CS 557 Computer Graphics Shaders

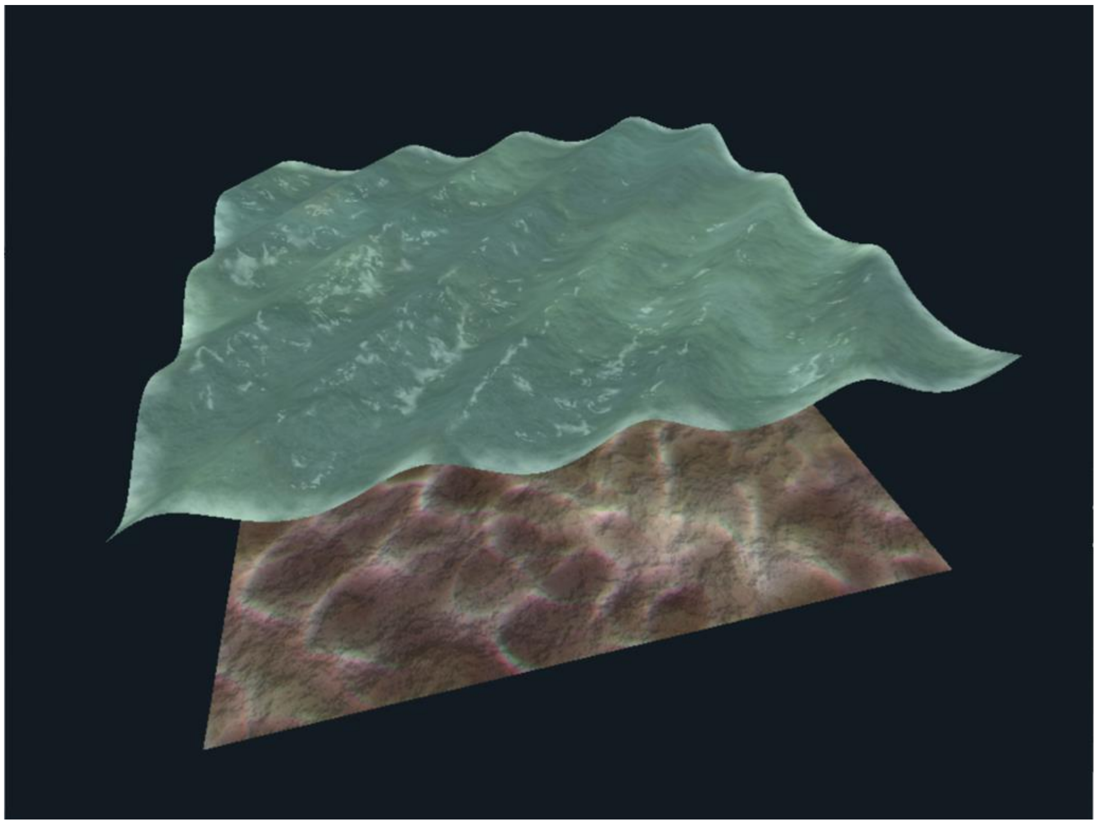
Final Project

Realistic Water Simulation Shader

Name: Rachel Xing (#934229189)

Email Address: xingru@oregonstate.edu

Video Link: https://youtu.be/W7H_So7CtAM



Description:

In this final project, I implemented a water shader to simulate both the water surface and the underwater floor. In addition to simulating water waves using mathematical models, a caustic effect and foam effect are added to enhance the realism of the rendering. It runs with "**water.glib**" using glman.

