Amit Joshi
Graphics Final Project (Voyage) report

GOALS:

Overall goal: to simulate water and how it interacts with other objects and light

Boat Bobbing/Height with motion of the wave: I wanted to have a boat that would bob up and down with the waves. The key components of this part was to calculate the time-varying displacement, as is done to render waves, and use this same height displacement to determine the height of the boat at any given time.

Boat Rotation and Forward/Backward Motion: To rotate the boat, I kept track of the angle the boat was rotated by, basically, how much the boat was rotated along the y (up) axis. Then, when the user rotates the boat using the controls, the boat should rotate in real time and any user-driven changes to the boat would reflect this rotation. Key idea is that this requires a change to the boat's model matrix.

Caustic water effects: I wanted to simulate the way the refraction light focused in certain areas after being refracted by the water. This is an approximation, as real-time physically accurate caustics are very slow. So, I used some of the hacks mentioned in https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch02.html.

Additive Blending: I wanted to show the caustic water effects while over water, and not just while under it. This involves blending rgba values from both the water and the ocean floor underneath it.

FPS Mode: I wanted to provide the user with a more fun, game-like view of the water, to more easily see how the boat bobs up and down and to get closer to the caustic water effects. It also gives the feel as if the user is driving the boat. Key idea is the eye should be on the boat.

Shadow Mapping: I wished to render shadows from the boat onto the water using point shadow mapping. This didn't end up working, but I have left my implementation details in the next page.

To fill out the ecis survey: I filled out the eCis survey for this class.

Implementation:

Boat Bobbing/Height with motion of the wave: The height offsets of each of the waves were calculated added together, and the boat was offset by this value in the y-direction before rendering so the boat bobs up and down. So, in other words, the model matrix that transformed vertices from local to global coordinates contained a translation in the y-direction to shift the boat up and down. This was nearly copied from the displacement mapping code, which changed the height of the water. (Most of this part was done as the extra credit for one of the previous assignments)

Boat Rotation: To render the rotated boat, this required a change to the boat's vertex shader, specifically, the model matrix that converts from the boat's local coordinates to global coordinates. So, I passed the direction the boat was facing (which is an angle), into the boat's vertex shader. Inside the boat's vertex shader, I computed the rotation matrix around the y-axis using the so it could rotate the boat around its center. After multiplying the x and z coordinates by this rotation matrix, the translation was applied. Both the translation, which reflected the boat's position, and the rotation angle were passed in as uniforms.

User-driven boat motion: Since I kept track of the angle the boat was rotated, I could adjust the user driven forward backward boat motions by simply adding a multiple of the direction vector to the current boat position.

Caustics: To implement caustic water effects, I needed to change the fragment shader of the water, as caustic water effects cause varying brightness on the ocean floor. When coloring a point on the ocean floor, I calculated the normal of the water right above, and used reverse Snell's law to find the direction of the ray as it leaves the water. So, while in the water, the simulated reverse light ray goes straight up, but when it leaves the water it gets refracted to some different direction. Then, I took the dot product of this refracted vector with the vector from the light to the point on the water. And then, I used this dot product to determine the brightness of the point on the ocean floor. But then, I used a hack in which instead of using the refracted ray, I just shot a ray in the ocean normal direction. This didn't change the effect too much. Also, the reason that only rays straight up from the ocean floor were taken into account is that when light enters water at an angle, it has to travel greater distance to hit the ocean floor, so the light will be weaker at that point, so for simplicity we can ignore askew rays.

FPS Mode: Simply relocate the eye to the boat and calculate new view matrices, looking in the direction of the boat.

Additive Blending: I recalculated the color of the floor in the water's fragment shader, and simply did a weighted average.

Shadow Mapping (Attempted, didn't work): First, I drew the scene from the perspective of the light, and calculated new view and projection matrices. Then, in the second pass, I drew the shadows on the ocean by comparing the depth value in the depth map with the current map. The coordinate of each point in the light space was calculated in the Ocean TES. Tried this for directional lights, didn't work so I just followed the tutorial for point shadows. The difference between the two approaches is that the one for point shadows used a cubemap.
Update: After much effort, I couldn't get shadow mapping to work in time either :(

Citations:

https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch02.html

https://learnopengl.com/Advanced-Lighting/Shadows/Point-Shadows