

Ocean Wave Simulation

王勁智

Email: chinchih15@gmail.com

NTU r98922125

吳家翔

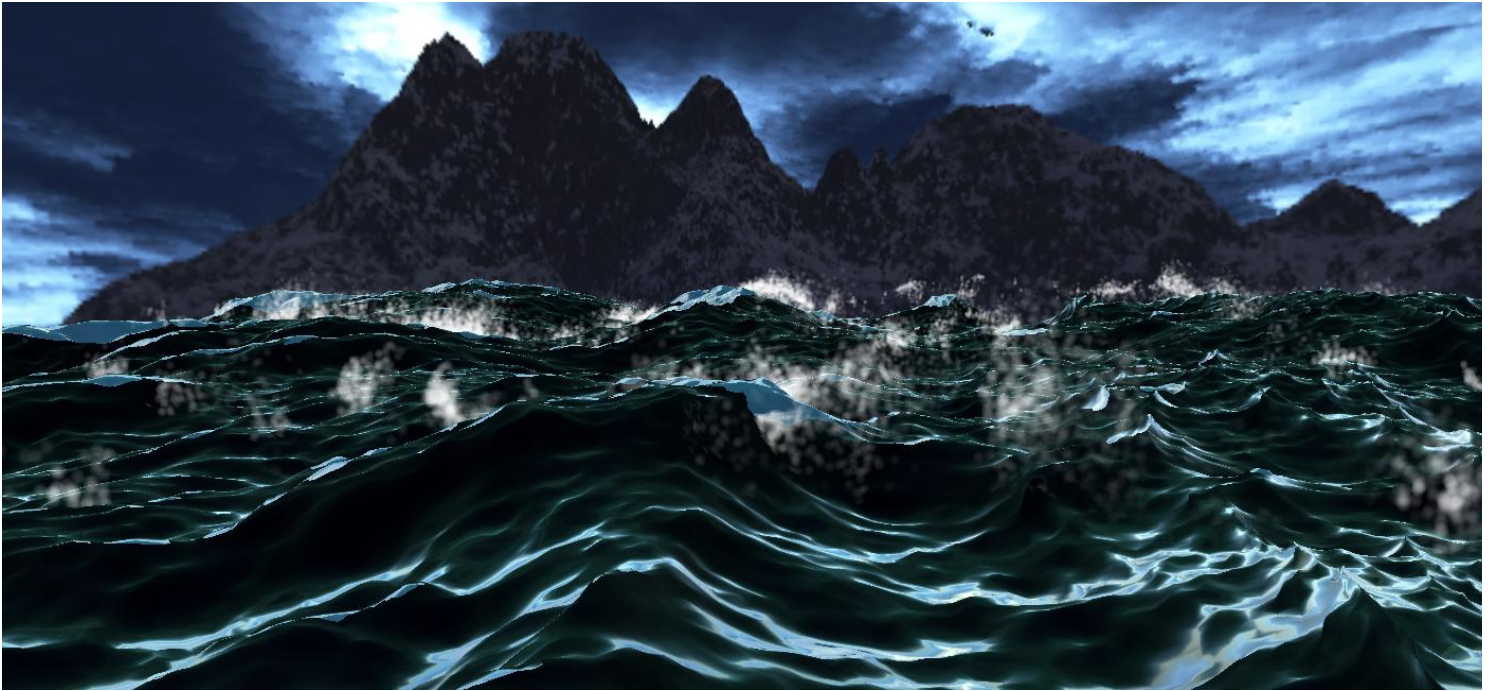
Email:taco.wu@gmail.com

NTU r98944027

顏昭恩

Email:twelalic@gmail.com

NTU r98944033



Abstract

Ocean simulation is a popular topic in computer graphics. Gerstner wave is a common approximation solution to model ocean wave. For realistic purpose, Fast Fourier Transforms is used to implement concept of Gerstner wave. To make ocean more fancy, we used choppy wave to generate sharp wave peak and used particle system to simulate break wave and last used cube map to shade the ocean wave. We implement our system on GPU to simulate ocean wave in real time.

Introduction

In this article, we implement a way associating several methods to simulating

ocean and spray with computer graphics programs. For recent ten years, we can see more and more CG water in our entertainment, especially in many movies, animations, and games. After graphics hardware efficiency increasing and using these resources more simply, we can compute mass information and reach real-time rendering coincide. In our implementation, we use CUDA to reach ideal efficiency when implementing algorithms.

