Ocean Water Simulation

Mengmeng Liu
liumengm@usc.edu

Yunxuan Cai
yunxuanc@usc.edu

Zicheng Wang
zichengw@usc.edu

ABSTRACT

Ocean water rendering is one of the most popular and challenging fields in computer graphics. In order to make the rendering more realistic, efforts are mainly carried out from two parts: wave shape and shading. In this project, we implemented Tenssendorf's FFT-based (spectrum-based) algorithm to obtain real-time execution waves and used Phong shading to get semi-realistic lighting.

Our video can be found here: https://youtu.be/OqywbDx8iuQ

1 INTRODUCTION

In the game and film industries, various methods depend on different hardware and effect requirements are used to simulate ocean water. These methods can be roughly divided into wave superposition, wave particles, fluid mechanics, and flow map.

As one of the basic wave superposition methods, Gerstner Waves was first modeled into computer graphics by Fournier and Reeves[1]. It is a computable method with a decent effect. However, oceanographic literature does not consider Gerstner waves as a realistic model of the ocean. Instead, statistical models are used[2]. By IFFT, ocean spectra can be transformed into the spatial domain.

Wave particles is a method that uses wave equation but considers waves as particles[3]. It is mostly used for simulating the ripples caused by the interaction between objects and the water surface.

The simulation of fluid mechanics based on Navier-Stokes Equations has a higher degree of realism. This method requires a large amount of calculation; hence it can only be rendered offline.

Flow map method is a kind of UV animation. It is a texture containing 2D flow vectors used in the pixel shader to distort the normal map in the flow direction[4].

The goal of our project is to create a real-time rendering ocean without interaction. Therefore, we generated waves based on Jerry Tenssendorf's algorithm (cited previously), which used Phillips spectrum as the statistical model.

2 WAVE GENERATION

2.1 Tessendorf's Algorithm

In the statistical models, the wave height is considered a random variable of horizontal position and time. The FFT-based representation of a wave height field $h(\mathbf{x}, t)$ can be expressed by the following inverse discrete Fourier transform (IDFT):

$$h(\mathbf{x}, t) = \sum_{\mathbf{K}} \tilde{h}(\mathbf{K}, t) exp(i\mathbf{K} \cdot \mathbf{x}) \tag{1}$$

where $\mathbf{x} = (x, z)$ is the horizontal position, $\mathbf{K} = (k_x, k_z)$ is the frequency domain position and t is the time. The height amplitude Fourier components $\tilde{h}(\mathbf{K}, t)$ can be computed from analytical wave spectrum model. In Tessendorf's algorithm, he used a modified Phillips spectrum to produce the Fourier amplitudes:

$$\tilde{h}(\mathbf{K}, t) = \tilde{h_0}(\mathbf{K}) exp\{i\omega(k)t\} + \tilde{h_0^*}(-\mathbf{K}) exp\{-i\omega(k)t\} \tag{2}$$

$$\tilde{h_0}(\mathbf{K}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P_h(\mathbf{K})} \tag{3}$$

$$\sqrt{P_h(\mathbf{K})} = A\frac{exp(-1/(kL)^2}{k^4}|\hat{K} \cdot \hat{\omega}|^2 \tag{4}$$

where ξ_r and ξ_i are ordinary independent draws from a gaussian random number generator, with mean 0 and standard deviation. $L = V^2/g$ is the largest possible waves arising from a continuous wind of speed V, g is the gravitational constant, and $\hat{\omega}$ is the direction of the wind. A is a numeric constant.

In order to find the surface normal, the slope vector of the wave height field is needed:

$$\epsilon(\mathbf{x}, t) = \nabla h(\mathbf{x}, t) = \sum_{\mathbf{K}} i\mathbf{K}\tilde{h}(\mathbf{K}, t) exp(i\mathbf{K} \cdot \mathbf{x}) \tag{5}$$

For generating choppy purpose, displacement vectors are also needed:

$$\mathbf{D}(\mathbf{x}, t) = \sum_{\mathbf{K}} -i\frac{\mathbf{K}}{k}\tilde{h}(\mathbf{K}, t) exp(i\mathbf{K} \cdot \mathbf{x}) \tag{6}$$

2.2 Our Implementation

The discrete Fourier transform needs a large amount of calculation, with a complexity of $O(N^2)$. Therefore, fast Fourier transform (FFT) is widely used to reduces the complexity to $O(NlogN)$. We used FFTW Library[5] to realize this process.

