NEA
Real-Time, Empirical Ocean Simulation & Physically Based Renderer

Zayaan Azam

Contents

0.1. Abstract

// synopsis

1. Analysis

1.1. Client Introduction

The client is Jahleel Abraham. They are a game developer who require a physically based, performant, configurable simulation of an ocean for use in their game. They also require a physically based lighting model derived from microfacet theory, including PBR specular, and empirical subsurface scattering. Also expected is a fully featured GUI allowing direct control over every input parameter, and a functioning orbit camera.

1.2. Interview Questions

1 Functionality
  1.1 “what specific ocean phenomena need to be simulated? (e.g. waves, foam, spray, currents)”
  1.2 “what parameters of the simulation need to be configurable?”
  1.3 “does there need to be an accompanying GUI?”
2 Visuals
  2.1 “do i need to implement an atmosphere / skybox?”
  2.2 “do i need to implement a pbr water shader?”
  2.3 “do i need to implement caustics, reflections, or other light-related phenomena?”
3 Technologies
  3.1 “are there any limitations due to existing technology?”
  3.2 “does this need to interop with existing code?”
4 Scope
  4.1 “are there limitations due to the target device(s)?”
  4.2 “are there other performance intesive systems in place?”
  4.3 “is the product targeted to low / mid / high end systems?”

1.3. Interview Notes

1 Functionality
  1.1 it should simulate waves in all real world conditions and be able to generate foam, if possible simulating other phenomena would be nice.
  1.2 all necessary parameters in order to simulate real world conditions, ability to control tile size / individual wave quantity
  1.3 accompanying GUI to control parameters and tile size. GUI should also output debug information and performance statistics
2 Visuals
  2.1 a basic skybox would be nice, if possible include an atmosphere shader
  2.2 implement a PBR water shader, include a microfacet BRDF
  2.3 caustics are out of scope, implement approximate subsurface scattering, use beckmann distribution in combination with brdf to simulate reflections
3 Technologies
  3.1 client has not started technical implementation of project, so is not beholden to an existing technical stack
  3.2 see response 3.1
4 Scope
  4.1 the simulation is intended to run on both x86 and arm64 devices
  4.2 see response 3.1
  4.3 the simulation is targeted towards mid to high end systems, ideally the solution would also be performant on lower end hardware

1.4. Technologies (Unfinished)

- Rust:
  - Fast, memory efficient programming language
- WGPU:
  - Graphics library
- Rust GPU:
  - (Rust as a) shader language
- Winit:
  - cross platform window creation and event loop management library
- Dear ImGui
  - Bloat-free GUI library with minimal dependencies
- Naga:
  - Shader translation library
- GLAM:
  - Linear algebra library
- Nix:
  - Declarative, reproducible development environment

1.5. Algorithm Overview (Unfinished)

// like 20x more complex than this Note that throughout this project we are defining the positive y direction as "up".

startup:

- generate gaussian random number pairs and store into texture on cpu

param change:

- generate energy spectrum for every wave and store into texture
- generate dispersion relation for every wave and store into texture

every frame:

- evolve spectrum
- inverse fft for all 3 axes
- inverse fft for all 5 derivatives
- store results into textures
- displace vertices per textures
- compute jacobian of textures
- inject foam into foam texture
- lighting
- color pixels for foam
- exponential decay function on foam

1.6. Spectrum Generation

1.6.1. Dispersion Relation [1], [2]

The relation between the travel speed of the waves and their wavelength, written as a function relating angular frequency ω to wave number $\vec{k}$. This simulation involves finite depth, and so we will be using a dispersion relation that considers it. This dispersion relation also considers capillary waves using an approximate relationship between surface tension and density [2].

$$\omega = \varphi(k) = \sqrt{\left(gk + \frac{\sigma}{\rho}k^3\right)\tanh(kh)}$$

$$\frac{d\varphi(k)}{dk} = \frac{h\left(\frac{\sigma}{\rho}k^3 + gk\right)\operatorname{sech}^2(hk) + \left(gk + \frac{\sigma}{\rho}k^3\right)\tanh(kh)}{2\sqrt{\left(gk + \frac{\sigma}{\rho}k^3\right)\tanh(kh)}}$$

where

- g is gravity
- h is the ocean depth
- $k = |\vec{k}|$, defined with the wave summation below
- σ is the surface tension coefficient Nm^{-1}
- ρ is the density kg m^{-3}