Before we go to detail introduction, we want to express first what we focus on in our implementation. In our rendering result, we don't concern about the interaction of light between water surface and clouds or air. So we

use cube map to reduce the computation on it. Then, we don't concern about the water volume under the waver surface, too. However, we focus on spray detection and production, it's very important to show the spray produced by wave in our implementation.

In section 2, we introduce the algorithms we use to generate the basic wave surface. In section 3, we add choppy wave to make basic wave also move horizontally. After that, we introduce how we detect and produce spray by wave in section 4. Last, we render our result in section 5.

1. Basic Ocean Wave

In this section, we focus on the main algorithms used in our implementation. First, Gerstner waves are introduced in section 2.1. It's the basic and simple wave approximate model. And then, we go to the Fast Fourier Transforms (FFT), the main method to generating the typical waves without storm.

1.1. Gerstner Waves

This solution for wave models was found in 1802. In [Tessendorf 2001], author shows that we can express the water surface as many points on it, and we describe the motion of each point to approximate the real wave motion. The point motion is circular when waves passed by. If a point on the surface is labeled $\mathbf{x}_0 = (x_0, z_0)$ and the height is $y_0 = 0$, we can describe the position of this point at time t when single wave passes with amplitude A by following expressions:

$$\mathbf{x} = \mathbf{x}_0 - \frac{k}{k_i} A \sin(\mathbf{k} \cdot \mathbf{x}_0 - \omega t)$$

.....(1)

$$y = A \cos(\mathbf{k} \cdot \mathbf{x}_0 - \omega t)$$

.....(2)

In these expressions, \mathbf{k} , which is called wavevector, is a horizontal vector that shows the direction of wave traveling. It has a

magnitude k related the wave length λ by

$$k = \frac{2\pi}{\lambda}$$

.....(3)

And ω is the frequency related vector \mathbf{k} .

Gerstner waves are simple but limited because they presents as a single horizontal and vertical sine wave. However, they can present more complicated waves by summing a set of sine waves. Given a set of wavevectors \mathbf{k}_i , amplitude A_i , frequency ω_i , and phase φ_i , we can get following expression:

$$\mathbf{x} = \mathbf{x}_0 - \sum_i \frac{k_i}{k_i} A_i \sin(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t + \varphi_i) \quad (4)$$

$$y = \sum_i \frac{k_i}{k_i} A_i \cos(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t + \varphi_i) \quad (5)$$

1.2. Fast Fourier Transforms

Instead of using Gerstner waves, we choose FFT to implement this ocean simulation because it's a good way for CUDA to compute parallel and CUDA provides library to assist programmer to done related computation.

FFTs represent the wave height $h(\mathbf{x}, t)$ at the position $\mathbf{x} = (x, z)$ and the time t in horizontal plane as the sum of sinusoids with complex, time-dependent amplitudes:

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k} \cdot \mathbf{x}) \quad (6)$$

where $\mathbf{k} = (k_x, k_z)$ is a vector, $k_x = \frac{2\pi n}{L_x}$, $k_z = \frac{2\pi m}{L_z}$, and m, n are both integers with bound $-\frac{N}{2} \leq n < \frac{N}{2}$ and $-\frac{M}{2} \leq m < \frac{M}{2}$, $\tilde{h}(\mathbf{k}, t)$ is the amplitude for each \mathbf{k} at time t . Therefore, we can generate the height field at each discrete point $\mathbf{x} = (\frac{nL_x}{N}, \frac{mL_z}{M})$ for corresponding n, m .

According to equation (6), we should get $\tilde{h}(\mathbf{k}, t)$ in order to receive the height at time t . In "Simulating Ocean Water", author provides an efficient method to get the amplitude. First, we consider this model for spatial spectrum, called *Phillips Spectrum*:

$$P_h(\mathbf{k}) = A \frac{\exp(-1/(kL)^2)}{k^4} |\hat{\mathbf{k}} \cdot \hat{\mathbf{w}}|^2 \quad (7)$$

where $L = V^2/g$ is the largest possible waves arising from a continuous wind of speed V , g is the constant of gravity, and $\hat{\mathbf{w}}$ is the direction of the wind. A is a numeral constant. The term $|\hat{\mathbf{k}} \cdot \hat{\mathbf{w}}|^2$ in spectrum eliminates wave to move vertically to the direction of wind.

