NEA
Real-Time, Physically Based Ocean Simulation & Rendering

Zayaan Azam

Contents

1. Analysis

1.1. Prelude

// Fill Later

1.2. Client

1.2.1. 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.

1.2.2. 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 shader 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?"

1.2.2.1. skibidi

1.2.3. 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 game is intended to run on both x86 and arm64 devices
   4.2 see response 3.1
   4.3 the game is targeted towards mid to high end systems, however it would be ideal for the solution to be performant on lower end hardware

1.3. Research

1.3.1. Technologies

- 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

1.3.2. Simulation Algorithms & Formulae

Fast Fourier Transform (Cooley-Tukey) [1]

- Currently do not have the prerequisite math to properly understand this - waiting until ive learnt roots of unity
- this is where most of the complexity of the project comes from

JONSWAP (Joint North Sea Wave Observation Project) Spectrum [2], [3]

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

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

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

where

- α is the intensity of the spectra
- $\beta = \frac{5}{4}$, a "shape factor", rarely changed [3]
- $\gamma = 3.3$
- $\sigma = \begin{cases} 0.07 \text{ if } \omega \leq \omega_p \\ 0.09 \text{ if } \omega > \omega_p \end{cases}$ [2]
- ω is the wave frequency ($\frac{2\pi}{s}$) [3]
- ω_p is the peak wave frequency
- $\omega_p = 22\left(\frac{g^2}{U_{10} F}\right)^{\frac{1}{3}}$

- 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]
- g is gravity

Jacobian

- // Similar to the FFT, I need more math knowledge to properly understand how to do this - waiting until Ive completed all of matrices.
- need to find the jacobian determinant of the transform
- of water displacement vectors
- and then offset to bias negative results

Exponential Decay [4]

$$N(t) = N_0 e^{-\lambda t}$$

where

- N_0 is the initial quantity
- λ is the rate constant

1.3.3. Non-PBR Lighting Algorithms & Formulae

Rendering Equation [5]–[7]

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

where

- F is the fresnel reflectance
- L_{scatter} (atlas subs approx) (includes ambient)
- L_{specular} either atlas approx or blinn-phong
- $L_{\text{env\_reflected}}$ cubemap reflections per acerola or atlas
- multiply specular and env. reflections by fresnel

to include surface foam, *lerp* between the foam color and L_{eye} based on foam density. Increase the roughness in areas covered with foam for L_{specular}.

Subsurface Scattering [5]

$$L_{\text{scatter}} = \frac{\left(k_1 H \langle \omega_i \cdot -\omega_o \rangle^4 (0.5 - 0.5(\omega_i \cdot \omega_n))^3 + k_2 \langle \omega_o \cdot \omega_n \rangle^2\right) C_{\text{ss}} L_{\text{sun}}}{1 + \Lambda(\omega_i)}$$

$$L_{\text{scatter}} += k_3 \langle \omega_i \cdot w_n \rangle C_{\text{ss}} L_{\text{sun}} + k_4 P_f C_f L_{\text{sun}}$$

where

- H 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
- 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 $\max(0, \omega_a \cdot \omega_b)$
- ω_n is the normal

Blinn-Phong Specular Reflection [8]

$$L_{\text{specular}} = \vec{H} \cdot \vec{N}$$

$$\vec{H} = \frac{\vec{L} + \vec{V}}{\left|\vec{L} + \vec{V}\right|}$$

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

Fresnel Reflectance (Schlick's Approximation) [7]–[9]

$$F(\theta) = F_0 + (1 - F_0)\left(1 - \vec{N} \cdot \vec{V}\right)^5$$

where
- $F_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$
- θ is the angle between the incident light and the halfway vector [8]
- n_1 & n_2 are the refractive indices of the two media [9]
- $\vec{N}$ is the normal vector
- $\vec{V}$ is the view vector

Environment Reflections [7]

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

where
- $\vec{N}$ is the normal vector for the point
- $\vec{V}$ is the camera view vector
- $\vec{R}$ is the vector that points to the point on the cubemap which we sample

Distance Fog Post Processing (Unfinished) [7]
- hides issues with fresnel at grazing angles [5]

Atmospheric Scattering (Unfinished) [7]
- attenuate distance fog based on height

General Effects (Unfinished) [7]
- additively blend a sun with skybox

- apply a bloom pass
- cinematic tone mapping

1.3.4. PBR Lighting Algorithms / Formulae

Microfacet BRDF [5]

$$f_{\text{microfacet}} = \frac{F(\omega_i, h)G(\omega_i, \omega_o, h)D(h)}{4(n \cdot \omega_i)(n \cdot \omega_o)}$$

where

- $F(\omega_i, h)$ is the Fresnel Reflectance
- $D(h)$ is the Distribution Function
- $G(\omega_i, \omega_o, h)$ is the Geometric Attenuation