1.6.2. Non-Directional Spectrum (JONSWAP) [2], [3], [4], [5]

The JONSWAP energy spectrum is a more parameterised version of the Pierson-Moskowitz spectrum, and an improvement over the Philips Spectrum used in [1], simulating an ocean that is not fully developed (as recent oceanographic literature has determined this does not happen). The increase in parameters allows simulating a wider breadth of real world conditions.

$$S_{\text{JONSWAP}}(\omega) = \frac{\alpha g^2}{\omega^5}\ \exp\ \left[-\beta\left(\frac{\omega_p}{\omega}\right)^4\right]\gamma^r$$

$$r = \exp\left[-\frac{\left(\omega - \omega_p\right)^2}{2w_p^2\sigma^2}\right]$$

$$\alpha = 0.076\left(\frac{U_{10}^2}{Fg}\right)^{0.22}$$

$$\omega_p = 22\left(\frac{g^2}{U_{10}F}\right)^{\frac{1}{3}}$$

$$\sigma = \begin{cases} 0.07 \text{ if } \omega \leq \omega_p \\ 0.09 \text{ if } \omega > \omega_p \end{cases}$$

where
- α is the intensity of the spectra
- $\beta = \frac{5}{4}$, a "shape factor", rarely changed [3]
- $\gamma = 3.3$
- $\omega = \varphi(k)$, the dispersion relation
- ω_p is the peak wave frequency
- U_{10} is the wind speed at $10m$ above the sea surface [3]
- F is the distance from a lee shore (a fetch) - distance over which wind blows with constant velocity [2], [3]
- g is gravity

1.6.3. Depth Attenuation Function (Approximation of Kitaiigorodskii) [2]

JONSWAP was fit to observations of waves in deep water. This function adapts the JONSWAP spectrum to consider ocean depth, allowing a realistic look based on distance to shore. The actual function is quite complex for a relatively simple graph, so can be well approximated as below [2].

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

$$\omega_h = \omega\sqrt{\frac{h}{g}}$$

where
- $\omega = \varphi(k)$, the dispersion relation
- h is the ocean depth
- g is gravity

Given time, I may derive a simpler approximation using $\tanh(k)$ as that seems interesting.

1.6.4. Directional Spread Function (Donelan-Banner) [2]

The directional spread models how waves react to wind direction [4]. This function is multiplied with the non-directional spectrum in order to produce a direction dependent spectrum [2].

$$\theta = \arctan\left(\frac{k_z}{k_x}\right) - \theta_0$$

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

$$\beta_s = \begin{cases} 2.61\left(\frac{\omega}{\omega_p}\right)^{1.3} \text{ if } \frac{\omega}{\omega_p} < 0.95 \\ 2.28\left(\frac{\omega}{\omega_p}\right)^{-1.3} \text{ if } 0.95 \leq \frac{\omega}{\omega_p} < 1.6 \\ 10^{\varepsilon} \text{ if } \frac{\omega}{\omega_p} \geq 1.6 \end{cases}$$

$$\varepsilon = -0.4 + 0.8393 \exp\left[-0.567 \ln\left(\left(\frac{\omega}{\omega_p}\right)^2\right)\right]$$

where
- θ_0 is a wind direction offset

1.6.5. Swell (Unfinished) [2]

Swell refers to the waves which have travelled out of their generating area [2]. In practice, these would be the larger waves seen over a greater area. the directional spread function including swell is based on combining donelan-banner with a swell function as below. It is worth noting that, "bar the 'magic value' of 16 seen in s_ξ, the spectrum (and thus the simulation) is fully empirical" [2]. The integral seen in Q_{final} is computed numerically.