Then, we produce the fourier amplitudes of a wave height field with spatial spectrum we have computed:

$$\tilde{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}} (\xi_r + i\xi_i) \sqrt{P_h(\mathbf{k})} \quad (8)$$

where ξ_r and ξ_i are normal independent draws from a gaussian random number generator, with mean 0 and standard deviation 1.

Finally, given a dispersion relation $\omega(\mathbf{k})$, we can get the wave amplitudes at time t :

$$\begin{aligned} \tilde{h}(\mathbf{k}, t) = & \tilde{h}_0(\mathbf{k}) \exp\{i\omega(\mathbf{k})t\} \\ & + \tilde{h}_0^*(-\mathbf{k}) \exp\{-i\omega(\mathbf{k})t\} \end{aligned} \quad (9)$$

This expression preserves the complex conjunction $\tilde{h}^*(\mathbf{k}, t) = \tilde{h}(-\mathbf{k}, t)$ by propagating waves “to the left” and “to the right”. We can compute FFTs results with these amplitudes.

2. Choppy Wave

— without choppy
— with choppy



We get a wave simulation without storm now, but it's not enough for real situation or dramatic effect. Therefore, we add choppy waves into our simulation.

According to “Simulating Ocean Water”, Jt[Tessendorf 2001] uses choppy waves to make the wave peak sharp and the wave bottom flat. we move \mathbf{x} to $\mathbf{x} + \lambda \mathbf{D}(\mathbf{x}, t)$ at time t in order to get sharper wave instead of sine wave, where λ is just a scale to let us control waves conveniently.

And the movement function $\mathbf{D}(\mathbf{x}, t)$ is as following:

$$\mathbf{D}(\mathbf{x}, t) = \sum_{\mathbf{k}} -i \frac{\mathbf{k}}{k} \tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k} \cdot \mathbf{x}) \quad (10)$$

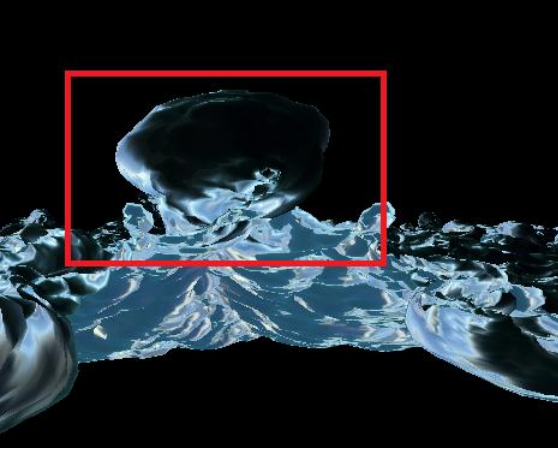
From expression (10), the waveform of the horizontal movement is similar to FFT waveform but shifts forward $3\pi/2$. So, the horizontal movement is lesser when the original FFT wave moves near the peak or trough, but larger when wave moves near the base. It makes our simulating waves much sharper.

However, choppy waves have some problems. When we increase the scale λ of $\mathbf{x} + \lambda \mathbf{D}(\mathbf{x}, t)$, it causes the “overlapping” on the wave simulation. It means the external force is so huge that waves cannot keep their shapes anymore. When wave peak breaks, it generates spray, foam, and splash in real situation. Therefore, we will introduce how to detect the overlapping wave and produce the spray in next section.

3. Spray

In previous section, we found the problem about overlapping on the peak of waves. Now, we can use this property of breaking waves to generate spray.

3.1. Detecting Spray Positions



According to “Simulating Ocean Water”, author[Tessendorf 2001] provides a simple method to detect overlapping by using the Jacobian. The Jacobian has the form:

$$J(\mathbf{x}) = J_{xx}J_{yy} - J_{xy}J_{yx} \dots\dots\dots(11)$$

where

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

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

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

$$J_{yx}(\mathbf{x}) = \lambda \frac{\partial D_y(\mathbf{x})}{\partial x} = J_{xy}(\mathbf{x})$$

and $\mathbf{D} = (D_x, D_y)$ means the function of two directions' amounts of horizontal plane.