Beckmann Distribution [5], [10]

$$k_s = \frac{\exp\left(\frac{-\tan^2 \alpha}{m}\right)}{\pi m^2 \cos^4 \alpha}$$

where

- $\alpha = \arccos(N \cdot H)$
- m is the RMS slope of the surface microfacets

Geometric Attenuation Function, Smith GGX (Unfinished)

-

Specular Reflection [5]

$$L_{\text{specular}} = \frac{L_{\text{sun}} F(\omega_h, \omega_{\text{sun}}) p_{22}(\omega_h)}{4(\omega_n \cdot \omega_{\text{eye}})(1 + \Lambda(\omega_{\text{sun}})) + \Lambda(\omega_{\text{eye}})}$$

where

- $\omega_{\text{sun}}, \omega_{\text{eye}}, \omega_h$ is the sun / eye / half vector direction
- ω_n is the macronormal, in this case $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

1.3.5. Prototyping

A project was undertook in order to test the technical stack and gain experience with graphics programming and managing shaders. I created a Halvorsen strange attractor [11], and then did some trigonometry to create a basic camera controller using Winit's event loop.

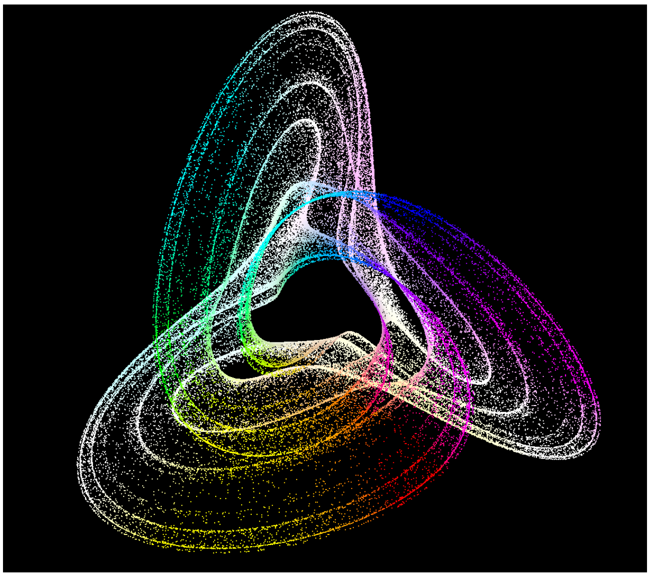

Figure 1: Found at https://github.com/CmrCrabs/chaotic-attractors

1.3.6. Project Considerations

- talk abt pbr complexities in each part
- complexitites of distribution functions
- microfacet theory
- (so) using blinn phong
- if time allows will also use pbr cubemap reflection sampling

1.4. Objectives

2. Bibliography

[1] Wikipedia, “Fast Fourier Transform.” Accessed: Sep. 08, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Fast_fourier_transform

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

[3] CodeCogs, “The JONSWAP spectra in the wave-frequency domain.” Accessed: Sep. 13, 2024. [Online]. Available: https://www.codecogs.com/library/engineering/fluid_mechanics/waves/spectra/jonswap.php

[4] Wikipedia, “Exponential Decay.” Accessed: Sep. 11, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Exponential_decay

[5] 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

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

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

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

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

[10] Wikipedia, “Specular Highlight.” Accessed: Sep. 11, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Specular_highlight

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

[12] Wikipedia, “Fourier Transform.” Accessed: Sep. 08, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Fourier_transform

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

[14] Fabio Suriano, “An introduction to realistic ocean rendering through FFT.” Accessed: Sep. 08, 2024. [Online]. Available: https://www.slideshare.net/slideshow/an-introduction-to-realistic-ocean-rendering-through-fft-fabio-suriano-codemotion-rome-2017/74458025#1

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

[16] Mark Mihelich and Tim Tcheblokov, “Wakes, Explosions and Lighting:Interactive Water Simulation in Atlas.” Accessed: Sep. 13, 2024. [Online]. Available: https://gpuopen.com/gdc-presentations/2019/gdc-2019-agtd6-interactive-water-simulation-in-atlas.pdf

[17] siggraph, “The Technical Art of Sea of Thieves.” Accessed: Sep. 14, 2024. [Online]. Available: https://history.siggraph.org/learning/the-technical-art-of-sea-of-thieves/