Simple Rule-Based Water Simulation

Jonathan Lew

CMPS162 Advanced Computer Graphics and Animation

Abstract. Water Simulation is often beautiful and stunning. Different methods exist to simulate water surfaces. One of the more well-known scenes in water simulation would be droplets of water falling onto a water surface and causing ripples. In this report, I would study popular implementation of water simulation, and discuss my implementation, which makes use of a simple and fast dual buffer algorithm.

1 Introduction

In this project, I attempted to implement a simple rule-based water simulation. I had always been fascinated by water surfaces, especially during rain, where ripples would pop up on water surfaces in a chaotic yet orderly fashion. Before starting on this project, I explored various techniques to attempt to recreate water surfaces and the rippling effect. I found two advanced techniques, and one simple technique. After extensively researching the two advanced methods, I decide against them due to a lack of time. As such, I decided to go with the dual buffer technique, which was supposed to be fast and simple.

1.1 Tessendorf's Ocean Simulation

In 1999, Jerry Tessendorf published a paper about simulation ocean water. In the heydays of computer graphics and special effects, CG water was rising in popularity. In his paper, Tessendorf highlighted many methods involved in simulating water surfaces. This paper is constantly quoted whenever water simulation is required, and it covers a comprehensive set of formulas and things to look out for when attempting to create a simulation of a water body.

Radiosity between the water surface, air, sun and water beneath the surface. Radiosity in this system is computed by calculating the sum of the light coming from the sun, atmospheric skylight and light from below the surface. This is then multiplied by the Fresnel reflectivity from the spot on the surface.

Radiosity below the surface is similar, in that it requires several components. Light from the sun that directly penetrates into the water and indirect light from the atmosphere that penetrates into the water are multiplied by Fresnel's tranmissivity. Furthermore, single scattered light from the sun and atmosphere and multiply scatter light are also included in this radiosity calculation.

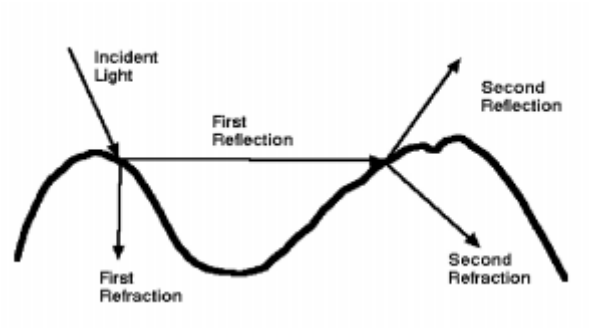

Figure 1 Light's path on the ocean surface Source – J Tessendorf

Building Height fields for waves using Gerstner waves, combined with Statistical Wave models and Fourier Transform to increase realism. For ocean waves, we could use both Fast Fourier Transforms or Gerstner Waves. Gerstner waves are used because the mathematics behind it is more straightforward. However Fast Fourier Transform can be used to simulate more complicated waves like "linear waves" or "gravity waves", which are more accurate analogs to their real world counterparts.

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_0 - (\mathbf{k}/k)A\sin(\mathbf{k}\cdot\mathbf{x}_0 - \omega t) \\ y &= A\cos(\mathbf{k}\cdot\mathbf{x}_0 - \omega t)\ . \end{aligned}$$

Figure 2 Formula for Gestner Waves

Gertsner Waves takes into consideration a wave vector, k, which is 2 PI divided by the wavelength. The frequency of the wave is calculated with respect to the wavelength. For the animation of waves can be made ore realistic by making use of the dispersion relation.

Statistical Wave Models or Fourier Transform can be used for a more realistic approach to simulate ocean water. The main advantage of a statistical model is that wave height fields could be decomposed into sine and cosine waves, making modeling the waves extremely easy. This decomposition can be achieved rapidly using Fast Fourier Transforms.

After looking at many more methods to increase realism of the waves, such as achieving the choppy effect of waves, Tessendorf continues to investigate the optics behind the surface waves, which investigates how a light ray and its energy is transmitted while interacting with the water ocean surface. Specifically, he investigates specular reflection and transmission, Fresnel Reflectivity and Transmissivity. Finally, Tessendorf looks at how the volume underneath the water surface disperses the water by discussing the caustic pattern.

These techniques all work together to create a realistic model of ocean water, the light interaction between all bodies in the system and water surface itself. Needless to say, within the timespan of a quarter, the scope of the project would not be anything close to that of what was described in the paper.

1.2 Tessellation Shaders

In 2007, Mark Finch and Cyan Worlds described in the book GPU gems, an effective water simulation from physical models using fourier

transform and shaders. The following section is a summary of the method described by Mark Finch.

In this simulation, the water simulation was found to be suitable for real-time game simulations. The method started off from combining two sine waves to obtain the height of the water.