As a 2D Fourier transform, equation 1 can be written as follow:

$$h(x, z, t) = \sum_{m=-\frac{M}{2}}^{\frac{M}{2}-1} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \tilde{h}(\frac{2\pi n}{L_x}, \frac{2\pi m}{L_z}, t) exp(i2\pi(\frac{n}{L_x}x + \frac{m}{L_z}z)) \tag{7}$$

where N and M are grid size, L_x and L_z are horizontal dimensions.

To make the summation start from 0, we denote $n = n' - \frac{N}{2}$, $m = m' - \frac{M}{2}$. Thus, $k_x = \frac{2\pi n' - \pi N}{L_x}$, $k_z = \frac{2\pi m' - \pi M}{L_z}$.

The final height equation of the ocean surface is

$$h'(x, z, t) = \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \tilde{h}'(n', m', t) exp(i(\frac{x(2\pi n' - \pi N)}{L_x} + \frac{z(2\pi m' - \pi M)}{L_z})) \tag{8}$$

We then applied inverse FFT to calculate displacement, height, and normal. The normal vector is given by,

$$N(\mathbf{x}, t) = normalize(-\nabla h_x(\mathbf{x}, t), 1, -\nabla h_z(\mathbf{x}, t)) \tag{9}$$

where $\nabla h(\mathbf{x}, t)$ can be calculated from equation 5.

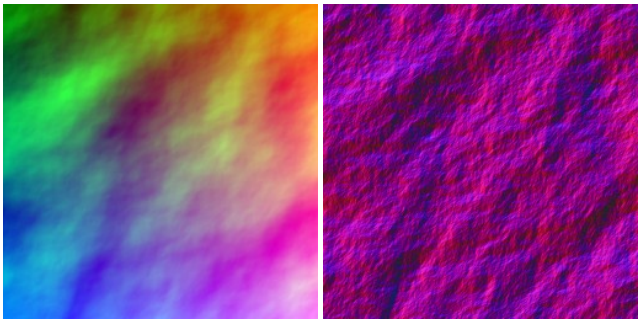

Figure 1: (a) Displacement (b) Normal

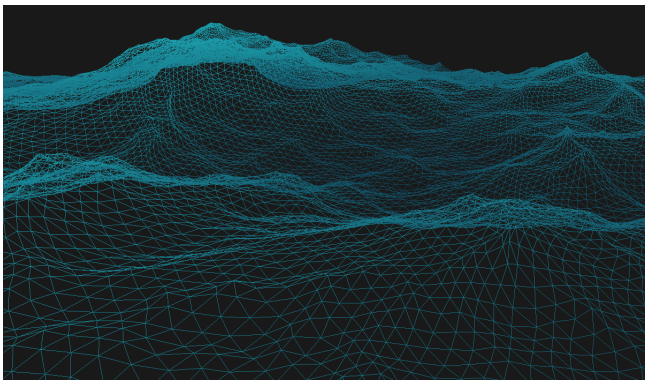

Figure 2: Wireframe mode

Here, we use (x, y, z) as RGB color to display displacement and normal fields. It is worth noting that the green color represents the height of waves. We set $N = M = 256$, $L_x = L_z = 1000$, $A = 6e-7f$, $V = 30$, $\omega = (1, 1)$ to get figure 1 and 2.

We also calculated Jacobian here in order to generate whitecap. This is explained in more detail in part four.

