Simulating Ocean Surface

Jiashuo Li jiashuol@usc.edu
Xie He xieh@usc.edu
Ting Gong tingg@usc.edu

December 8, 2015

1 Introduction

This is the team project report for CSCI-580 Computer Graphics mentored by Prof. Ulrich Neumann in Fall 2015, at University of Southern California.

Modeling and rendering of ocean surface is a difficult topic in computer graphics. In this project, we simulated a real-time ocean surface.

The core of our algorithm is based on Jerry Tessendorf's algorithm [Tessendorf 2001]. In this model, the ocean surface is a combination of waves with different wave lengths and directions. The height field is then computed by Inverse Fourier Transform.

The lighting method is invented by ourselves to mimic a semi-realistic ocean surface, with just a few basic computations.

The rendered video can be found on YouTube: `https://youtu.be/GruIqDi5Gas`

2 Previous Work

There are plenty of methods to simulate ocean surface.

The simplest way is to use textures to build a static surface, like in the popular game Minecraft. Bump map can also be used to generate the normal field of the surface to get better lighting. However, the result is not very pleasing.

A more complicated way is to generate the surface on-the-fly by synthesizing some waves. [Isidoro et al. 2002] uses four sine waves in vertex shaders to build the height and orientation. [Laeuchli 2002] uses three Gerstner waves instead.

Some more complex methods take the ocean spectra into consideration. There are several famous spectra, such as JONSWAP spectrum, Pierson-Moskowitz spectrum and Phillips spectrum. The height field of the ocean surface can then be computed based on these spectra by direct computation or Fourier Transform.

In our project, we use Tessendorf's model to build the ocean's height field, since Oceanographic literature do not deem Gerstner wave as a realistic model. Therefore, statistical models are used. [Tessendorf 2001]

3 Tessendorf's Algorithm

3.1 Main idea

In this model, the height field is represented by a random function of position and time. $\tilde{h}(\mathbf{k}, t)$ is the Fourier amplitudes of the wave field at time t. By doing an Inverse Fourier Transform, we can get the height of a specific point at $\mathbf{x}$ and time t:

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

The slope can also be computed in a similar form

$$\epsilon(\mathbf{x}, t) = \nabla h(\mathbf{x}, t) = \sum_{\mathbf{k}} i\mathbf{k}\ \tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k} \cdot \mathbf{x}) \tag{2}$$

The normal can thus be computed using the above result.

$\tilde{h}(\mathbf{k}, t)$ is computed using the Phillips spectrum

$$P_k(\mathbf{k}) = A \frac{\exp\left(-1/(kL)\right)^2}{k^4} \left|\hat{\mathbf{k}} \cdot \hat{\omega}\right|^2 \tag{3}$$

where $\hat{\omega}$ is the direction of the wind and $L = V^2/g$, which is decided by the speed of the wind V.

Other details are omitted here. The original paper has more detailed explanation.

3.2 FFTW Library Integration

However, by implementing the above algorithm, the computation is extremely slow. We have to use Fast Fourier Transform to accelerate it.

Let's first look at the formula of Inverse Fourier Transform

$$f(n, m) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{2\pi i \left(\frac{nu}{N} + \frac{mv}{M}\right)} \tag{4}$$

Since equation 1 and equation 2 are expressed in similar form, we can easily compute the final result using an FFT library. Here we choose FFTW library to achieve that.

In Tessendorf's algorithm, the value of $\mathbf{k}$ and $\mathbf{x}$ is expressed as follows.

$$\mathbf{k} = \left(\frac{2\pi n}{L_x} + \frac{2\pi m}{L_z}\right) \tag{5}$$

$$\mathbf{x} = \left(\frac{n_x L_x}{N} + \frac{m_x L_z}{M}\right) \tag{6}$$

Where $-N/2 \leq n \leq N/2$ and $-M/2 \leq m \leq M/2$.

However, we can find that the domain of n, m in equation 5 do not correspond to those in equation 4.

In order to use FFT library, we have to change the domain of variables.