$$D_{\text{final}}(\omega, \theta) = Q_{\text{final}}(\omega) D_{\text{base}}(\omega, \theta) D_\varepsilon(\omega, \theta)$$

$$Q_{\text{final}}(\omega) = \left(\int_{-\pi}^{\pi} D_{\text{base}}(\omega, \theta) D_\xi(\omega, \theta) d\theta\right)^{-1}$$

$$D_\xi = Q_\xi(s_\xi) \ |\cos\left(\frac{\theta}{2}\right)|^{2s_\xi}$$

$$s_\xi = 16 \tanh\left(\frac{\omega_p}{\omega}\right)\xi^2$$

$$Q_\xi(s_\xi) =$$

where
- ξ is a "swell" parameter, in the range $0..1$
- Q_ξ is a normalisation factor to satisfy the condition specified in equation (31) in [2]

1.6.6. Directional Spectrum Function [2]

The TMA spectrum below is an undirectional spectrum that considers depth, combining the above functions.

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

This takes inputs ω, h, whilst we need it to take input $\vec{k}$ per Tessendorf [1] - in order to do this we apply the following 'transformation'. Similarly, to make the function directional, we also need to multiply it by the directional spread function [2].

$$S_{\text{TMA}}\left(\vec{k}\right) = 2S_{\text{TMA}}(\omega, h) D(\omega, \theta) \frac{d\omega(|k|)}{d\ |k|} \frac{1}{|k|} \Delta\vec{k}_x \Delta\vec{k}_z$$

$$\Delta\vec{k}_x = \Delta\vec{k}_z = \frac{2\pi}{L}$$

where

- L is the lengthscale defined below

1.7. Ocean Geometry & Foam (Unfinished)

1.7.1. the Statistical Wave Summation [1], [4], [5], [6], [7]

For a height field, of dimensions L_x and L_z, we calculate the height (h) at a position $\vec{x}$ by summating multiple sinusoids with complex, time dependant amplitudes. [1]. The frequency domain representation of the waves are converted to the spatial domain using an inverse discrete fourier transform. This is split into 2 components, with the derivatives computed seperately to find exact normals / the jacobian:

$$\text{Vertical Displacement (y)} : h(\vec{x}, t) = \sum_{\vec{k}} \hat{h}\Big(\vec{k}, t\Big) e^{i\vec{k}\cdot\vec{x}}$$

$$\text{Horizontal Displacement (x): } \lambda D_x(\vec{x}, t) = \sum_{\vec{k}} -i\frac{\vec{k}_x}{k} \hat{h}\Big(\vec{k}, t\Big) e^{i\vec{k}\cdot\vec{x}}$$

$$\text{Horizontal Displacement (z): } \lambda D_z(\vec{x}, t) = \sum_{\vec{k}} -i\frac{\vec{k}_z}{k} \hat{h}\Big(\vec{k}, t\Big) e^{i\vec{k}\cdot\vec{x}}$$

where

- t is the time
- $\vec{k} = [k_x, k_z]$, the wave vector, direction vector of the spectrum's texture
- $k = |\vec{k}|$, the magnitude of the wave vector
- $\vec{x} = [x_x, x_z]$, the direction vector for the height map for which we are summing
- $\hat{h}\Big(\vec{k}, t\Big)$ is the frequency spectrum function
- $h(\vec{x}, t)$ gives the vertical displacement vector at the point x at time t
- $\vec{D}(\vec{x}, t) = \Big[\vec{D}_x, \vec{D}_z\Big]$ gives the horizontal displacement at $\vec{x}$ at time t, used to simulate "choppy waves". We would split the normalised vector $\vec{k}$ into its components and compute them seperately [1]
- λ is a convenient scale factor "choppiness" in order to create sharper wave peaks [1]

1.7.2. Derivatives (Unfinished) [1], [4]

$$\text{Height Derivative} : \frac{\delta h(\vec{x}, t)}{\delta x} = \sum_{\vec{k}} i\vec{k}\hat{h}\Big(\vec{k}, t\Big) e^{i\vec{k}\cdot\vec{x}}$$

$$\text{Displacement Derivative (x) ?} : \lambda\nabla D_x(\vec{x}, t) = \sum_{\vec{k}} \vec{k}\frac{\vec{k}_x}{k} \hat{h}\Big(\vec{k}, t\Big) e^{i\vec{k}\cdot\vec{x}}$$

where

- $\nabla h(\vec{x}, t)$ gives the rate of change of the height, used to calculate the normal vector
- $\nabla \vec{D}(\vec{x}, t)$ gives the rate of change of the displacement, used to calculate the normal vector