3 SHADING

3.1 Baisc Lighting

The surface basic shading of our project is based on Phong shading. Three reflection behaviors, including ambient reflection, diffuse reflection, and specular reflection, are calculated. The light direction is the same as the sun in the sky model, which changes every frame. To make the water looks translucent, we use a depth-based lookup table. Every pixel in rasterization is assigned a blue-like color based on its relative height:

$$H_{relative} = \frac{H - H_{min}}{H_{max} - H_{min}} \tag{10}$$

$$color = H_{relative} \cdot color_{light} + (1 - H_{relative}) \cdot color_{dark} \tag{11}$$

The maximum and minimum wave heights of each frame are computed in the CPU. The light color in our project is set to RGB(0.1, 0.86, 0.92), and dark color is set to RGB(0.02, 0.09, 0.2). We use this mixed color as wave texture and set ambient and diffuse coefficients equal to it.

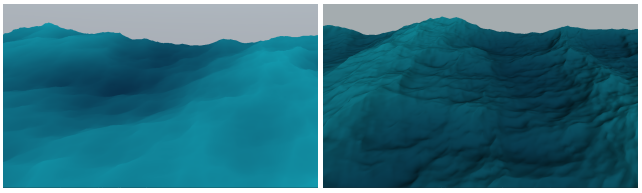

Figure 3: (a) Ambient light (b) Diffuse reflection

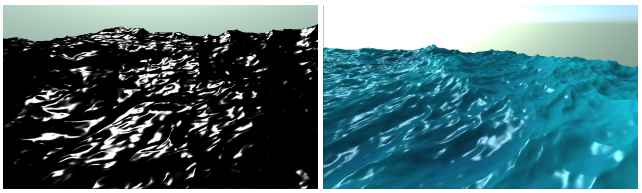

Figure 4: (a) Specular reflection (b) Phong shading

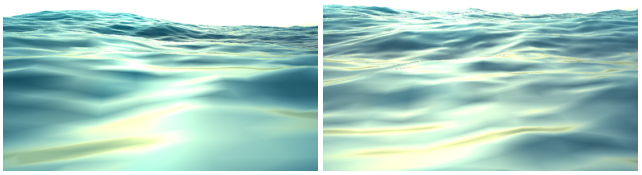

Figure 5: (a) No SSS (b) FSSS

3.2 Subsurface Scattering

As shown in Figure 4(b), the basic shading looks a little bit viscous. In order to improve translucency, we applied a subsurface scattering method brought up by Frostbite Engine at GDC 2011. This is an approximating translucency method for a fast and convincing effect. When looking at the water surface under the sun, the phenomenon that some light scattered by the subsurface in the water is very significant.

The amount of back lighting in this Fast Subsurface Scattering is

$$I_{back} = saturate(V \cdot - < L + N\delta >)^p \cdot s \tag{12}$$

where δ is the subsurface distortion, p(power) and s(scale) are used to change the properties of the curve. The <> symbol here is a normalize function.

In our project, we set $\delta = 0.522$, $p = 2.28$, and $s = 0.918$. Figure 5 shows the contrast without SSS and with FSSS.

3.3 Environment Reflection

The environment reflection in our project has two parts: a sky model and objects above the surface. For the sky model, we used Hosek-Wilkie Sky Model[6] to simulate the offshore ocean's sky environment. The spectral irradiance L is given by,

$$L_{\Omega,\lambda} = \frac{\partial L_\Omega}{\partial \lambda} = HW(l, \lambda, t, \sigma_g, l_s) \tag{13}$$

Figure 6: Box reflection

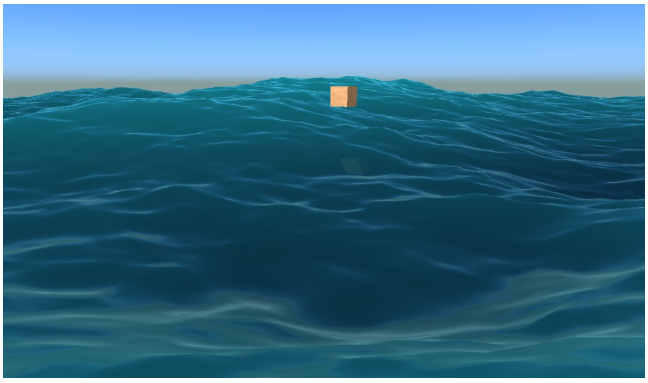

Figure 7: Shading result

where l is the direction to center of the hemisphere, λ is wavelength, t is atmospheric turbidity, σ_g is surface albedo, and l_s is the sun's position.

