CS5610 Final Project:
Realistic Water Simulation with openGL

Members:
Ben Felsted
EunGyoung Han

Team Name:
glDeepBlue()

Goal:
We had planned on implementing the paper “Interactive Animation of Ocean Waves” from the SIGGRAPH 2004 proceedings. However, resorted to a simplification of the wave model by referring to models suggested in the paper from the SIGGRAPH '86 proceedings: “A Simple Model of Ocean Waves.” This implementation is known as the spectral approach, which entails using the inverse Fast Fourier Transformation to compute height points on a grid of lattice points.

Motivation:
Many projects have been done on rendering realistic water simulation, typically with ocean water being illuminated by a light source overhead, and reflection/refraction and bump mapping being generated at the surface of the ocean water. We thought it would be cool to render this phenomenon in real world. Light is refracted from bottom of the water and the sky is also reflected off of the surface of the water, hence a believable model of ocean water. We would like to extend the project so that we can research more advanced water simulation rendering techniques in the future to complete the more realistic ocean water simulator.

What we achieved:
The project centered around the implementation of a real-time realistic water simulator. The amount of detail that we could have put into this project was virtually unbounded. Although, we considered the governing constraint of this project to be user interactivity—or, more specifically, frame rates; otherwise, we would have been better off implementing a ray-tracer. However, despite the time constraints of this project, we did end up implementing all of the following:

- Use of GPU vertex and fragment processors to accelerate rendering.
  - Reflection
  - Refraction
  - Fresnel Reflectance
    - Fresnel bias
    - Fresnel scale
    - Fresnel power
- Interactive frame-rates.
- Statistically correct model of deep ocean waves—same one used commercially in the movie Titanic and Waterworld.
- User-controllable parameters, including: wave height, animation speed, and index of refraction.
- Small-scale terrain rendering.
- Realtime global illumination (all surfaces are specular and the only emitters are the cube maps).
Tools: We used OpenGL as the programming language and its included vertex and fragment shader language. The code was written in C++ using the MS Visual Studio.Net. The graphics card that was required was GeForce2.

We are completely implement realistic interactive deep water. The images from our ocean model have following images from the screen shot.
History of progress:
   a) Learned interactive deep-water animation and rendering related research works.
   b) Researched, designed, and implement the necessary data structures and numerical algorithms in C++.
   c) Implemented wireframe ocean wave surface model in openGL on the CPU.
   d) Enable animation of wireframe wave mesh.
   e) Coded up normal vector computation.
   f) Implemented flat polygon coloring.
   g) Implemented Goraud interpolation.
   h) Ported CPU code to GPU vertex and fragment shader programs.
   i) Added a skybox.
   j) Cleaned up user interface.
   k) Implemented reflection and refraction on the GPU via environment mapping.
   l) Integrated ocean model with underwater/above water terrain model.
   m) Cleaned up code (which included tracking down any lingering bugs).
   n) Wrote up professional documentation.
   o) Prepared for the presentation.
   p) Delivered the presentation.

Resource from the outside: We refered by the documentation:
   Tessendorf, Jerry, “Simulating Ocean Water” from SIGGRAPH ’86.

Additionally, we read over the SIGGRAPH 2001 Course notes. We also considered improving the wave model by referring to models suggested in the paper from the SIGGRAPH’90 proceedings: “Rapid, Stable Fluid Dynamics for Computer Graphics.”

What we learned: advanced vertex and fragment shader GPU programming techniques—reflection, refraction, environment mapping, animation and simulation, modeling with openGL, and how to budget our time.

If we had more time: Supposing that we had 4 more weeks to accentuate this project, we would definitely implement bump-mapping. As this project stands, the normal vectors are calculated on a per-vertex basis—leading to all sorts of unnatural artifacts. We would like to use a finite-difference PDE solver to compute on a per-fragment basis surface normals conforming to a computational fluid dynamics model: the Navier-Stokes equations. Not only would this provide a more realistic water surface, but it may help eliminate some of the artifacts stemming from the pre-vertex interpolation of normal vectors.

Another addition is of the subsurface scattering variety. The technique entails using a water-color cube map referenced as the refraction cube map. Thus, rays that are perpendicular to the xz-plane (assuming y is up) would be mapped to dark blue, whereas rays that intersect a column of water would be colored light blue.

Before the project deadline, we felt that we were pretty close to rendering the scene with a dynamic cube map. This would allow us to incorprorate terrain in our reflection/refraction water model.
Something that would look really cool is an interactive water simulator with HDR environment maps. We would throw in light bloom, sparkles, lens flare, and whatever else is practicle.

Other stuff includes: fog, caustics, bouyant objects, interactive wave creation, foam textures, spray particle billboards, underwater scene rendering, and adaptive meshing for unbounded fly-overs.
Some artifacts are visible with lowered grid resolutions.

Occasionally, some are visible at high grid resolutions.