We can expand equation 1 as

$$h(\mathbf{x}, t) = \sum_{n=-N/2}^{N/2} \sum_{m=-M/2}^{M/2} \tilde{h}(\mathbf{x}, t) \exp\left(i2\pi\left(\frac{nn_x}{N} + \frac{mm_x}{M}\right)\right) \tag{7}$$

Let $n = n' - N/2$ and $m = m' - M/2$, equation 7 becomes

$$\begin{aligned} & h(\mathbf{x}, t) \\ &= \sum_{n'=0}^{N} \sum_{m'=0}^{M} \tilde{h}(\mathbf{x}, t) \\ & \exp\left(i2\pi\left(\frac{(n' - N/2)n_x}{N} + \frac{(m' - M/2)m_x}{M}\right)\right) \\ &= \sum_{n'=0}^{N} \sum_{m'=0}^{M} \tilde{h}(\mathbf{x}, t) \\ & \exp\left(i2\pi\left(\frac{n'n_x}{N} + \frac{m'm_x}{M}\right) - i\pi(n_x + m_x)\right) \end{aligned} \tag{8}$$

By directly applying FFTW, treating $\tilde{h}(\mathbf{x}, t)$ as starting from 0, the calculation is

$$h'(\mathbf{x}, t) = \sum_{n'=0}^{N} \sum_{m'=0}^{M} \tilde{h}(\mathbf{x}, t) \exp\left(i2\pi\left(\frac{n'n_x}{N} + \frac{m'm_x}{M}\right)\right) \tag{9}$$

We can just replace $h(\mathbf{x}, t)$ with $h'(\mathbf{x}, t)$ by a translation. However, the term $i\pi(n_x + m_x)$ will flip the sign of the result when $n_x + m_x$ is odd. Therefore, in the final result, vertices with odd sum of x and z coordinate have to get their sign flipped.

Another thing that needs attention is that FFTW computes an unnormalized Discrete Fourier Transform. Thus, computing a forward followed by a backward transform (or vice versa) results in the original array scaled by NM. [10]

3.3 A Mistake in Choppy Wave Formula

In the implementation of Tessendorf's algorithm, we found a mistake in computing the choppy wave equation. In the paper, equation 29 gives:

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

$$\mathbf{x}' = \mathbf{x} + \lambda \mathbf{D}(\mathbf{x}, t) \tag{11}$$

When summing a specific $\mathbf{k}$, let A denote $\mathbf{k} \cdot \mathbf{x}$. Using Euler's equation, equation 10 gives

$$-i\frac{\mathbf{k}}{k}\tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k} \cdot \mathbf{x}) \tag{12}$$

$$= -i\frac{\mathbf{k}}{k}\tilde{h}(\mathbf{k}, t) \cdot (\cos(A) + i\sin(A)) \tag{13}$$

$$= \frac{\mathbf{k}}{k}\tilde{h}(\mathbf{k}, t) \cdot (-i\cos(A) + \sin(A)) \tag{14}$$

When the real part of equation 1 is passing through 0 from positive to negative value, the real part of 12 is reaching its maximum, and equation 11 will lead to less choppy waves, which is against what we want.

Therefore, in equation 10 the minus sign before i is redundant. The correct form is

$$\mathbf{D}(\mathbf{x}, t) = \sum_{\mathbf{k}} i\frac{\mathbf{k}}{k}\tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k} \cdot \mathbf{x}) \tag{15}$$

Another solution is to choose negative value for λ.

3.4 Result

Here we will show the graphical results of the 2-D spectrum and the height field.

By setting $L_x = L_z = 10000m, N = M = 64, \omega = (1, 1), V = 50m/s$ and $A = 3\text{e} - 7$, we can get the spectrum in figure 1.