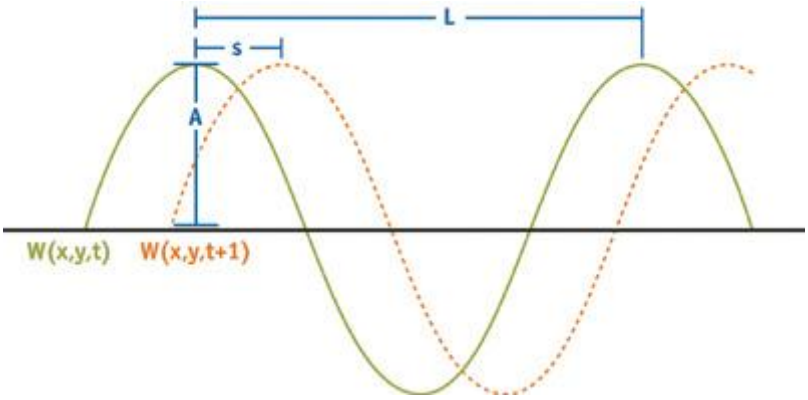

Figure 3 Combining Two Sine Waves - Source - GPU Gems

$$H(x,y,t) = \sum\left(A_i \times \sin\left(\mathbf{D}_i \cdot (x,y) \times w_i + t \times \varphi_i\right)\right),$$

Figure 4 Formula for obtaining height of wave - Source GPU Gems

Since we are making use of vertices, we would have to process each vertices based on its horizontal position. We need to generate a normal map from the sum of sines.

We first computing the wavelength, amplitude, speed and direction of the wave. There is also a need to compute the binormal and tangent. Most of the math here involves partial derivatives. We will then be able to get a formula for the height of any point on the water surface. The end product is a geometric wave formula that varies with respect to a constant.

In large water bodies, we would prefer to use directional waves instead of radial waves as it allows for an accurate simulation of winds. Circular waves add interesting results because their interference is more unique. In this implementation, the focus is placed on directional

waves. Because summing sine and cosine waves tend to result in waves that have too much "roll", Gerstner waves are used, just like Tessendorf.

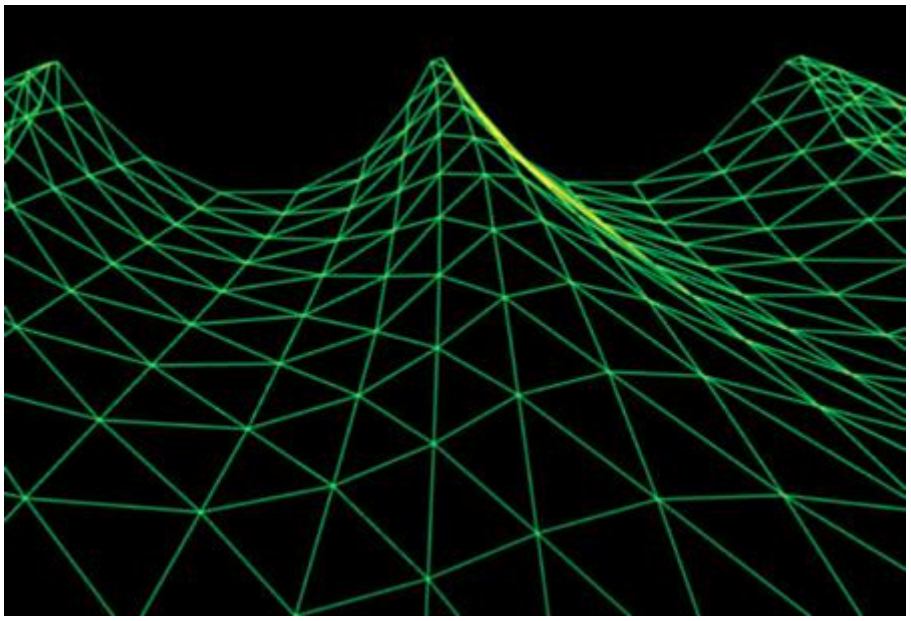

Figure 5 Gerstner Waves, Source - GPU Gems

$$\mathbf{P}(x,y,t) = \begin{bmatrix} x + \sum\left(Q_i A_i \times \mathbf{D}_i.x \times \cos\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right), \\ y + \sum\left(Q_i A_i \times \mathbf{D}_i.y \times \cos\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right), \\ \sum\left(A_i \sin\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right) \end{bmatrix}.$$

Figure 6 Formula for Gerstner Wave - Source GPU Gems

The method described by Finch continues to vary the wave by modifying amplitude, direction and speed. The methods described focuses on trying to give a natural look, so some formulas may be modified in order to avoid an unnatural look. This is done by specifically selecting appropriate wavelengths.

2 Dual Buffer Method

Eventually, I decided to implement the simpler algorithm describe by Hugo Elias on his website. This method was described to be simple, without sine or cosine functions. Instead, in relied on a rule-based simulation, which tries to simulate the movement of water surface using two buffers. One of these buffers would be used to hold the previous versions.