1.7.3. Frequency Spectrum Function [1], [4], [5]

This function defines the amplitude of the wave at a given point in space at a given time depending on it's frequency. The frequency is generated via the combination of 2 gaussian random numbers and a energy spectrum in order to simulate real world ocean variance and energies.

$$\hat{h}\left(\vec{k},t\right) = \hat{h}_0\left(\vec{k}\right)e^{i\varphi(k)t} + h_0(-k)e^{-i\varphi(k)t}$$

$$\hat{h}_0\left(\vec{k}\right) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{S_{\text{TMA}}\left(\vec{k}\right)}$$

where

- $\hat{h}$ evolves $\hat{h}_0$ through time using eulers formula. by combining a positive and negative version of the wave number you ensure the functions output is real [1]
- $\hat{h}_0$ is the initial wave state as determined by the energy spectra & gaussian distribution. This is only computed on parameter change / startup and then stored into a texture
- ξ are gaussian random numbers defined below
- $S_{\text{TMA}}\left(\vec{k}\right)$ is the spectrum function defined above

1.7.4. Gaussian Random Numbers [2], [8]

The ocean exhibits gaussian variance in the possible waves. Due to this the frequency spectrum function is varied by gaussian random numbers with mean ($\tilde{x}$) 0 and standard deviation (σ) 1. [1]
In order to create repeatability, we create pseudorandom numbers for x, by by seeding a random number generator with a hash of the wave number truncated to 5 digits, preventing numerical imprecision from changing the seeding [2]. These are generated in pairs and then stored into the red and green channels of a texture to be accessed.

$$\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{\left(x-\tilde{x}^2\right)}{2\sigma^2}}$$

where

- σ is the standard deviation
- $\tilde{x}$ is the mean
- x is a random number, $-1..1$

1.7.5. Foam, The Jacobian & Eigenvalues [1], [2], [5], [6]

The jacobian describes the "uniqueness" of a transformation. This is useful as where the waves would crash, the jacobian determinant of the displacements goes negative. Per Tessendorf [1], we compute the determinant of the jacobian for the horizontal displacement, $D(\vec{x},t)$.

$$J(x) = J_{\text{xx}}J_{\text{zz}} - J_{\text{xz}}J_{\text{zx}}$$

$$J_{\text{xx}} = 1 + \lambda\frac{\delta D_x(\vec{x})}{\delta x}$$

$$J_{\text{zz}} = 1 + \lambda\frac{D_z(\vec{x})}{\delta z}$$

$$J_{\text{xz}} = J_{\text{zx}} = 1 + \lambda\frac{\delta D_x(\vec{x})}{\delta z}$$

we then threshold the value such that $J(x) < \mu$, wherein if true we "inject" foam (I_{injected}) into the simulation at the given point. This value should accumulate (and decay) over time to mimic actual

ocean foam, which can be achieved with the following, the result stored into a folding map texture. Finally, the folding map is multiplied by an artistic foam texture to add some detail.

$$I_{\text{decayed}} = I_0 e^{-\zeta}$$

$$I_{\text{final}} = I_{\text{decayed}} + I_{\text{injected}}$$

where
- I_0 is the previous foam value
- μ is a threshold value determining whether foam is injected
- ζ is a constant which determines the rate of decay
- λ is the choppiness parameter

1.7.6. Level of Detail (LOD) Optimisations (Unfinished) [6]

// i do not want to do this // will include frustum culling, gpu instancing & LOD scaling based on distance to camera

1.8. The IFFT (Unfinished)

Everything in this section is subject to significant change, I am opting not to work on this now so I can begin implementation faster

1.8.1. Cooley-Tukey Fast Fourier Transform (FFT) (Unfinished) [1], [4], [6]

The Cooley-Tukey FFT is a common implementation of the FFT algorithm used for fast calculation of the DFT. The direct DFT is computed in $O(N^2)$ time whilst the FFT is computed in $O(N \log N)$. This is a significant improvement as we are dealing with M (and N) in the millions.

