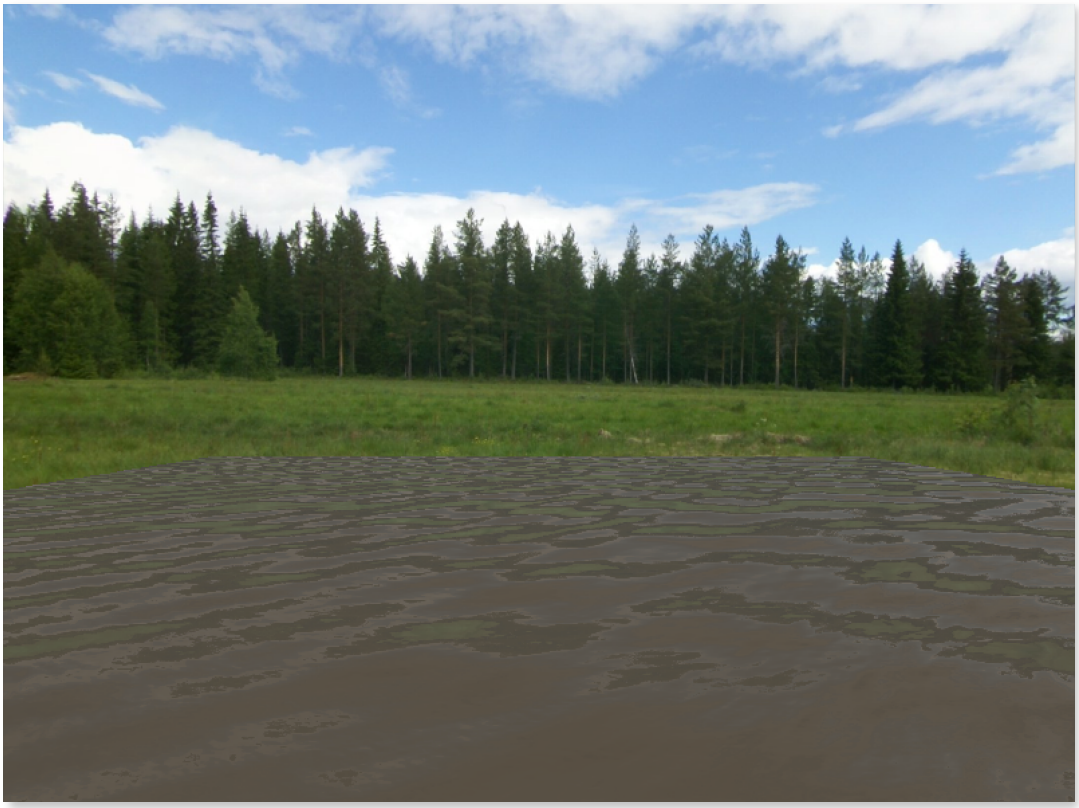
Realtime Water Simulation

CS148: Introduction to Computer Graphics and Imaging



Benjamin Harry

Summer 2016

Realtime Water Simulation

CS148: Introduction to Computer Graphics and Imaging

Problem Statement

Implement a realtime water simulation that demonstrates realistic wave animations and utilizes physical lighting techniques, such as reflections on the water surface and transmission of light through the water. To demonstrate transmission of light through the water (refraction), caustic lighting on the bottom surface was implemented. A stretch goal for this project was to implement wave interaction where the water surface reacts to objects that touch the water surface, such as a boat moving along the surface.

Why a Realtime Water Simulation

It was decided that a realtime water simulation would be attempted due to the many computer graphics challenges involved. A realtime water simulation involves transforming the vertices of a geometric surface in a realistic fashion, modeling physical lighting such as reflection and refraction, and utilizing OpenGL to manipulate geometry in the vertex shader and to perform realistic rendering techniques in the fragment shader in realtime.