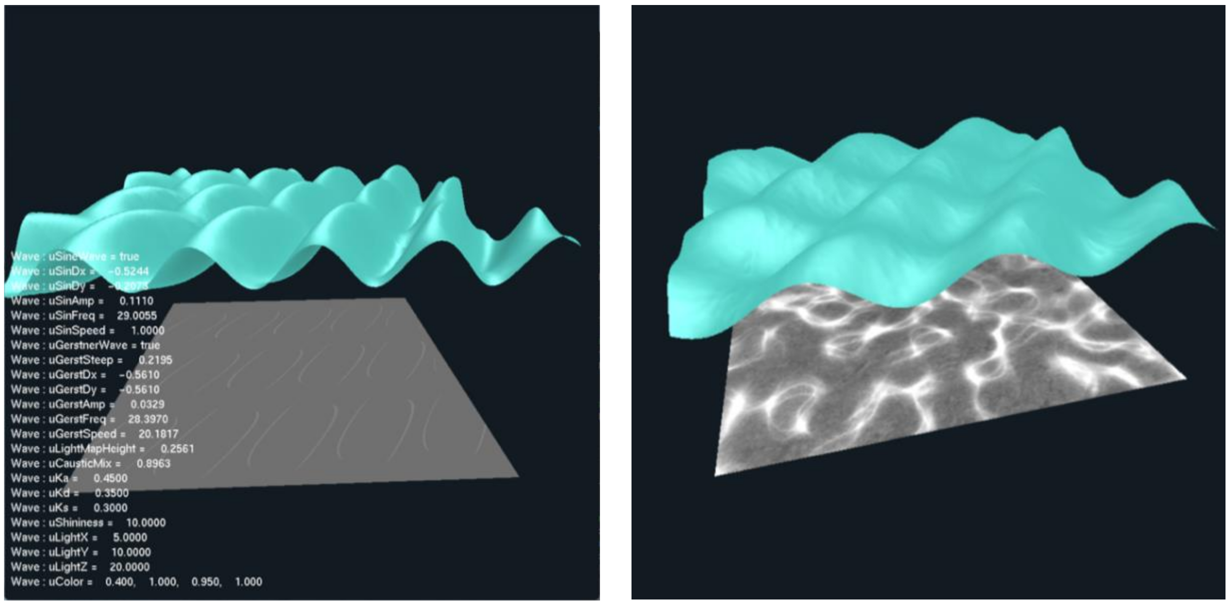
In this shader, there are two XZ quads: the above one is the water surface and the below one is the underwater floor. The implementation consists of a vertex and fragment shader, but the proposed geometry shader was not implemented. I originally thought that the 2D mesh for the water surface was further processed in the geometry shader. However, after re-reading my referenced article, I learned that the 2D mesh was actually created before the vertex shader in the pipeline, which means that the geometry can be generated directly in glib. Also, the geometry shader is only needed for normal calculation in the article, which can also be handled within the vertex shader. Thus, I abandoned the geometry shader in the final implementation. The stretch goal was not completed due to time constraints before the deadline. Also, I found that integrating this water shader into a terrain shader in glib proved to be more complicated than I expected. However, I plan to explore this further during spring break when I have more time.

According to *GPU Gems* Chapter 1, the sine wave is suitable for calm, small ponds, while the Gerstner wave is better for large, rough seas that can generate sharp waves. Therefore, I chose to include both a sine wave model and a Gerstner wave model in my project to create more realistic waves. The user can select which wave model(s) to apply and adjust the parameters for each wave. The wave computation is done in the vertex shader, where each vertex of the water surface is displaced based on the wave functions, and its associated normal is re-computed accordingly. The sine wave and Gerstner wave models are referenced from *GPU Gems* Chapter 1, and the normal is calculated as the cross product of the binormal and tangent of the displaced vertex. The following wave parameters can be adjusted in the glman interface:

Parameter	**Description**
uSineWave	Whether a sine wave is added
uSinDx	Control the x direction of the sine wave
uSinDz	Control the z direction of the sine wave
uSinAmp	Control the amplitude of the sine wave
uSinFreq	Control the frequency of the sine wave
uSinSpeed	Control the speed of the sine wave
uGerstnerWave	Whether a Gerstner wave is added
uGerstSteep	Controls the steepness of the Gerstner wave. When set to 0, the wave becomes a sine wave.
uGerstDx	Control the x direction of the Gerstner wave
uGerstDz	Control the z direction of the Gerstner wave
uGerstAmp	Control the amplitude of the Gerstner wave
uGerstFreq	Control the frequency of the Gerstner wave
uGerstSpeed	Control the speed of the Gerstner wave

All the visual effects of the water are computed in the fragment shader. For the simulation of water caustics, I initially planned to follow the backward ray tracing method described in *GPU*

