# Sailing Simulator:
## COS426 Final Project Intermediary Report

**Caio Costa**
Princeton University
ccosta@princeton.edu

### Abstract

In this project, we present a physically-motivated simulation of an ocean environment. The user controls the motion of a sailing raft as it is propelled by virtual wind through the procedurally-generated scenery. The ocean is simulated by overlappping trains of Gerstner/Rankine waves.

# 1 Introduction

Our goals in creating this simulation were two-fold. First and foremost, we envisioned an aesthetic and polished final product, with the capability to deliver beautiful ocean vistas. The second was to produce an amusing and interactive sailing simulator. Lastly, we sought an, if not totally physically accurate, at least physically plausible model of the ocean and sailing mechanics. To this end, where needed, we prioritized the look of the project over the feel, and the feel over the physics.

# 2 Related Work

## 2.1 Ocean Simulation

Our oceanographic model draws on a long line of both oceanography and computer graphics. The paper we based the waves off of was the seminal paper on ocean simulation, [2], which is in turn based off of a $19^{\text{th}}$ century oceanographic model arrived at by both Gerstner and Rankine independently. The model is based on a *troichoid*, the general form of a cycloid. This geographical shape is in essence a modified sinusoidal wave, which can intuitively be thought of as the trace of a point some distance from the center of a wheel as it rotates along a flat surface. Fournier and Reeves make several modification to the basic trochoid. They orient them in a specified direction, increase the phase angle at the peaks of the waves to model crests, modify the wavelength as the seafloor depth changes, and flatten the trochoidal circles into ellipses as the waves approach the shore to model breaking.

They bundle these trochoids into regular rectangles which they coin "wave trains". Each wave train contains waves moving in the same direction and speed, and moves along a bigger ocean surface. As they move along the ocean, each train accumulates cumulative depth information used to approximate an integral and calculate the water wave refraction effect. The wave trains have a few tricks to break up the regularity of the waves, including adding subtractive inverted Gaussian "holes" according to a Poisson disk distribution and using amplitude functions perpendicular and parallel to the wave direction to decay the waves at the boundaries of the wave train rectangle and add randomness to the heights of the waves in each train. They also modulate the wavelengths to further differentiate the waves.

They further describe a particle system for adding foam and spray to cresting waves, and go over some details of their rendering and lighting process. Of course, as the paper was written some 35 years ago, their system was designed for offline rendering rather than interactive. Specifically, they generate what is essentially a point cloud representing the ocean's parametric surface, and use Catmull-Rom patches to interpolate the points into a smooth surface. At the time, this took some 10 hours and 21 minutes to produce a single $2048 \times 1228$ image. Interactive versions of Gerstner waves

started appearing around the turn of the millenium, including in the helpful SIGGRAPH course notes from [3] and in Nvidia's GPU Gems 1 [1].

There of course exist alternative models of the ocean. The major alternatives seem to be a height or displacement map based approach, or a full-fledged fluid dynamics simulation, which accounts for the 3D volume of water, at the cost of need to solve relatively complex differential equations and Navier-Stokes systems. We opted to pursue the Gerstner simulation for a few reasons. First, the waves displace the vertices in such a way as to concentrate them near the peaks, allowing for a higher LOD at the points of visual interest with the same number of vertices. Second, they allow for crescents in the wave crests, which a height map approach would not, since each position in the horizontal plane could only have a single vertex above or below it. Finally, it seemed that the Gerstner waves would be fairly adaptable and modifiable, allowing for both large rolling storm waves and smaller breakers near shore.

### 2.2 Terrain Generation

Terrain generation is a well studied area of computer graphics, with models ranging from hand crafted algorithms to fault line based generators to neural generative adversarial networks. We chose to follow the first option in creating our own unique island generation algorithm, described in more detail below.

## 3 Methods

Our project is primarily built using the Three.js library, in conjunction with Node.js, and hosted on GitHub Pages.

### 3.1 Ocean Simulation

The ocean simulation, as mentioned above, is based heavily on the work of [2], utilizing the course notes and interpretations of [3] and [1]. Much like the ocean itself, our implementation has been in constant flux.

The wave surface itself is implemented use a PlaneBufferGeometry. At each update time step, we loop over the vertices in the geometry, and over all the wave trains, adding up the displacements from each train, and moving the vertex to the appropriate final position.