Generate Waves Using a Height Field

It was decided that the water's surface would be represented using a three dimensional plane, where the x and z axes represent the horizontal water surface and the y axis represents the vertical height of the wave at a given x, z location. Thus, the first task of this project was to generate a planar surface to be rendered using OpenGL. The plane was implemented using triangle strips and degenerate triangles as described in [2].

Once the planar surface was rendered, it was now possible to move forward and generate waves using this surface. There are many techniques that could be used to

modify the height of the vertices on the plane. For the most realistic wave simulations, Tessendorf [10] recommends using Fast Fourier Transform (FFT) techniques to model realistic ocean waves. While this method was desired, it was not determined how to implement the FFT in the vertex shader to manipulate the y values of the plane's vertices.

The method that was first used to generate waves is the one described by Matthias Müller-Fischer [9]. For this method, the first thing that is done is to initialize the plane's vertex heights (y values) using some function to represent waves. A sum of sines function was used to initialize the vertex heights. Next, each vertex has a corresponding horizontal velocity, which is initialized to zero for each vertex. At each time step in the simulation, the wave's vertex heights are updated based on the algorithm represented by the following pseudocode.

```
for all i, j {
  f = c² * (u[i+1,j] + u[i−1,j] + u[i,j+1] +
      u[i,j−1] − 4u[i,j]) / h²;
  v[i, j] = v[i, j] + f * Δt;
  u_new[i, j] = u[i, j] + v[i] * Δt;
}
for all i, j: u[i, j] = u_new[i, j];
```

Here, f is the force on the wave, c is the horizontal speed of the wave, u is the height at each x (i) and z (j) coordinate, and v is the velocity at each x, z. The h term above is the horizontal width of each column of water centered at each x, z coordinate. This value was set to one for the purposes of this simulation.

The waves generated by this method had a smooth animation, but the wave edges were not smooth when observed up close. Furthermore, this calculation was performed on the CPU because the vertex data was generated and stored on the CPU. As a result, the calculations to update this data were performed at each time step. This proved to be inefficient as two loops were needed: one to first compute vertex heights and another to copy the new heights. This $O(n^2)$ operation resulted

in poor frame rates as the was data first updated and then copied to the GPU's vertex buffer at each time step.

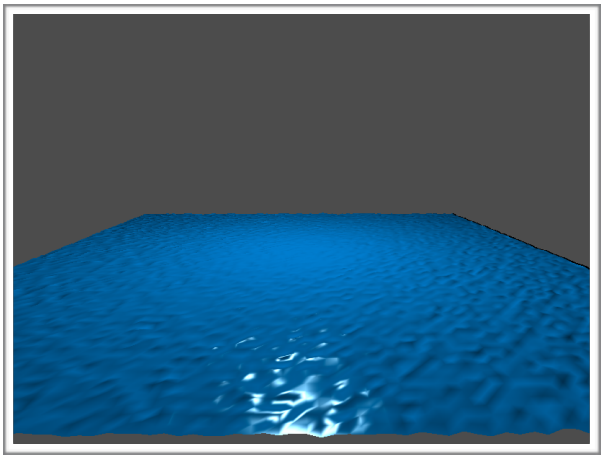


Figure 1: Waves from Müller-Fischer method

Sum of Sines Implementation

As a result of the poor performance of the previous method, a new method was needed that would generate waves in realtime without degrading the simulation's frame rate. After more research, the sum of sines method described by Finch [5] was used. For this method, equations for sine waves with varying amplitudes, directions (x, z), frequencies, and phase were used to compute heights at each x, z position.

$$H_i(x, z, t) = A_i * \sin(\mathbf{D}_i \cdot (x, z) * w_i + t * \varphi_i$$

The height for each sine wave is summed to give the height at each x, z position. In addition, the derivative for this equation is needed to compute the wave's normal at each vertex. The derivative values for each sine wave are also summed to compute the normal at each vertex.

The benefit of the sum of sines was that the calculations were done on the GPU in the vertex shader. As a result, the performance of the simulation did not appear to degrade as it did when using the method on the CPU described above.

