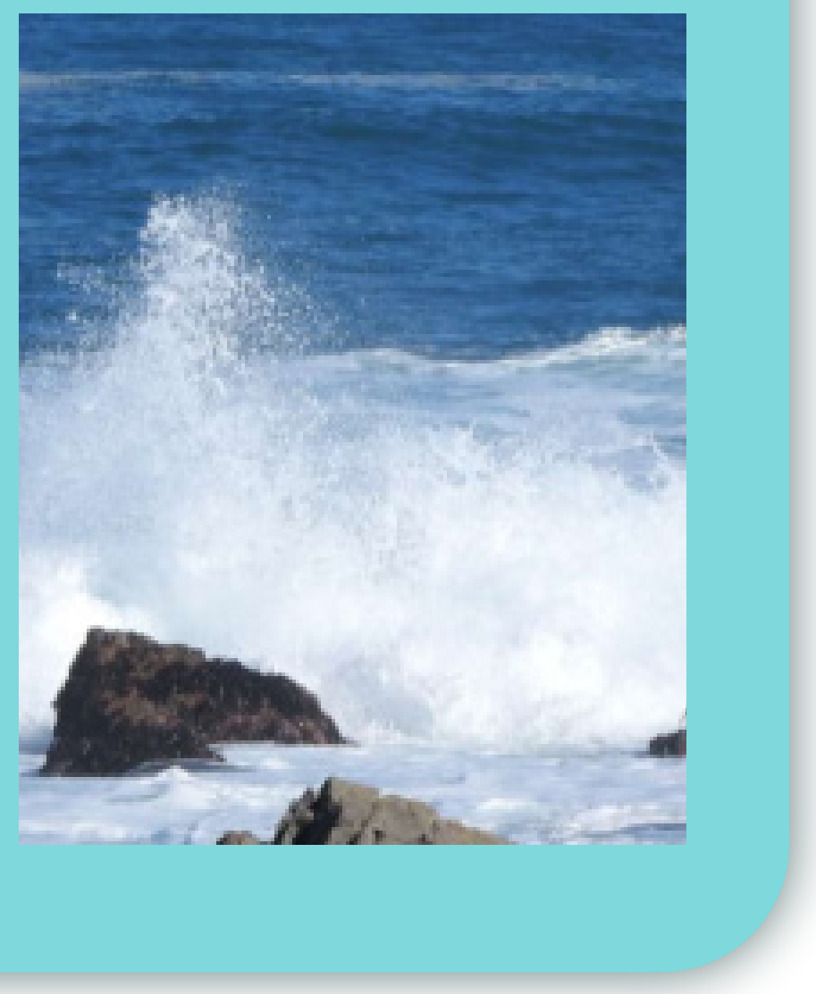


## Concluding Analysis:

The sum of sines algorithm produces stylized oceans for wave counts under 256. At low mesh resolutions and high wave counts, it has an unexpectedly higher average frame rate than the FFT. Despite being more complex, the FFT is recommended for ocean water since it remains realistic at high wave counts and outperforms the sum of sines at high mesh resolutions. These trends are expected to apply to ocean shaders on any platform.