#### 3.1.1 Wave Trains

At present, the base unit of the ocean is the wave train. Each wave train has $X$ and $Z$ coordinates relative to the water surface. We define a local coordinate system within each train, which is used to calculate the trochoidal wave shapes. This coordinate system is oriented in the direction the wave train is travelling (i.e. perpendicular to the wave crests). Beyond this, the trains also carry subtractive information about holes in the wave crests, and a cumulative depth grid, which accumulates depth information as the train travels across the ocean. The wave parameters are stored in the train as well, so that each train can have differing wave characteristics. Lastly, each train calculates its own time, to account for speeding up or slowing down as the wind changes.

#### 3.1.2 Wave Parameters

- Steepness (User-defined): This parameter controls whether waves come to a sharp point or rolling tops. As a generalization, at low steepness, waves are approximately sinusoidal, while at high steepness, waves are approximately cycloidal.
- Median Wavelength (User-defined - Stochastic): New wave trains generated will have a wavelength between half and twice the defined median value.
- Median Amplitude (User-defined - Stochastic): Once the wavelength is (randomly) determined, the train will have an amplitude in the same ratio as the median wavelength and median amplitude.
- Lambda (User-defined): This parameter scales the amount the waves crest (i.e. the factor by which the height of waves increments the phase angle).

- Train Height/Width (User-defined - Stochastic): The user can define a minimum and maximum size. Newly generated wave trains will pick their dimensions uniformly at random from the specified range.
- Wave Height Scaling (User-defined): A scaling factor applied to the $y$ values of the waves.
- Wave Height Frequency (User-defined): The frequency of the Perlin noise component multiplied with the amplitude envelopes of the local forward dimension. Higher values introduce more randomness to the heights of waves.

## 3.2 Lighting

We use a combination of different lights and materials to illuminate our scene. The islands and boat use MeshToonMaterials with a small gradient map for a cartoon effect. The ocean mesh itself uses the physically-based rendering (PBR) material made available through Three.js, along with a bump map for small textures, and an environment map of the sky for reflections. The ocean material is transmissive and uses a clear coat for a water-like effect.

The sun is modelled as a far-away directional shadow-casting light. However, this makes it computationally expensive to compute accurate boat shadows. To solve this, we also include a second directional light with a much smaller area and proportionally lower intensity at the sun's position, which handles the raft's shadows.

Finally, we also include hemisphere and a dim ambient light for more complete lighting effects.

## 3.3 Island Generation

Island generation is handled on a $3 \times 3$ chunk system. Once the center of the scene leaves the center chunk, the far three chunks get moved to the the other side. In this way, the center chunk is always positioned near to the origin, minimizing floating point issues.

The land is also modelled using PlaneBufferGeometrry. The seabed is generated using low frequency fractional Brownian motion. The parameters (octaves, frequency, amplitude, etc) are exposed to the user, allowing for experimentation on the fly. The islands are then added on using a 2D Gaussian distribution. The size and number of Gaussians added per chunk are determined by the user. The effect of the Gaussian is attenuated using the smoothstep function as it reaches the borders of the chunk, to prevent islands from crossing chunks. This can cause steep cliff-like terrain, but prevents discontinuities at the borders, since islands are stored on a per-chunk basis. Land vertices above a certain $y$ value have an additional higher frequency fractional Brownian motion added on to add texture to the islands, which can also be controlled by the user. As yet, there are not trees or color on the islands, but we hope to include these in the final product.

## 3.4 Sailboat

### 3.4.1 Cloth Physics

The flashiest part is the moving cloth. The physics and basic mechanics of the sail cloth were adapted directly from our Assignment 5 code. We adapted these to an ES6 style class, and further refined the material and position of the cloth to suit our needs.

### 3.4.2 Motion and Controls

To give the illusion of motion, the world translates around the center model of the boat, which we designed in Tinkercad. As the boat moves, the islands shift around it using the previously explained chunk system. The waves also translate, with the bump map texture translating so that the small details move with the boat. The boat motion is calculated based on the sum total force applied to the sail particles in the horizontal plane. Gravity and buoyancy are not modeled directly on the boat, with the boat simply animating to a floating position on the surface of the water plane as wave pass by.

### 3.5 Stylization

We hoped to have a non-photorealistic style. To do so, we used constrained gradient palettes on most of the materials, including the skybox and islands. The ocean transparency was more photorealistic (as well as performance intensive) but was offset with a relatively lower resolution bump map and stylized reflections to try and maintain the toon style. To give everything a stylized final coat, we apply an OutlineEffect to simulate cel shading.