Gems Chapter 2. This method assumes an area light source above the water surface. From each fragment, a vertical ray is sent upward, and its refraction from water to air through the water surface is calculated using Snell's Law. If the refracted ray intersects the light source, the fragment contributes to the caustic and is rendered in white. I implemented a shader based on this approach, which can be run using "wave.glib" in glman. However, this method can only produce hard-edged caustics that are similar to the examples in the book, and the caustics are not obvious when the wave amplitude is low.



```
Wave : uSineWave = true
Wave : uSinDx =    -0.5244
Wave : uSinDy =    -0.2075
Wave : uSinAmp =    0.1110
Wave : uSinFreq =   29.0055
Wave : uSinSpeed =    1.0000
Wave : uGerstnerWave = true
Wave : uGerstSteep =    0.2195
Wave : uGerstDx =   -0.5610
Wave : uGerstDy =   -0.5610
Wave : uGerstAmp =    0.0329
Wave : uGerstFreq =   28.3970
Wave : uGerstSpeed =   20.1817
Wave : uLightMapHeight =    0.2561
Wave : uCausticMix =    0.8963
Wave : uKa =    0.4500
Wave : uKd =    0.3500
Wave : uKs =    0.3000
Wave : uShininess =   10.0000
Wave : uLightX =    5.0000
Wave : uLightY =   10.0000
Wave : uLightZ =   20.0000
Wave : uColor =   0.400,   1.000,   0.950,   1.000
```

Figure 1. Water caustic rendering using (left) the backward tracing method and (right) texture mapping.

Then, I discovered a more practical approach in this blog titled "Rendering realtime caustics". It uses a caustic texture map and applies texture mapping to the target object to simulate a caustic effect. I implemented this approach in my water simulation shader, and as shown in Figure 1, the resulting caustic are much more natural compared to the backward tracing method. To make the caustic look more real, I made the caustic texture sampling dynamic by offsetting the sampler based on vertex displacement. This allows the caustics to move naturally with the waves. The shader uses different caustics for the sine wave and the Gerstner wave. This ensures that the overall caustic motion aligns with the interaction of multiple waves. In addition, I applied a chromatic aberration technique, which separates the RGB channels of the sampled texel and offsets them relative to each other to simulate refraction for a more realistic caustic effect.

The following parameters for the caustic effect can be adjusted in the glman interface:

Parameter	**Description**
uCausticVisible	Adjust the visibility of caustics on the water floor
uCausticWave	Control the influence weight of wave motion on caustic sampling
uCausticOffset	Adjust the offset between caustics for the sine wave and the Gerstner wave

uChromaticOffset	Adjust the offset for the chromatic aberration effect in the caustics

For both the water surface and the water floor textures, texture mapping and normal mapping are applied. The water floor is sampled from a rock texture, with a normal map used to simulate detailed surface features to add realism. For the water surface, a blend of turquoise color and the water floor color is used to simulate light refraction, and a tileable normal map is applied to create irregular water surface details. Here are adjustable parameters for the water surface appearance:

Parameter	**Description**
uWaterTiling	Control the tiling of the normal map on the water surface
uWaterRefractMix	Control the mix between the water floor texture and the solid color

Lastly, the user can toggle foam generation along the boundaries of the water surface by **uShowFoam**. This method follows the approach described in the blog titled "Water Breakdown", where a foam texture with random red and blue circles on a black background is used. The texture is sampled twice, once for the red channel and once for the blue channel, then the results are added together and subtracted from 1.0 to determine the weight of white. The water surface color is then blended with the resulting white color to generate the foam. The foam gradually fades from the periphery to the center. The foam intensity is adaptive to the wave amplitude, and the foam is sampled dynamically based on wave speed.

Per-fragment lighting is implemented in the fragment shader, as in previous projects, with adjustable parameters (**uKa, uKd, uKs, uShininess, uLightX, uLightY, uLightZ**) that can be modified through the glman user interface. Besides, the parameter **uColor** allows users to change the color of water using glman's color palette.