complex, will write up after learning roots of unity & partial derivatives

1.8.2. The Inverse Discrete Fourier Transform (IDFT) (Unfinished) [1], [4], [6], [7]

The IDFT can be computed using the fast fourier transform if the following conditions are met:

- $N = M = L_x = L_z$
- the coordinates & wavenumbers lie on regular grids
- $N, M, L_x, L_z = 2^x$, for any positive integer x

For implementation, the statistical wave summation is represented in terms of the indices n' and m', where n', m' are of bounds $0 \leq n' < N$ & $0 \leq m' < M$

where

- N, M are the number of points & waves respectively, the simulation resolution
- L_x, L_z are the worldspace dimensions
- $\vec{k} = \left[\frac{2\pi n}{L_x}, \frac{2\pi m}{L_z}\right]$
- $\vec{x} = \left[\frac{nL_x}{N}, \frac{mL_z}{M}\right]$

note that in Tessendorf's paper [1], n & m are defined from $-\frac{N}{2} \leq n < \frac{N}{2}, -\frac{M}{2} \leq m < \frac{M}{2}$, but for ease of implemntation we shift the bounds (and all subsequent values) to begin at 0. I am thus glossing over some redundant information, further details on how / why are seen at [4], [7]

1.9. Post Processing

1.9.1. Rendering Equation [5], [9], [10]

This abstract equation models how a light ray incoming to a viewer is "formed" (in the context of this simulation). Due to there only being a single light source (the sun), subsurface scattering [9] can be used to replace the standard L_{diffuse} and L_{ambient} terms.

To include surface foam, we *lerp* between the foam color and L_{scatter} based on foam density [9]. We also Increase the roughness in areas covered with foam for L_{specular} [9].

$$L_{\text{eye}} = (1 - F)L_{\text{scatter}} + L_{\text{specular}} + FL_{\text{env\_reflected}}$$

where

- F is the fresnel reflectance term
- L_{scatter} is the light re-emitted due to subsurface scattering
- L_{specular} is the reflected light from the sun
- $L_{\text{env\_reflected}}$ is the reflected light from the environemnt

1.9.2. Normalisation & Vector Definitions [11]

When computing lighting using vectors, we are only concerned with the direction of a given vector not the magnitude. In order to ensure the dot product of 2 vectors is equal to the cosine of their angle we normalise the vectors. Henceforth, a vector $\vec{A}$ when normalised is represented with $\hat{A}$. Throughout all post processing effects a set of distinct vectors are used, defined as:

- $\hat{H}$ is the halfway vector
- $\hat{N}$ is the surface normal
- $\hat{V}$ is the camera view vector
- $\hat{L}$ is the light source vector

1.9.3. Surface Normals [1], [4]

In order to compute the surface normals we need the derivatives of the displacement(s), the values for which are obtained from the fourier transform above.

$$\vec{N} = \begin{bmatrix} -\frac{\frac{\delta h}{\delta x}}{1+\frac{\delta D_x}{\delta x}} \\ 1 \\ -\frac{\frac{\delta h}{\delta z}}{1+\frac{\delta D_z}{\delta z}} \end{bmatrix}$$

note that we need to normalise this for actual usage.

1.9.4. Subsurface Scattering [5], [9]

This is the phenomenon where some light absorbed by a material eventually re-exits and reaches the viewer. Modelling this realistically is impossible in a real time context (with my hardware). Specifically within the context of the ocean, we can approximate it particularly well as the majority of light is absorbed. An approximate formula taking into account geometric attenuation, a crude fresnel factor, lamberts cosine law, and an ambient light is used, alongside various artistic parameters to allow for adjustments. [9]

$$L_{\text{scatter}} = \frac{\Big(k_1 W_{\max}\langle \hat{L}, -\hat{V}\rangle^4 \Big(0.5 - 0.5\Big(\hat{L}\cdot\hat{N}\Big)\Big)^3 + k_2\langle \hat{V}, \hat{N}\rangle^2\Big) C_{\text{ss}} L_{\text{sun}}}{1 + \lambda_{\text{GGX}}}$$