## Future Research:

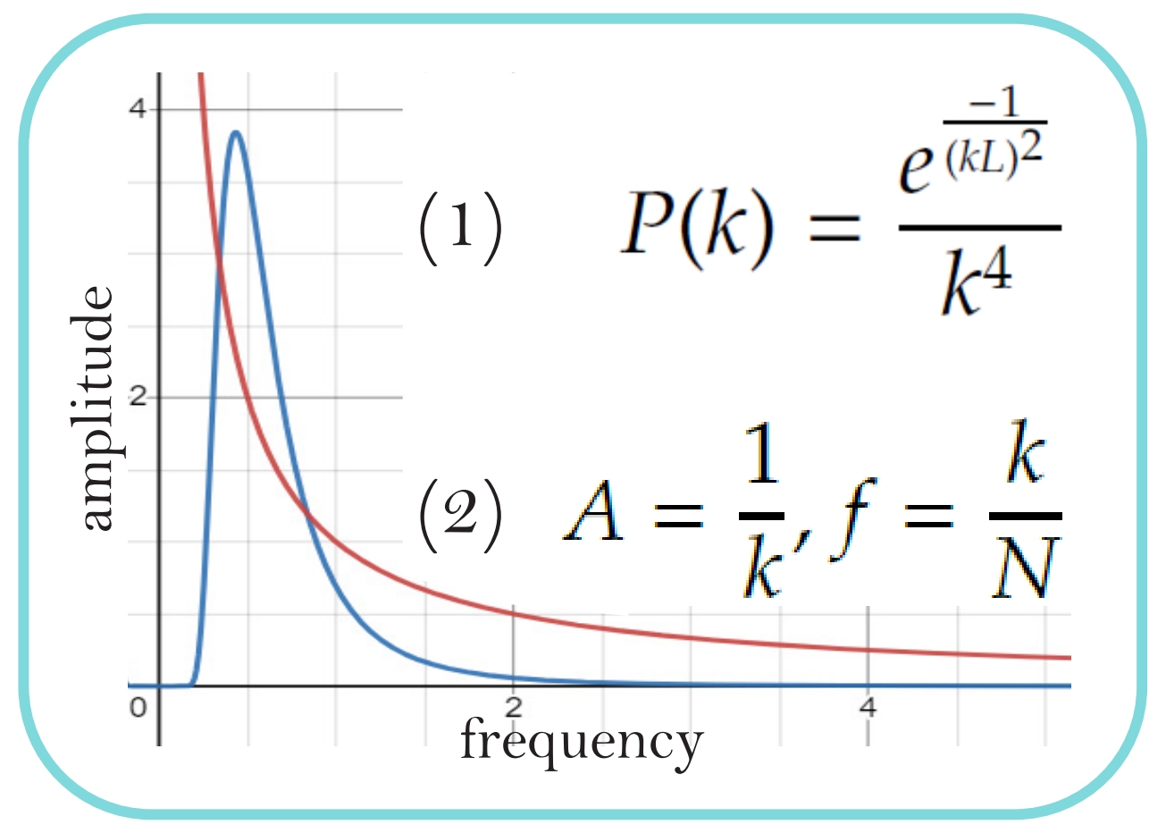
- Add level-of-detail cascades to improve performance
- Profile the effect of varying foam textures
- Implement realistic ray-traced lighting
- Investigate emerging particle simulations



## Achieving Realism

### Wave Energy Spectra

Energy spectra relate wave frequency and amplitude. (1) The Phillips spectrum is oceanographically accurate, (2) the custom dispersion function produces stylized waves.



$$(1) \quad P(k) = \frac{e^{\frac{-1}{(kL)^2}}}{k^4}$$

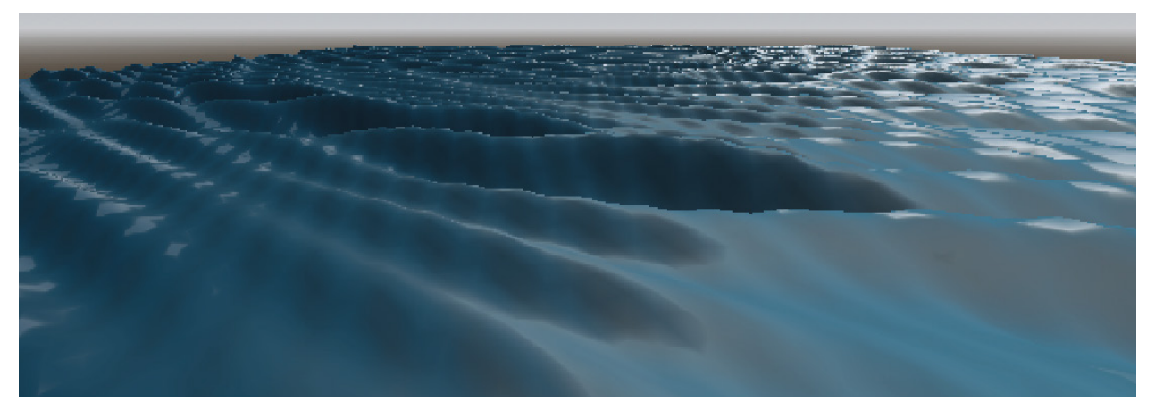
$$(2) \quad A = \frac{1}{k}, f = \frac{k}{N}$$