The code provided by the authors can sample HDR color of the sky from any angle. We wrote an interface to generate a 2D (SDR) skybox texture in real-time for ocean dynamic environment texture. HDR textures also give our project the potential for image-based lighting.

A textured box is used to represent objects above the ocean surface. We set another camera beneath the ocean surface to render the underside of the box. Use it as texture to simulate the reflection effect. The box reflection shows in figure 6.

To make the reflection more realistic, we used Fresnel equation to get the ratio of reflection and transmission. Considering the complexity of this equation, we actually used a Simplified Fresnel equation proposed by Christophe Schlick[7], which is widely used in real-time rendering:

$$F(\vec{v}) = F_0 + (1 - F_0)(1 - (\vec{v} \cdot \vec{n}))^5 \tag{14}$$

F_0 is the Fresnel value when light is incident perpendicularly. $\vec{n}$ is the normal and $\vec{v}$ is the view direction, which is from the incident point to the camera. We set $F_0 = 0.05$ in our project.

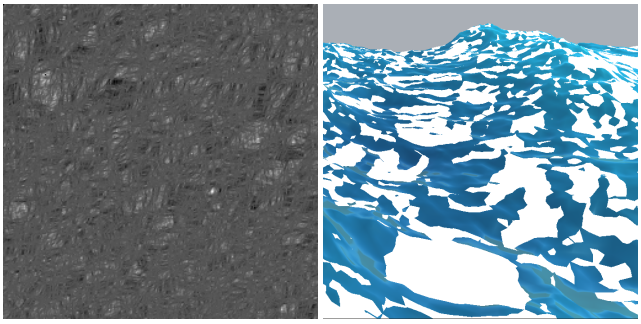

Figure 8: (a) Folding map (b) J<0 area

4 FOAM

4.1 Foam Region Test

In Tessendorf's Algorithm, he mentioned that we could use Jacobian determinant to check if the region is overlap. When the displacement is zero, the Jacobian is 1. When there is displacement, the Jacobian has the form:

$$J(\mathbf{x}) = J_{xx}J_{yy} - J_{xy}Jyx \tag{15}$$

with individual terms

$$J_{xx}(\mathbf{x}) = 1 + \lambda \frac{\partial D_x(\mathbf{x})}{\partial x}$$

$$J_{yy}(\mathbf{x}) = 1 + \lambda \frac{\partial D_y(\mathbf{x})}{\partial y}$$

$$J_{xy}(\mathbf{x}) = J_{xy}(\mathbf{x}) = \lambda \frac{\partial D_x(\mathbf{x})}{\partial y}$$

Besides, many other methods can be used to check if the region should generate foam. GPU Gem 2 introduced a method that checks wave crest based on the relative height[8]. Sea of Thieves team brought up an interactive method based on the scene depth[9].

Here we choose to use Jacobian determinant to decide the foam region.

4.2 Foam Render

All Jacobian matrices are calculated in the CPU. They generated a folding map, as shown in figure 8(a). We then pass them as vertices' attribute to the shader. The rasterization process interpolated J to each pixel. After setting the threshold to -0.0001, we got the whitecap area shown in figure 8(b). Here we set the color into white and without any shading just to check the area.

Many papers and talks have put forward their idea on how to render the foam area. Most of them made improvements on how to decide the color ratio[10][11]. Our project uses a simple linear interpolation method. For the pixel has $J > -0.0001$, no foam generated, thus, stay original color. For the pixel has $J < -0.002$, set its color into white. For the pixel has $J \in [-0.002, -0.0001]$, uses a linear ratio to decide its color. Threshold numbers are results of the test.

