NEA
Real-Time Physically Based Ocean Simulation

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?"

1.2.3. 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. 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

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

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

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

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

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

Fresnel Specular Reflection (Schlick's Approximation) [4], [5]

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

where

- $R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$
- θ is the angle between the incident light and the halfway vector [5]
- n_1 & n_2 are the refractive indices of the two media [4]

Beckmann Distribution [6]

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

where

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

Microfacet BRDF

-

(Approximate) Subsurface Scattering (Atlas Paper)

-

Distance fog post processing

-

attenuate distance fog based based on height

-

sample hdri skybox for reflections, multiple yiwth schlicks

-

jacobian

-

exponential decay

-

Asynchronous GPU Readback

-

1.3.3. 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 [7], and then did some trigonometry to create a basic camera controller.

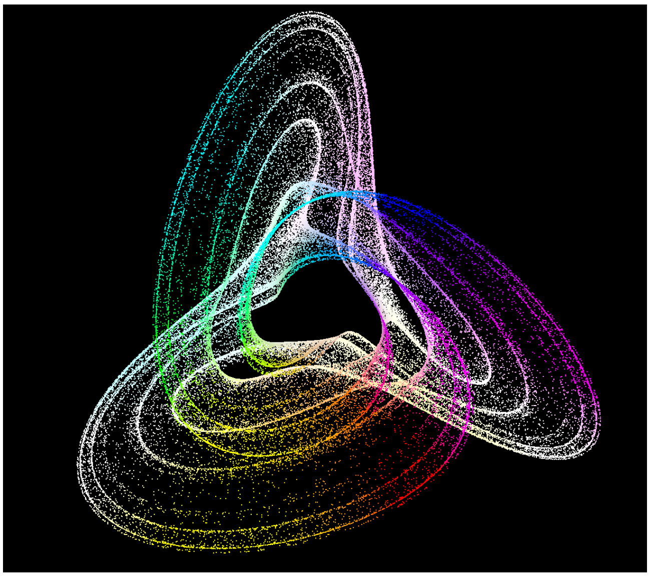

Figure 1: Found at https://github.com/CmrCrabs/chaotic-attractors

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

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

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

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

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

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

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

[11] 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

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

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

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