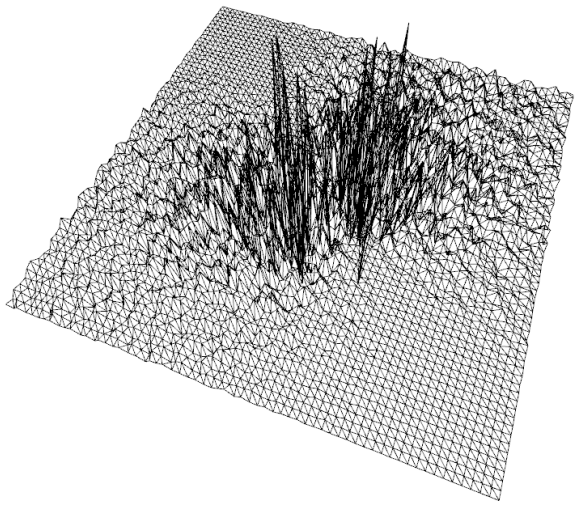


Figure 1: A 2-D spectrum

We can see from the figure that waves that are perpendicular to the wind direction have very small Fourier amplitude. Waves that have the same direction as the wind have large Fourier amplitude.

By setting $L_x = L_z = 1000m, N = M = 64, \omega = (1, 1), V = 50m/s$ and $A = 3\text{e} - 7$, we can get this wireframe of the height field in figure 2.

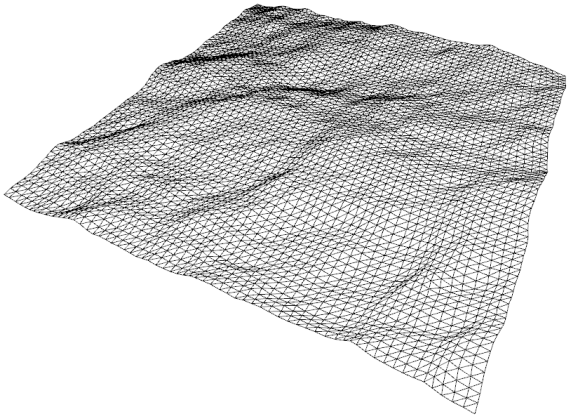


Figure 2: Wireframe

This wireframe is quite realistic but has no details. We can simply increase N and M to get better result, but this will increase the load of the CPU and cannot be accomplished in real-time.

4 Our Lighting

We developed the lighting by ourselves based on Phong-Shading and some empirical observations. The color is computed in 4 steps.

4.1 Height Color

This is the base color of the surface, which is decided by the relative height of every fragment (pixel that is going to be drawn). The height is mapped to $[-1, 1]$ and used to interpolate from deep blue (0.02, 0.05, 0.10) to light green (0.0, 0.64, 0.68).

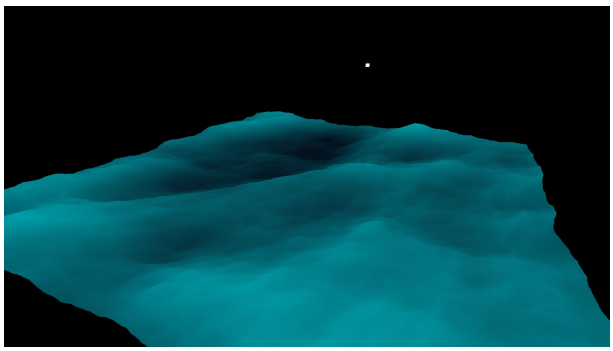


Figure 3: Height color

4.2 Spray Color

The spray color is a pseudo-color that is applied when the height is near the peak, the calculation is similar to height color. However, the combination of this color is not simple addition.

$$\begin{aligned}&\textbf{combinedColor}\\&= (1 - \mathit{SprayCoeff}) \cdot \textbf{combinedColor} + \textbf{spray}\end{aligned}$$

The height color first get attenuated and then combined with the spray light to get a more realistic color transformation.

Figure 4: Spray color

4.3 Pseudo-Reflect Color

The selection of sky color and computation of this color is quite empirical. When we look straight at the ocean surface, the reflection effect tends to be fairly weak. When we look from the side, the reflection gets stronger. Therefore, the computation is

$$\begin{aligned}\mathit{reflCoeff} &= \mathbf{n} \cdot \mathbf{v}\\ \textbf{reflect} &= \textbf{skyColor} \cdot (1 - \mathit{reflCoeff}^{C})\end{aligned}$$

where $\mathbf{n}$ is the normal of a fragment and $\mathbf{v}$ is the view direction pointing at the camera, and C is a constant like shininess in Phong-Shading. In our experiment, 0.3 can be a feasible value.

