Project Freedom – Making an Ocean Shader

By Robert Kilkenny and Manu Kolehmainen

The basis of this project was to create a working simulation that allows a user to observe an ocean shader in an empty scene to display different properties of shader matrices. The equations used in this project were sourced from research papers and games with competent ocean graphics (specifically using resources from the games *Sea of Thieves* and *Atlas* and information from NVIDIA). To accomplish this, we needed to utilize the GPU of computers to handle many similar computations in parallel and optimize the computations we assign to the model to ensure that the simulation's performance does not falter. However, before the simulation could be started, we had to make sure OpenGL was able to render shaders in the application by transforming the water surface's vertices into the basis of the camera by using a series of transformation matrices[1].

The first type of wave simulation we used for the ocean was a Sum of Sines approximation: $\sum_0^n e^{\sin(x)-1}$. While a Gerstner wave (also known as a Trochoidal wave)[2] could have also been used, these waves are complex, and since we wanted to offer the ability to alter values in the simulation. It was likely the user could insert values to break the simulation and make unnatural waves, since Gerstner waves can curl easily when changing the amplitude of the function. Also, this function has an easily computed partial derivative, which was used for expressing the normal of the mesh which is required for lighting and reflections.

By adding smaller features in conjunction with these wave simulations, such as specular reflections with a skybox, we could make the waves look much more realistic akin to how the waves behave in AAA games. As such, we decided to implement a skybox into the environment so that if we were able to implement dynamic reflections into the simulation, we could utilize it to create much more accurate-looking waves as they would react to the skybox. We imported several different skyboxes so we would have the ability to test the effects of the skybox on the reflections from the ocean shader. Unfortunately, we eventually phased out reflections out of the sprint for this project in favor of working on a better system for simulating waves and hopefully being able to use complex numbers to better reflect how ocean waves flow.

Finally, we wanted to enable the code to be able to apply complex wave equations, so we wanted to implement the Discrete Fourier transform (DFT). By using a Fourier transform, the equations used to simulate the waves could be much more complex as we could apply easier patterns found in the frequency domain and then use it to translate them into the time domain for the simulation.[3] This could be done more optimally by saving the results of the Fourier transform into a render texture, but we left that functionality for future work. We decided to improve the system by updating it from using DFT to Fast Fourier transforms (FFT). The difference between these two is that FFT runs with an O(n*log(n)) while DFT is $O(n^2)$. While we were able to implement DFT and FFT to simulate the waves, we did not have enough time to find realistic spectrum for the transforms to operate on, leaving the waves unnecessarily sharp. For future work, implementing the JONSWAP[4] spectrum could produce highly realistic results.

[1] https://learnopengl.com/Getting-started/Coordinate-Systems
[2] https://en.wikipedia.org/wiki/Trochoidal_wave
[3] https://www.keithlantz.net/2011/10/ocean-simulation-part-one-using-the-discrete-fourier-transform/
[4] https://www.orcina.com/webhelp/OrcaFlex/Content/html/Waves,Wavespectra.htm