Screenshots:

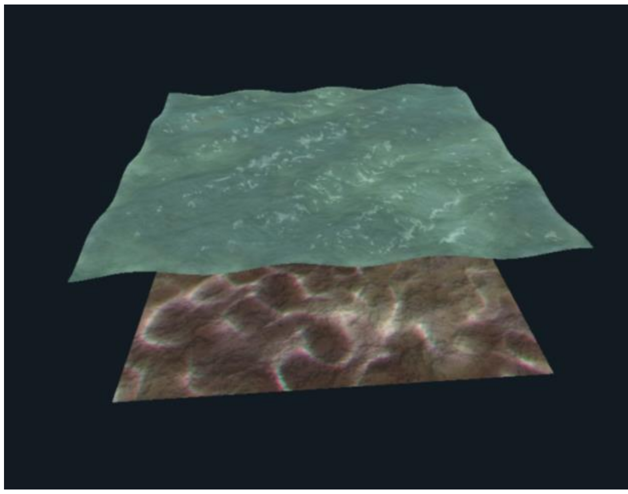


Figure 2. Simulation of a calm lake using only the sine wave

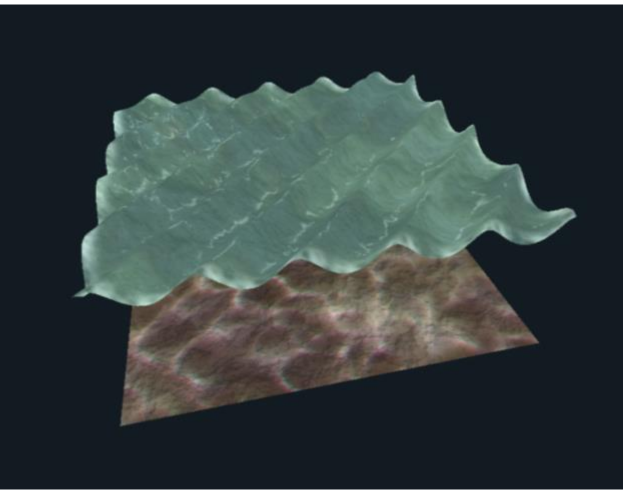


Figure 3. Simulation of a rough sea using a combination of a sine wave and a Gerstner wave

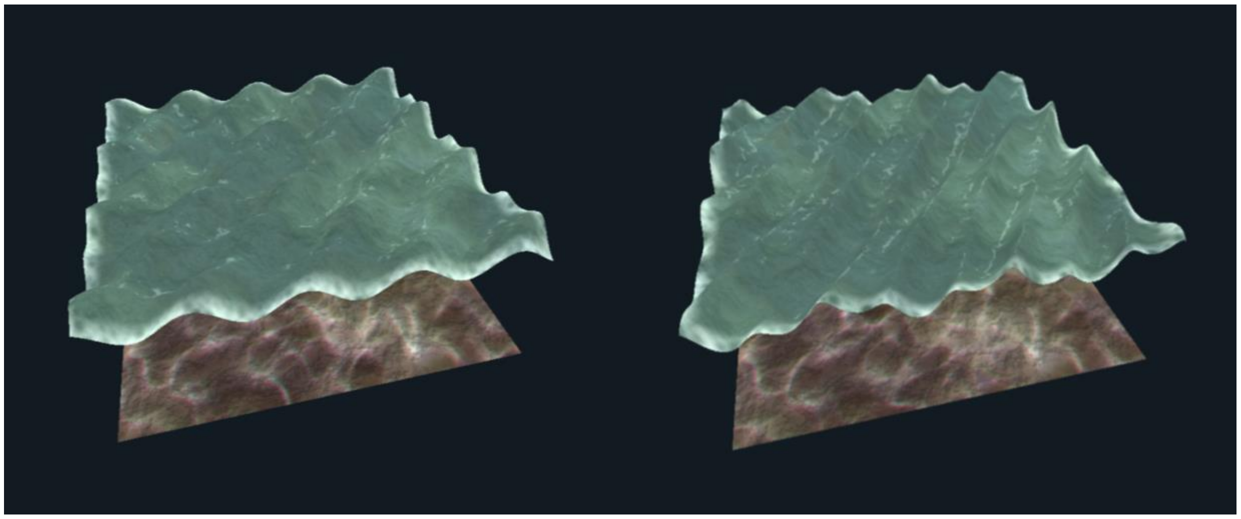


Figure 4. More interesting water wave examples. You can observe that the foam becomes more obvious as the wave amplitude increases