$$L_{\text{scatter}} += k_3\langle \hat{L}, \hat{N}\rangle C_{\text{ss}} L_{\text{sun}} + k_4 P_f C_f L_{\text{sun}}$$

where
- $W_{\max}$ is the $\max(0, \text{wave height})$
- k_1, k_2, k_3, k_4 are artistic parameters
- C_{ss} is the water scatter color
- L_{sun} is the color of the sun
- C_f is the air bubbles color
- P_f is the density of air bubbles spread in water
- $\langle \omega_a, \omega_b\rangle$ is the $\max(0, (\omega_a \cdot \omega_b))$
- λ_{GGX} is the masking function defined under Smith's G_1

1.9.5. Fresnel Reflectance (Schlick's Approximation) [10], [11], [12], [13]

The fresnel factor is a multiplier that scales the amount of reflected light based on the viewing angle. The more grazing the angle the more light is refleceted.

$$F\Big(\hat{N}, \hat{V}\Big) = F_0 + (1 - F_0)\Big(1 - \hat{N}\cdot\hat{V}\Big)^5$$

where
- $F_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$
- n_1 & n_2 are the refractive indices of the two media [12]
- if using a microfacet model, replace $\hat{N}$ with the Halfway vector, $\hat{H}$) [12]

1.9.6. Blinn-Phong Specular Reflection [11], [13]

This is a simplistic, empirical model to determine the specular reflections of a material. It allows you to simulate isotropic surfaces with varying roughnesses whilst remaining very computationally efficient. The model uses "shininess" as an input parameter, whilst the standard to use roughness (due to how PBR models work). In order to account for this when wishing to increase roughness we decrease shininess.

$$L_{\text{specular}} = \Big(\hat{N}\cdot\hat{H}\Big)^S$$

$$\hat{H} = \hat{L} + \hat{V}$$

where
- S is the shininess of the material

1.9.7. Environment Reflections [10], [11]

In order to get the color of the reflection for a given pixel, we compute the reflected vector from the normal and view vector. We then sample the corresponding point on the skybox's cubemap and use that color as the reflected color. This method is somewhat simplistic, and not physically based.

$$\hat{R} = 2\hat{N}\left(\hat{N} \cdot \hat{V}\right) - \hat{V}$$

where
- $\hat{R}$ is the normalised vector that points to the point on the cubemap which we sample

1.9.8. Microfacet BRDF [5], [9], [13], [14]

This BRDF (Bidirectional Reflectance Distribution Function) is used to determine the specular reflectance of a sample. There are many methods of doing this - the one used here is derived from microfacet theory. D can be any distribution function - the geometric attenuation function G changing accordingly.

$$L_{\text{specular}} = L_{\text{sun}} f_{\text{microfacet}}\left(\hat{N}, \hat{H}, \hat{L}, \hat{V}\right) = \frac{F\left(\hat{N}, \hat{H}\right) G\left(\hat{L}, \hat{H}\right) D\left(\hat{N}, \hat{H}\right)}{4\left(\hat{N} \cdot \hat{L}\right)\left(\hat{N} \cdot \hat{V}\right)}$$

where
- L_{sun} is the color of the sun
- $F\left(\hat{N}, \hat{H}\right)$ is the Fresnel Reflectance
- $D\left(\hat{N}, \hat{H}\right)$ is the Distribution Function
- $G\left(\hat{L}, \hat{V}, \hat{H}\right)$ is the Geometric Attenuation Function

1.9.9. GGX Distribution [13], [14]

The distribution function used in the BRDF to model the proportion of microfacet normals aligned with the halfway vector. This is an improvement over the beckmann distribution due to the graph never reaching 0 and only tapering off at the extremes.

$$D_{\text{GGX}} = \frac{\alpha^2}{\pi\left((\alpha^2 - 1)\left(\hat{N} \cdot \hat{H}\right)^2 + 1\right)^2}$$

where
- $\alpha = \text{roughness}^2$

1.9.10. Geometric Attenuation Function (Smith's G_1 Function) [14]

Used to counteract the fresnel term, mimics the phenomena of masking & shadowing presented by the microfactets. The λ_{GGX} term changes depending on the distribution function used (GGX).