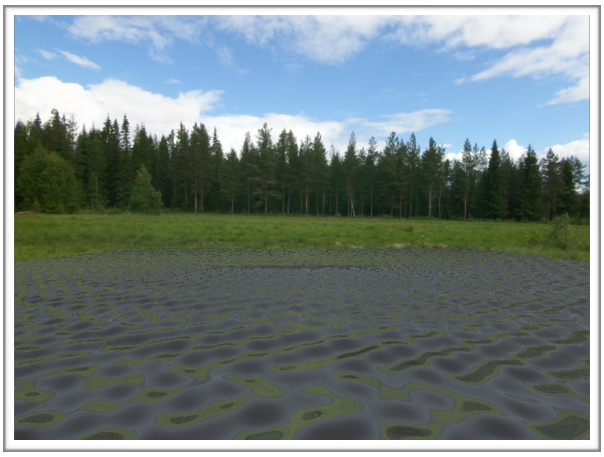


Figure 2: Sum of Sines implementation.

The waves generated using the sum of sines had a "blobby" appearance, which was slightly improved by experimenting with equation coefficients for each sine wave equation. However, a more desirable fix was needed to make the waves look more realistic.

It was determined that the waves needed more high frequency detail that the sum of sines implementation was not providing. To achieve more high frequency detail, a normal map with high frequency normal data was used [1]. The normal map was sampled four times, with time offsets, to produce varying texture coordinates in the fragment shader to provide an animated effect. As a result of adding the high frequency detail, the waves took on a much or realistic appearance.

Now that more convincing waves are generated, it was time to add better lighting that utilized the reflection and transmission of light.

Light Reflection - Fresnel Effect

Light that intersects a water surface will be reflected and also transmitted through the water's surface. How much light is reflected/transmitted depends upon the light's incident angle with the water surface's normal. Tessendorf [10] provides a shader for Renderman that computes the amount of reflected and transmitted light from the surface normal, the light's incident direction vector, and the index of

refraction for air and water. This shader was used as a reference and converted to GLSL for the OpenGL fragment shader. By doing so, the reflected color obtained

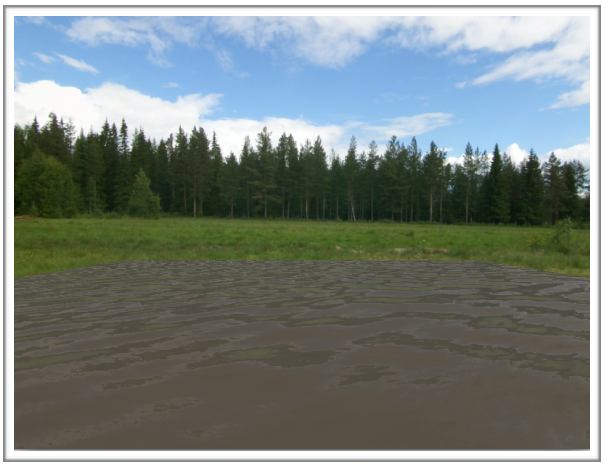


Figure 3: Reflections with Fresnel Effect

from the OpenGL cube map was enhanced with the Fresnel Effect.

Bottom Surface Caustic Lighting

As mentioned above, light arriving at the water's surface is either reflected or transmitted based upon the incident light vector direction. Some of the transmitted light will arrive at the bottom surface, resulting in caustic lighting patterns. The method used to add caustics to the bottom surface was inspired by Guardado [7].

The main idea of the method is to send rays of light in reverse direction from the light source. Thus, the ray originates from the bottom surface (oriented along the surface normal) and then travels to the water surface. The surface normal is then used along with Snell's law to determine the ray that is transmitted into the air. Guardado uses a light map texture to sample based upon the direction of the transmitted ray.