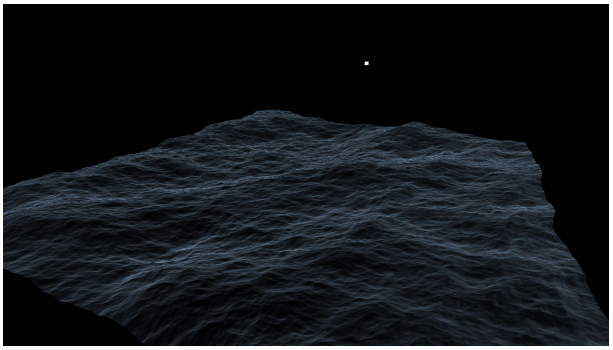


Figure 5: Pseudo-reflect color

4.4 Specular Color

The specular color is the same as that in Phong-Shading. The only difference is how to combine it into other colors.

We use the following formula

$$\textbf{combinedColor} = (1 - \textit{specCoeff}) \cdot \textbf{combinedColor} + \textbf{specular}$$

The computation is similar to that of spray color. With such calculation, we can get a more realistic effect instead of plastic-like highlight.

Figure 6: Specular color

4.5 Combined Color

The colors described above are accumulated step by step to get the final result, which is shown in the last few pages.

5 Framework and Tools

The framework we used is from `http://www.learnopengl.com/`, which includes basic camera placement and movement calculations.

The window system is GLFW library and OpenGL Shading Language framework is GLEW.

The Tessendorf algorithm, our lighting method and BRDF method are totally implemented by ourselves without using any other library. As described above, the FFT computation is performed by FFTW library.

We also use SOIL library to capture a dedicated framebuffer and save the result from `glGetPixels` to BMP files. It costs 4 hours to capture 1800 screenshots with $N = M = 1024$ at 1920x1080 resolution and 16x antialiasing. Then FFmpeg is used to convert the BMP files to a MP4 file.

Finally, this paper is composed using LaTeX.

6 Future Work

In this project we use FFTW library to compute the FFT, but the performance is not very satisfactory when the resolution of the mesh gets higher, i.e. N and M get larger, since the computation is totally done on CPU. The largest N and M for a real-time computation is 64 on a Intel Core i3 M350 2.27GHz CPU. A feasible method to increase the performance is to use cuFFT in CUDA library provided by Nvidia. cuFFT uses GPU to compute the FFT and is much faster than FFTW's CPU computation.

The model is built using a regular mesh. The problem is that vertices that are far away from the camera get mixed and make the scene blurred. Vertices that are close to the camera will make the scene have low sampling rate. One solution is to sample the surface adaptively by increasing the sampling rate for parts of the ocean surface that are close to the camera and decrease the sampling rate for those that are far away.

For our lighting method, the whole light procedure is pseudo and based on simple observations. The spray color can be improved by computing the local maxima and applying some more realistic textures.

For BRDF lighting method, we did not have enough time to figure out the meaning of every formula. If we have more time on it, this model will generate better result.

References

[1] Jerry Tessendorf, *Simulating Ocean Water*, ACM SIGGRAPH course notes, 2001.

[2] Eric Bruneton, Fabrice Neyret, Nicolas Holzschuch, *Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF*, Computer Graphics Forum, Wiley, 2010, 29 (2), pp.487-496.

[3] Christopher J. Horvath, *Empirical Directional Wave Spectra for Computer Graphics*, DigiPro '15 Proceedings of the 2015 Symposium on Digital Production, 2015, Pages 29-39.

[4] John Isidoro, Alex Vlachos, and Chris Brennan *Rendering Ocean Water*, 2002.

[5] Jesse Laeuchli, *Simple Gerstner Wave Cg Shader*, 2002.