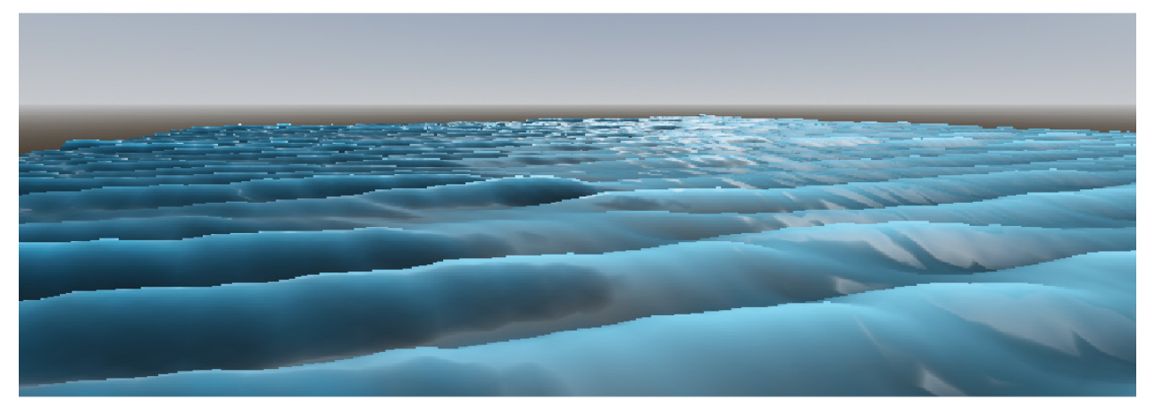
### Light Transmissivity

More light passes through wave crests than through wave troughs.

