Technical Documentation

CSCI711 Final Project

By:

Noya Cai

Tsingtao Zhang

# Program Description

This ocean simulation program renders an ocean wave in real-time and interactively\*, with the feature of whitted ray tracing, procedurally generated waves and a fast caustics effect.

The user can click on the ocean surface to create the ripple effect.

The whitted ray tracing is developed throughout the checkpoints in CSCI 711.

The procedurally generated waves are constructed by a series of [Gerstner waves](https://www.tandfonline.com/doi/pdf/10.2991/jnmp.2008.15.S2.7#:~:text=Gerstner%E2%80%99s%20wave%20is%20given%20by%20emb,x%20%3D%20a%20%2B%20sin%20m%28a).

The fast caustics effect is created in real time and inspired by [Evan Wallace](https://madebyevan.com/)’s [Rendering Realtime Caustics in WebGL](https://medium.com/@evanwallace/rendering-realtime-caustics-in-webgl-2a99a29a0b2c).

\*The interactive part is done in the course IGME 740 as a final project.

# Historical Development of Program and Current Status

[Noya Cai](mailto:qc6338@g.rit.edu), contributed the wave function and scene layout.

[Tsingtao Zhang](mailto:qz3017@g.rit.edu), contributed the caustics effect.

## Modules of the project:

The ray tracing framework.

The scene layout (Primitives defined in fragment shader).

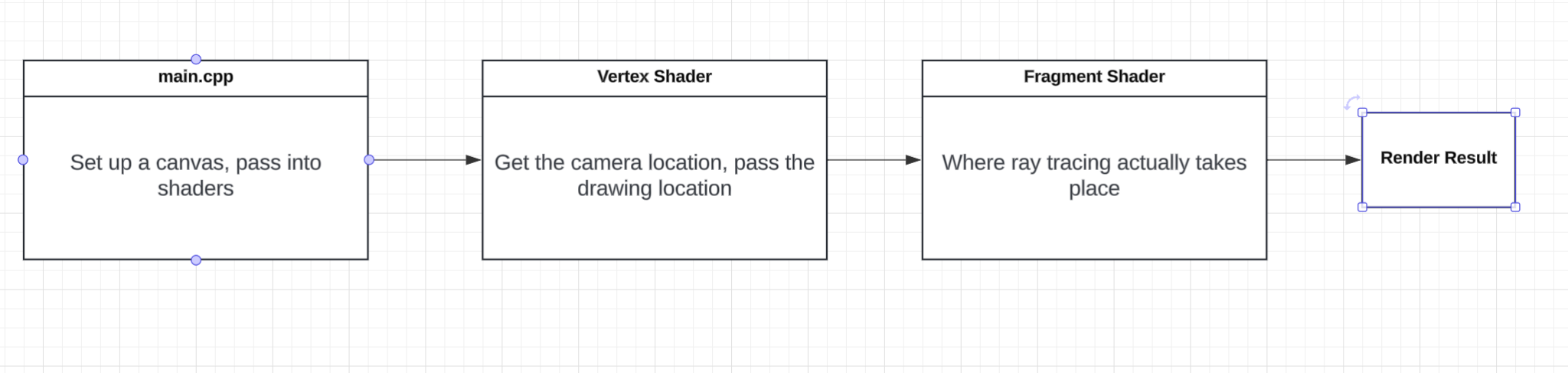
The Gerstner wave function (directional and ripple).

The caustics effect (coefficient on specular and diffuse shading)

## Testing criteria

Windows x64 platform, with integrated graphics card, over 22 GB of RAM.

# Overall System Structure



The program consists of three major files: main.cpp, Water.vert, and Water.frag. We set up the environment in Microsoft Visual Studio (x64 Debugging platform).