### 3.6 Result

Here we present a sample image of the project as it currently stands.

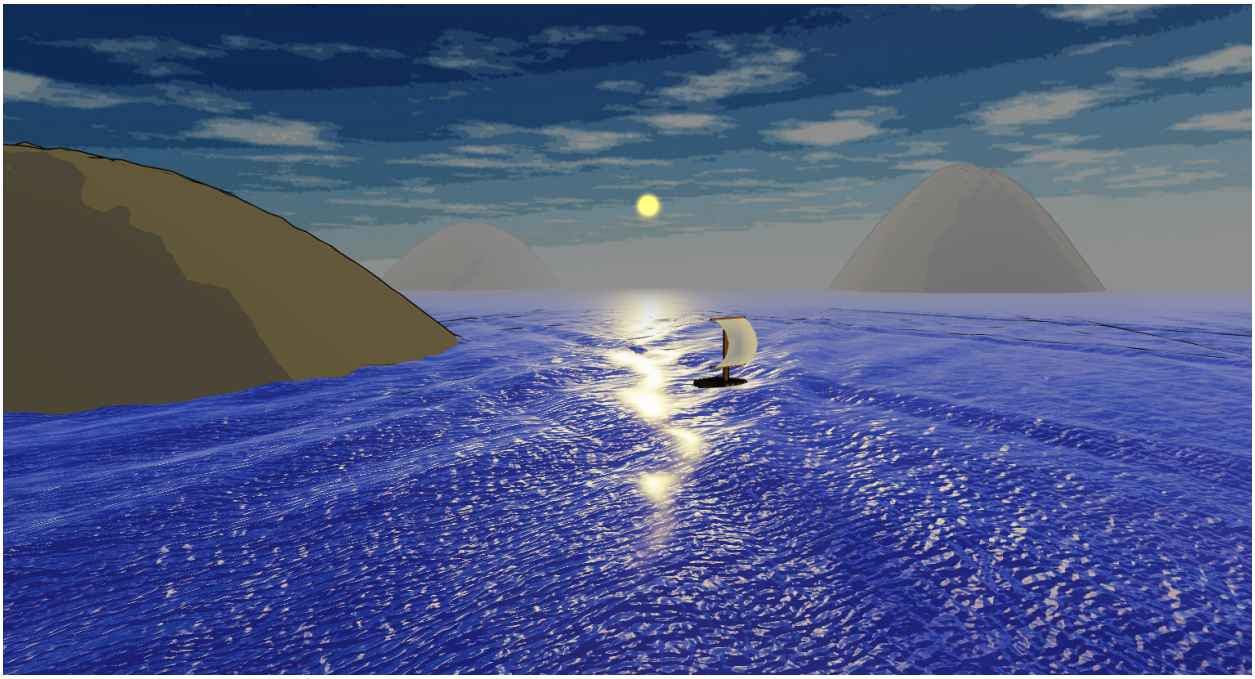

Figure 1: Sail boat with ocean simulation. Note the textured tops of the near and far islands, as well as the filled sail cloth. Ocean simulated by three overlapping wave trains.

## 4 Future Work

There are still several areas we hope to improve on. The boat motion is not yet directly controlled by the user. In the final product, we hope to have either arrow key or mouse-based control. In both cases, we hope to have a rudder for visual feedback as the boat direction changes. Also, there is still further fine-tuning needed for the boat force calculations, as well as collisions with islands.

We also hope to spend some time adding performance toggles, such as downgrading transparency materials, perhaps simplifying the wave calculations, or reducing the size of the ocean patch simulated. Even on a powerful discrete laptop GPU (1070M), we only push out 35 frames per second.

#### Acknowledgments

Thanks to friends and family for giving feedback on the visuals and the realism of the sailing mechanics.

### Honor Code

This paper represents my own work in accordance with University regulations.

## 5 Contributions

As I worked solo, the entirety of the project code and writeup was done by me.

## References


[1] Randima Fernando et al. *GPU gems: programming techniques, tips, and tricks for real-time graphics*, volume 590. Addison-Wesley Reading, 2004.

[2] Alain Fournier and William T Reeves. A simple model of ocean waves. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, 1986.

[3] Jerry Tessendorf et al. Simulating ocean water. *Simulating nature: realistic and interactive techniques. SIGGRAPH*, 1(2):5, 2001.