$$I(h) = e^{p*h} + t$$

*p*=peak intensity,
*t*=trough intensity

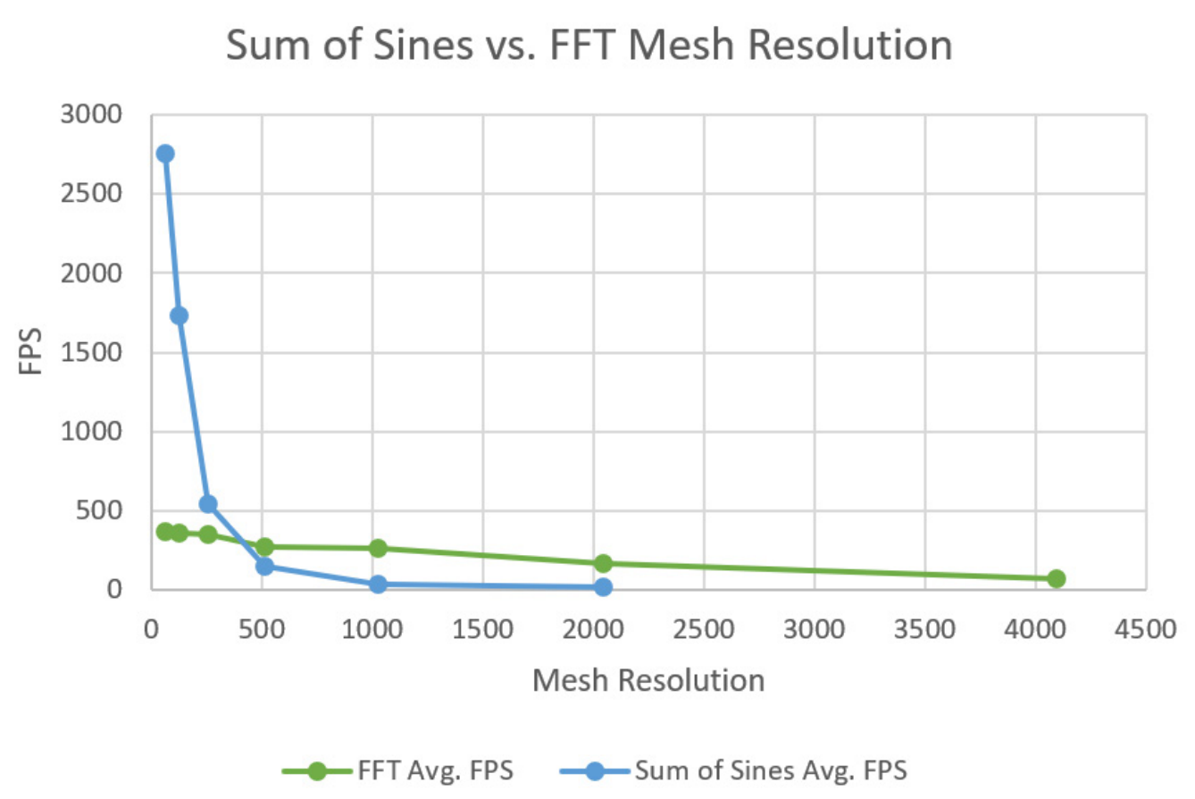




## Performance Profiling

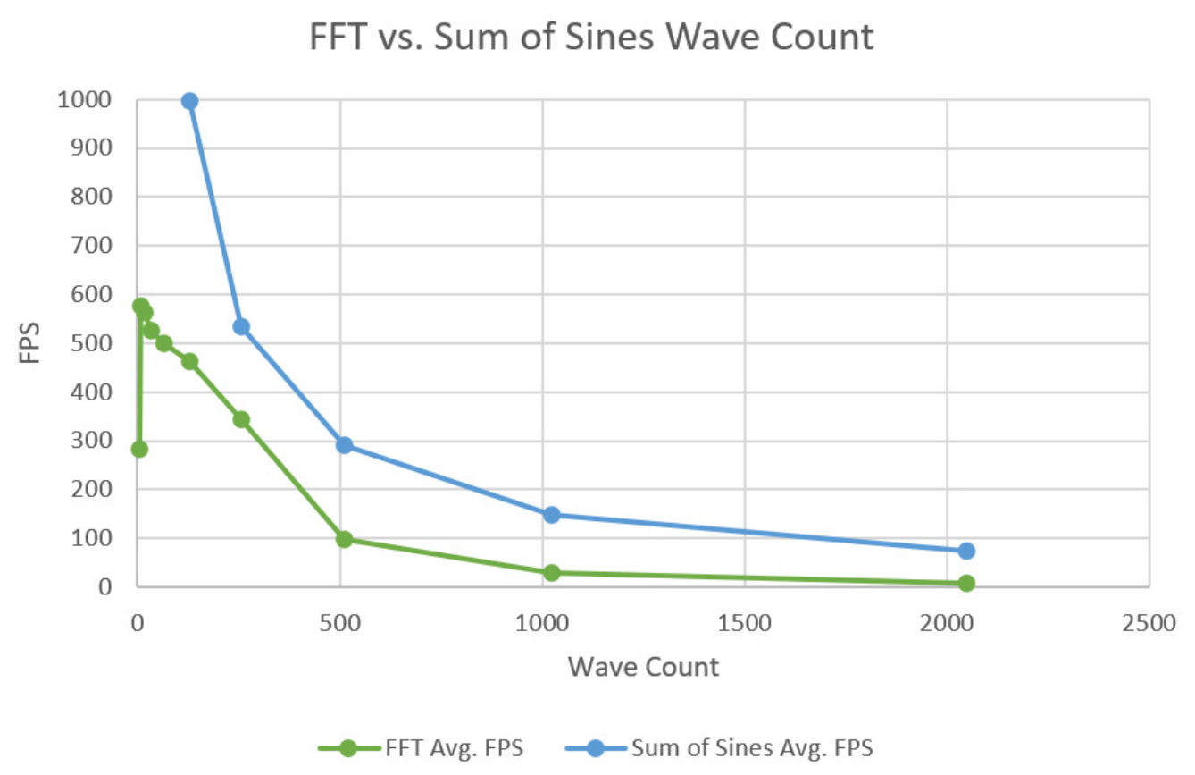
*36 configurations tested on the NVIDIA Quadro P4000*

### Adjusting Mesh Resolution



The FFT is more efficient at higher resolutions (above 256x256), which follows expected algorithmic complexity.

### Adjusting Wave Count



Unexpectedly, the sum of sines is more efficient for all wave counts tested. This could be due to buffer transfer costs and differing Unity and Godot pipelines.