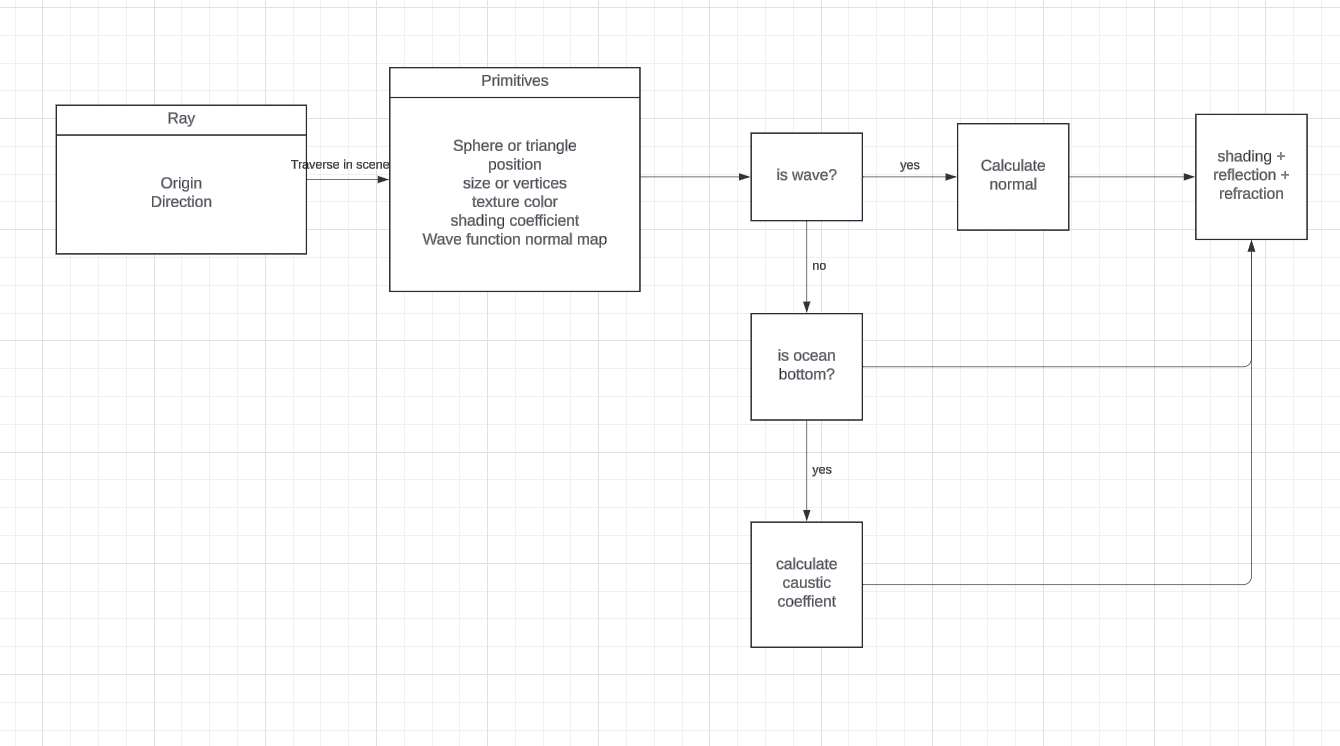
The main.cpp file is where we set up the OpenGL environment, and pass in the parameters into the shaders. The reason we use OpenGL and shaders is that we want to exploit the GPU’s parallelism in order to achieve the real time rendering.

However, we didn’t take the standard modern approach (the approach introduced by [LearnOpenGL](https://learnopengl.com/Introduction)) which pass in the mesh’s data to shaders and use rasterization to do shading, because in this way the shaders don’t have the global memory (or shared memory) to do the ray tracing. Instead, we only set up the OpenGL environment, and define two triangles that cover up the whole screen, and set up the vertex and fragment shaders for only these two triangles. Consider these two triangles as a canvas (similar to [using compute shaders for rendering](https://learnopengl.com/Guest-Articles/2022/Compute-Shaders/Introduction)). And we only use vertex shader to pass in the drawing location (consider this as how we let the fragment shader know its pixel position).

Now we finally reach where things really matter. Inside the fragment shader, we define every primitives (spheres and triangle) in the scene, the light sources, we initiate the ray tracing process, and we construct our wave and caustics effect.

The user can open the RayTracer.sln and run the program, keep in mind, though, that this program requires over 22GB of RAM, so most modern standalone GPU won’t be able to compile, so we need to use integrated graphics.

# Overall Program Structure

**

Whitted ray tracing: The ray traverses through the scene, and intersecting with primitives, returns the shading color. If the primitive is reflective or (and) reflective, spawn reflect or (and) refract ray accordingly.

And the Gertner wave function is changing the primitive’s normal, which will result in different shading; the caustics effect is calculated based on the area ratio of projected triangles (will explain more below).

# Description of Key Data Structures

* Ray (vec3 origin, vec3 direction)
  + Essential part in ray tracing, intersect primitives and return shading color.
  + Also used for calculating caustics.
  + Initiated from camera position, can be spawned if the intersecting primitive has reflection / refraction.
* Primitive - Triangle (3 \* vec3 vertices position, vec3 texture color, float shading coefficients (ka, kd, ks, kr, kt), bool isWave identifier
  + Primitive for ground and water surface.
  + If applied to a wave function, its normal will change based on time and phase.
  + Intersected by the ray and it returns the normal, shading coefficients and other parameters.
  + If reflection or (and) refraction (kr, kt) is larger than 0, it will initiate a new ray from the intersecting point.
* Primitive - Sphere (3 \* vec3 vertices position, vec3 texture color, float shading coefficients (ka, kd, ks, kr, kt)
  + Primitive for ground and water surface.
  + Intersected by the ray and it returns the normal, shading coefficients and other parameters.
  + If reflection or (and) refraction (kr, kt) is larger than 0, it will initiate a new ray from the intersecting point.
* Light (vec3 position, vec3 color)
  + The light source for Phong shading.
  + Light color can influence the shading color.

# Testing/Acceptance Criteria