2.1 Switching between buffers

```
damping = some non-integer between 0 and 1

begin loop
    for every non-edge element:
    loop
        tempValue = (Buffer1(x-1,y)
                     Buffer1(x+1,y)
                     Buffer1(x,y+1)
                     Buffer1(x,y-1))/4
        tempValue -= Buffer2(x,y)

        Buffer2(x,y) = tempValue * damping
    end loop

    Display Buffer2
    Swap the buffers

end loop
```

The pseudo code for this algorithm is described above. The core concept behind this algorithm is to use two buffer of height maps. One buffer stores the current height map, while the other buffer stores the previous height map. Using the current height map and previous height map, we can calculate the next frame, and we will use that to overwrite the buffer.

To understand how the calculation is done, we will first look at the problem as a 1-dimensional wave. Taking reference from figure 5, we can see that the wave is travelling left. The arrows indicate how the water level is going to change at that point. Assuming that the wave is moving at constant velocity, we would be able to compute the current

frame by using the previous two frames since the displacement is proportionate.

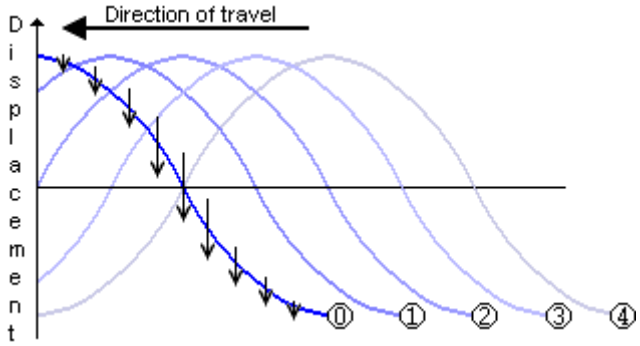

Figure 7 Explanation Behind Buffer Switching Algorithm

In our algorithm, we achieve this movement by the line "`tempValue -= Buffer2(x,y)`". Since our wave is going to travel in a 2D plane, we would need to do some smoothening. This is done by taking the 4 direct neighbors and finding the average. Finally, we want the wave to loose energy, and this is done by using a dampening factor.

2.2 Actual Implementation

In our actual code, what we would need to do would be maintain a pointer on the current buffer. This allows us to know which buffer to add a new drop of water to when mouse or keyboard interactions are triggered. With each call of the OpenGL display function, we will need to run the calculation for buffers again.

Once we have the buffer, all we need to do is to link up all the points into polygons to create a water surface. There is also the matter of rendering the water surface. Since we have the height map, we can easily use the height map to draw the color of the polygon.

Droplets of waters are simulated by manually setting the height of a point to a certain height. When the height map is calculated at each frame, the "droplet" of water would quickly be spread out to its neighbors.

For mouse input, I implemented mouse picking by color, which I've learn over the course of this class. This basically involves an off screen rendering of the polygons, but in unique colors using their row and column positions in the buffer. A mouse click would return a single pixel with its color. Using those color, I can re-identify the polygon involved. Correspondingly, I can set the "previous" buffer to a height that simulates a brand new drop of water.

2.3 Limitation of algorithm

There are a few problems with this implementation of a water surface. Firstly, because we are only accessing the left, right, top and bottom neighbors in the buffer, the shape of the ripple is squarish in shape. This effect can be more clearly observed when it is zoomed in.

Also, even because of the way the height map is calculated, waves might end up passing by each other. Waves can be reflected off the edge, however it has a very rigid look, making it feel unrealistic. This can be minimized by tweaking the dampening factor and the height at which a droplet is dropped. Also, randomization could be done to the height of droplets, allowing for different sized ripples, which closer to a real world scenario.

A third problem is that even though for a small water surface this implementation is efficient, being $O(n^2)$, once the size of the water surface increased, I was able to observe severe lag.

2.3 Possible Improvement

The first improvement that I would like to make to this project is to use actual lighting instead of pseudo lighting, which is simply setting the color based on the height map.

Another way I would improve this method would be add multiple levels of dual buffers. With that, I can simulate an actual droplet, instead of directly modifying the height of the buffer. I would be able

to add additional rules such as a huge droplet causing a splashback of a new droplet that travels up and down.

Another possible improvement would be to make the surface affect how light travels. Since the buffer structure already discretizes the water surface, I can set up a view grid with the same number of buffers and project a wave and refract it. By putting an object underneath the water surface, I might be able to reproduce a more interesting and useful rendering of the water surface, which deals with refraction.

However the second improvement brings up the problem of deciding how to deal with cases where the refraction causes one discrete unit of the water surface to spread over multiple surfaces.

3 Conclusion

Unfortunately, mainly due to time constrains, I was unable to investigate the more complicated simulation models. However, I feel that the dual buffer simulation method is extremely fast and simple for a pseudo-realistic simulation of water droplets. Given more time and resources, I would most likely want to pursue the model as describe by GPU Gems.

References

1. Tessendor, Jerry “Simulating Ocean Water”, (1999)
2. Finch, Mark, and Cyan Worlds. GPU Gems Chapter 1. 5thth ed. N.p.: Pearson Education, Inc, 2007. Print.
3. Hugo Elias. (1998). Graphics. In 2D Water. Retrieved Apr 1, 2012, from http://freespace.virgin.net/hugo.elias/graphics/x_water.htm.