$$G_1 = \frac{1}{1 + a\lambda_{\text{GGX}}}$$

$$a = \frac{\hat{H} \cdot \hat{L}}{\alpha\sqrt{1 - \left(\hat{H} \cdot \hat{L}\right)^2}}$$

$$\lambda_{\text{GGX}} = \frac{-1 + \sqrt{1 + a^{-2}}}{2}$$

where

- $\alpha = \text{roughness}^2$

1.9.11. Distance Fog & Sun [10]

To hide the imperfect horizon line we use a distance fog attenuated based on height. In order to do this we use the depth buffer to determine the depth of each pixel and then based on that scale (lerp?) the light color to be closer to a defined fog color. Finally we blend a sun into the skybox based on the light position.

1.10. Prototyping

A prototype was made in order to test the technical stack and gain experience with graphics programming and managing shaders. I created a Halvorsen strange attractor [15], and then did some trigonometry to create a basic camera controller using Winit's event loop.

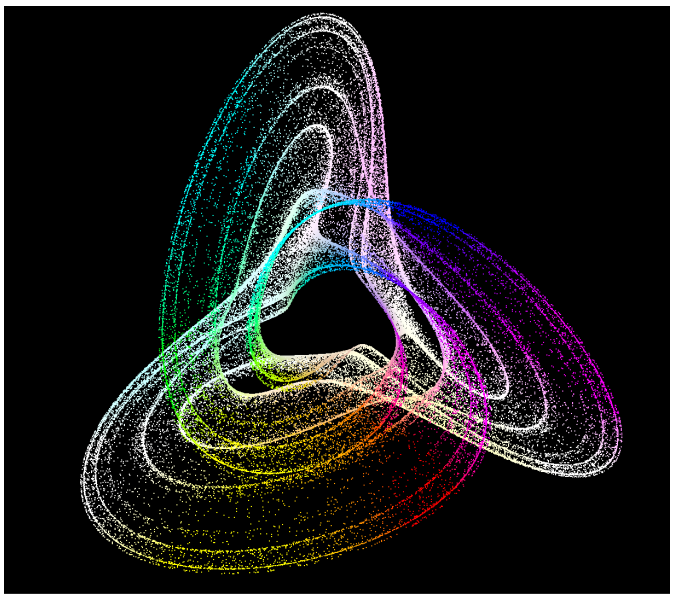

Figure 1: Found at https://github.com/CmrCrabs/chaotic-attractors

1.11. Project Considerations

The project will be split into 4 major stages - the simulation, implementing the IFFT, non PBR lighting, and PBR lighting. The simulation will most likely take the bulk of the project duration as implementing the spectrums, DFT and a GUI with just a graphics library is already a major undertaking. I will then implement the Blinn-Phong lighting model [11] in conjunction with the subsurface scattering seen in Atlas [9]. Beyond this I will implement full PBR lighting using a microfacet BRDF and statistical distribution functions in order to simulate surface microfacets.

1.12. Additional Features

If given enough time I would like to implement the following:

- Swell [2], the waves which have travelled out of their generating area [2].
- Further post processing effects, such as varying tonemapping options and a toggleable bloom pass
- A sky color simulation, as this would allow the complete simulation of a realistic day night cycle for any real world ocean condition.
- LEADR environment reflections, based on the paper by the same name (Linear Efficient Antialiased Displacement and Reflectance Mapping)

1.13. Project Objectives (Unfinished)

1 Scene
- 1.1 Language & Environment Setup
  - 1.1.1 setup all dependencies
  - 1.1.2 have development shell to ensure correct execution
  - 1.1.3 ensure compatability for all engines
- 1.2 Window & Compatability
  - 1.2.1 ensure compatability with windows, macos & wayland (& X11?) linux
  - 1.2.2 title & respects client side rendering of respective os
- 1.3 Data Structure
  - 1.3.1 talk abt shared data structures
  - 1.3.2 create struct for all variables
  - 1.3.3 camera struct etc
- 1.4 Render Pipeline
  - 1.4.1 list steps and that it works
- 1.5 Event Loop
  - 1.5.1 able to detect mouse movement for camera inputs
  - 1.5.2 able to detect mouse down for camera inputs
  - 1.5.3 escape to close
  - 1.5.4 resize
  - 1.5.5 redraw requested