For this project, the dot product of the transmitted ray and the up vector was used to determine how much light to use from this ray. This assumption works if you treat the sun as being directly above the water's surface. Again, the high frequency

normal map added to the realism of the caustic effect. The caustics from the sum of sines wave were not enough to create convincing caustic effects.

Challenges Encountered

There were two main challenges encountered when working on this project. First, it was difficult to determine how to generate animated waves using OpenGL. It is documented that generating waves using a FFT produces the most realistic waves. However, it was not clear what these equations were and how they could be implemented using OpenGL.

Furthermore, it was hard to determine whether or not to do the wave computations on the CPU or GPU. The GPU was preferred as it would provide better simulation performance, but it was not clear how to do advance buffer techniques to preserve data between time steps for the method described by Müller-Fischer. Thus, the first attempt to create waves was done on the CPU.

The second challenge with this project was improving the "blobby" appearance of the waves generated with the sum of sines implementation. Again, it was desired to add high frequency waves to add more wave detail. It was discovered that high frequency waves could be added to the water surface with the use of a normal map. The normal map was sampled four times per time step with uv coordinates derived with different coefficients in combination with the simulation time step [1]. The use of this technique greatly improved the realism of the water's surface and the caustic lighting on the bottom.

Water Surface Interaction

As mentioned in the beginning, it was a stretch goal to add interaction with the water surface (such as a boat wake or splashes from objects falling into the water). The ability to generate water ripples from a point was added to the simulation. With these ripples, an attempt was made to try to animate them in some way that might represent a boat wake. The results of this investigation did not lead to usable results.

Furthermore, more investigation was done to add animated boat wakes by using boat wake normal map and height map textures. The idea was to use two dimensional scale, rotation, and translation matrices in the vertex shader to move the wake texture with a position passed from the main application [6]. Before running out of time, the wake image was animated in a circular path on the water surface. However, there was not enough time to orient the wake along the path heading. In addition, better wake textures were needed to produce more convincing results.

Potential Improvements

While the results of this simulation are nice, better waves could be generated by figuring out how to implement the FFT method described by Tessendorf. In addition, the use of the shallow water equations would most likely produce the type of waves that were desired in the beginning. Both Tessendorf and Müller-Fischer mention the use of shallow water equations.

By using FFT's and these equations to make better waves, it is assumed that the bottom surface caustics could also be improved. Without the use of the high frequency normal map the caustics do not look right.

Finally, spending more time on boat wake implementation would produce nice object-water surface interaction. Animating this effect using similar techniques to the high frequency normal map would most likely result in a nice wake effect.

References

1. Bouny, Jeremy. "Realistic Water Shader for Three.js." GitHub. github.com/jbouny/ocean
2. Brothaler, Kevin. "An Introduction to Index Buffer Objects." www.learnopengles.com/android-lesson-eight-an-introduction-to-index-buffer-objects-ibos/

3. de Greve, Bram. "Reflections and Refractions in Ray Tracing." Nov 13, 2006. graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf

4. de Vries, Joey. Learn OpenGL. www.learnopengl.com

5. Finch, Mark. "Effective Water Simulation from Physical Models." GPU Gems. 2004. developer.nvidia.com/gpugems/GPUGems/gpugems_ch01.html

6. Gonzalez Vivo, Patricio and Lowe, Jen. "The Book of Shaders." thebookofshaders.com

7. Guardado, Juan. "Rendering Water Caustics." GPU Gems. 2004. developer.nvidia.com/gpugems/GPUGems/gpugems_ch02.html

8. Hollasch, Steve. Steve's Web Pages. Nov 4, 2007. steve.hollasch.net/cgindex/render/refraction.txt

9. Müller-Fischer, Matthias. "Fast Water Simulation for Games Using Height Fields." GDC 2008. matthias-mueller-fischer.ch/talks/GDC2008.pdf

10. Tessendorf, Jerry. "Simulating Ocean Water." 1999 - 2001. graphics.ucsd.edu/courses/rendering/2005/jdewall/tessendorf.pdf