However, the final result is unsatisfactory. After extensive tests and long time analysis, we concluded two reasons might lead to this unpleasant result.

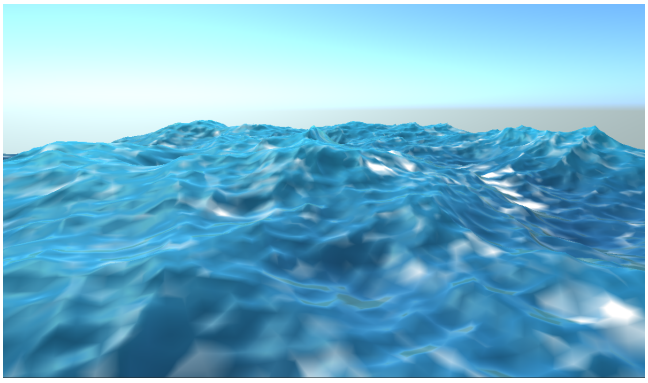

Figure 9: Simple whitecap. Specular lighting is off.

The first reason is that the Jacobian matrix has lost accuracy after the whole computation. Actually, Tessendorf has mentioned this problem in his paper. He reminded the reader should be careful because the limits of numerical accuracy for floating point calculations can become noticeable. A figure in his paper also shows that the folding map value fluctuated wildly between -0.1 and 0.6. Therefore, the chances are the accuracy problem causes the folding map to look chaotic.

The second one is the lack of texture and animations, which could be improved in the future.

5 LIBRARIES AND FRAMEWORKS

Our project is built on openGL. Combined with GLFW and GLAD to configure an OpenGL context and spawn a window. We used some of the basic functions, such as motion control and camera. The coding frame is adapted from https://learnopengl.com/. We also use GLM to do matrix transform, including translate, scale and rotate.

For the wave generation part, as mentioned before, we use FFTW Library to calculate IFFT.

We also use stb library created by Sean Barrett to read and write images. More specifically, we use *stb_image* to read texture files and *stb_image_write* to create displacement, normal and folding maps in figure 1 and figure 8.

6 FUTURE WORK

For wave generation:

- The FFT calculation in our project is currently in the CPU. We plan to switch to a more efficient GPU-based FFT calculation. For example, the Compute Shader in openGL.
- In order to generate a larger size ocean with higher efficiency, except increasing the mesh size in FFT, we could also apply some level of detail (LOD) techniques.

For shading part:

- Replace Phong lighting with image-based lighting, which is the HDR texture sampled from the sky model. In this way, the light color will change as the sun penetrating the atmosphere.
- As mentioned before, the accuracy problem needs to be solved in the foam part. We also plan to add shore foam.
- Use wave particles method to realize the interaction with the water.

REFERENCES

[1] Alain Fournier and William T. Reeves. A simple model of ocean waves. In David C. Evans and Russell J. Athay, editors, *SIGGRAPH*, pages 75–84. ACM, 1986.

[2] J. Tessendorf. Simulating ocean water. 2004.

[3] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007.

[4] Alex Vlachos. *Water Flow in Portal 2*, 2010.

[5] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[6] Alexander Wilkie and Lukas Hosek. Predicting sky dome appearance on earth-like extrasolar worlds. In *29th Spring conference on Computer Graphics (SCCG 2013)*, May 2013.

[7] Christophe Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994.

[8] Matt Pharr, editor. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley, 2005.

[9] Nigel Ang, Andrew Catling, Francesco Cifariello Ciardi, and Valentine Kozin. The technical art of sea of thieves. In *ACM SIGGRAPH 2018 Talks*, SIGGRAPH '18, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Lining Chen, Yicheng Jin, and Yong Yin. Ocean wave rendering with whitecap in the visual system of a maritime simulator. *Journal of Computing and Information Technology*, 25:63–76, 03 2017.

[11] Jonathan Dupuy and Eric Bruneton. Real-time animation and rendering of ocean whitecaps. In *SIGGRAPH Asia 2012 Technical Briefs*, SA '12, New York, NY, USA, 2012. Association for Computing Machinery.