2 Simulation
- 2.1 Startup
- 2.2 On Parameter Change
- 2.3 Every Frame
- 2.4 Optimisations
  - 2.4.1 dynamic render scaling stuff

3 Rendering
- 3.1 Lighting
  - 3.1.1 calculate light / view / halfway / normal vectors
  - 3.1.2 normalise all vectors
  - 3.1.3 fresnel
  - 3.1.4 subsurface scattering
  - 3.1.5 specular reflections
    - 3.1.5.1 blinn-phong
    - 3.1.5.2 pbr
      - 3.1.5.2.1 microfacet brdf
      - 3.1.5.2.2 distribution function
      - 3.1.5.2.3 geometric attenuation
  - 3.1.6 env reflections
    - 3.1.6.1 acerola
    - 3.1.6.2 LEADR
  - 3.1.7 lerp between this and foam

2. Bibliography

[1] Jerry Tessendorf, "Simulating Ocean Water." Accessed: Sep. 08, 2024. [Online]. Available: https://people.computing.clemson.edu/~jtessen/reports/papers_files/coursenotes2004.pdf

[2] Christopher J. Horvath, "Empirical directional wave spectra for computer graphics." Accessed: Oct. 20, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/2791261.2791267

[3] wikiwaves, "Ocean-wave Spectra." Accessed: Sep. 08, 2024. [Online]. Available: https://wikiwaves.org/Ocean-Wave_Spectra

[4] Jump Trajectory, *Ocean waves simulation with Fast Fourier transform*. Accessed: Sep. 13, 2024. [OnlineVideo]. Available: https://youtu.be/kGEqaX4Y4bQ

[5] Acerola, *I Tried Simulating The Entire Ocean*. Accessed: Sep. 08, 2024. [OnlineVideo]. Available: https://www.youtube.com/watch?v=yPfagLeUa7k

[6] "An introduction to Realistic Ocean Rendering through FFT - Fabio Suriano - Codemotion Rome 2017." Accessed: Sep. 25, 2024. [Online]. Available: https://www.slideshare.net/slideshow/an-introduction-to-realistic-ocean-rendering-through-fft-fabio-suriano-codemotion-rome-2017/74458025

[7] "Ocean simulation part one:using the discrete fourier Fourier transform." Accessed: Sep. 27, 2024. [Online]. Available: https://www.keithlantz.net/2011/10/ocean-simulation-part-one-using-the-discrete-fourier-transform/

[8] Wikipedia, "Normal Distribution." Accessed: Oct. 20, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Normal_distribution

[9] Mark Mihelich and Tim Tcheblokov, "Wakes, Explosions and Lighting:Interactive Water Simulation in Atlas." Accessed: Sep. 13, 2024. [Online]. Available: https://www.youtube.com/watch?v=Dqld965-Vv0

[10] Acerola, *How Games Fake Water*. Accessed: Sep. 13, 2024. [OnlineVideo]. Available: https://youtu.be/PH9q0HNBjT4

[11] Wikipedia, "Blinn–Phong reflection model." Accessed: Sep. 11, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Blinn-Phong_reflection_model

[12] Wikipedia, "Schlick's Approximation." Accessed: Sep. 10, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Schlick's_approximation

[13] Acerola, *The Secret Behind Photorealistic And Stylized Graphics*. Accessed: Sep. 17, 2024. [OnlineVideo]. Available: https://youtu.be/KkOkx0FiHDA

[14] Jakub Bokansky, "Crash Course in BRDF Implementation." Accessed: Sep. 17, 2024. [Online]. Available: https://boksajak.github.io/files/CrashCourseBRDF.pdf

[15] Dynamic Mathematics, "Strange Attractors." Accessed: Jun. 14, 2024. [Online]. Available: https://www.dynamicmath.xyz/strange-attractors/

[16] Saulius Vincevičius, "Realistic Ocean Simulation using Fourier Transform." Accessed: Oct. 20, 2024. [Online]. Available: https://github.com/Biebras/Ocean-Simulation-Report