The Jacobian is less than zero if the \mathbf{x} is in the overlap region. It means if some point moves to the position of its adjacent point (i.e. $J_{xx}J_{yy}$ is too small. In other words, $\lambda \frac{\partial D_x(\mathbf{x})}{\partial x}$ or $\lambda \frac{\partial D_y(\mathbf{x})}{\partial y}$ is too small.), or in other situation, the adjacent point on x-axis intersects the adjacent point on y-axis (i.e. $J_{xy}J_{yx}$ is too large, on other words, $\lambda \frac{\partial D_x(\mathbf{x})}{\partial y}$ and $\lambda \frac{\partial D_y(\mathbf{x})}{\partial x}$ are both too large or too small.)

Now, we can find the positions with overlapping by Jacobian. Furthermore, we

can use the information of Jacobian to give the initial velocity to spray we prepare to produce.

To get the velocity, we have to get the eigenvalues and eigenvectors from Jacobian matrix. According to “Simulating Ocean Water”, we can count eigenvalues and eigenvectors with following expressions:

$$J_{\pm} = \frac{(J_{xx}+J_{yy})}{2} \pm \frac{((J_{xx}-J_{yy})^2+4J_{xy}^2)^{\frac{1}{2}}}{2} \quad (12)$$

for the eigenvalues.

$$\hat{e}^{\pm} = \frac{(1, q_{\pm})}{\sqrt{1+q_{\pm}^2}} \quad (13)$$

and

$$q_{\pm} = \frac{J_{\pm}-J_{xx}}{J_{xy}} \quad (14)$$

for eigenvectors.

The two eigenvalues, J_+ and J_- , mean the larger and smaller eigenvalue, where $J_- \leq J_+$. If the Jacobian is less than zero, J_- must be less than zero and J_+ must be larger than zero. We can check $J_- < 0$ instead of $J(\mathbf{x}) < 0$, and the eigenvector corresponding J_- is the direction of spray velocity.

3.2. Generating Particle Spray

In our implementation, we choose particle system to present spray from breaking waves.

In each frame, we set the position and velocity of new spray to parts of the particles that are already dead or don't initialize yet. The direction of spray velocity is decided by the eigenvector corresponding smaller eigenvalue, and the scale of spray velocity is decided by a random percentage of $(J_T - J_-)$, where J_T is a fixed number.

After we initialize particles in each frame, we update the data of all living particles, including their velocity, position, and age. In

our case, we just consider the effect of gravity because the force of gravity is much larger than other force generating by particles interacting for a bubble thrown out from the wave surface.

When a particle is dead, we stop updating its information and let it invisible until we set new spray initial information to it.

In our result, we can run 131072 particles at the same time and keep 30 fps with (Intel E2180 2.0GHz and NVIDIA Geforce 9800GT).

4. Rendering

Our rendering step goes as follow: First draw the sky box, whose surrounded whole scene in six pictures with a fine look, then sampling a cube map[OpenGL Cube Map Texturing] as the reflection on water surface. The next step, we take the original grid mesh(256*256) add the height changes calculated from FFT ocean simulation wave, also, the choppy from previous calculation as horizontal movement to form the waves. So far, we have a fine look ocean waves with both vertical and horizontal movement. Finally, render the spray particle as a point sprite, which simply provide a fine look as water particle without complicated calculations.

All the work is done by shader, so that we can keep all the data (waves and particles) in graphic card's memory to save the redundant communication data with CPU.

5. Conclusion

We implement an ocean wave simulator with a basic simulate model with FFT and choppy effect. And detect overlap region to spray particle. With the help of GPU, we makes large scale ocean wave simulation(1024*1024 grid) in real-time (more than 30 fps) and large particles spraying(131072 particles)become

possible. Beyond the works in the project, we found the

GPU provide us the computing power which totally exceed our expectations .The computing power of GPU give us opportunity to simulate complicated Nature phenomenon and render it in real-time.

Reference

- TESSENDORF, J. 2001. Simulating ocean water. In *Simulating Nature: Realistic and Interactive Techniques*. SIGGRAPH 2001
- Course Notes 47.
- OpenGL Cube Map Texturing
- http://developer.nvidia.com/object/cube_map_ogl_tutorial.htm
- 1