[6] Hasselmann K., T.P. Barnett, E. Bouws, H. Carlson, D.E. Cartwright, K. Enke, J.A. Ewing, H. Gienapp, D.E. Hasselmann, P. Kruseman, A. Meerburg, P. Mller, D.J. Olbers, K. Richter, W. Sell, and H. Walden. *Measurements of wind-wave growth and swell decay during the Joint North Sea Wave Project (JONSWAP)*, Ergnzungsheft zur Deutschen Hydrographischen Zeitschrift Reihe, A(8) (Nr. 12), p.95, 1973.

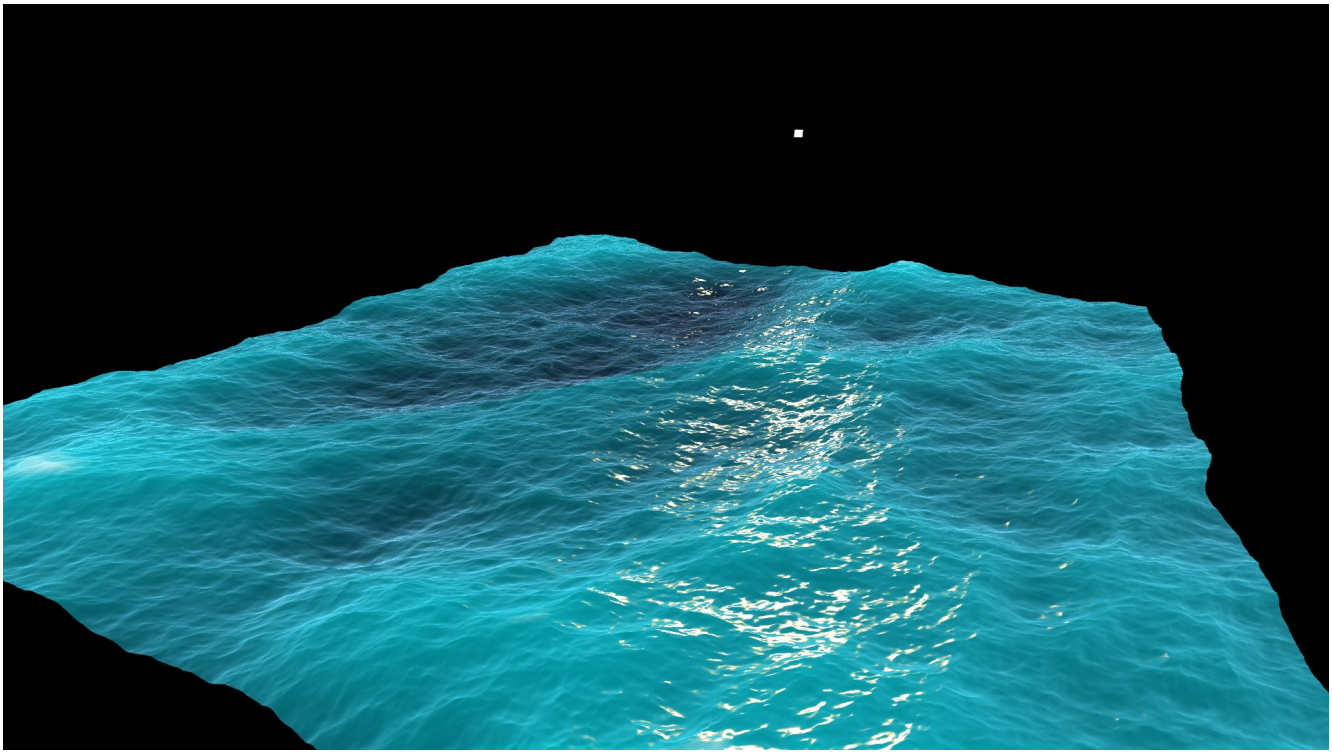
[7] Ross V., Dion D., Potvin G., *Detailed analytical approach to the gaussian surface bidirectional reflectance distribution function specular component applied to the sea surface*, Journal of Optical Society of America A 22 (Nov. 2005), 2442-2453.

[8] Smith B., *Geometrical shadowing of a random rough surface*, IEEE Transactions on Antennas and Propagation 15 (Sept. 1967), 668-671.

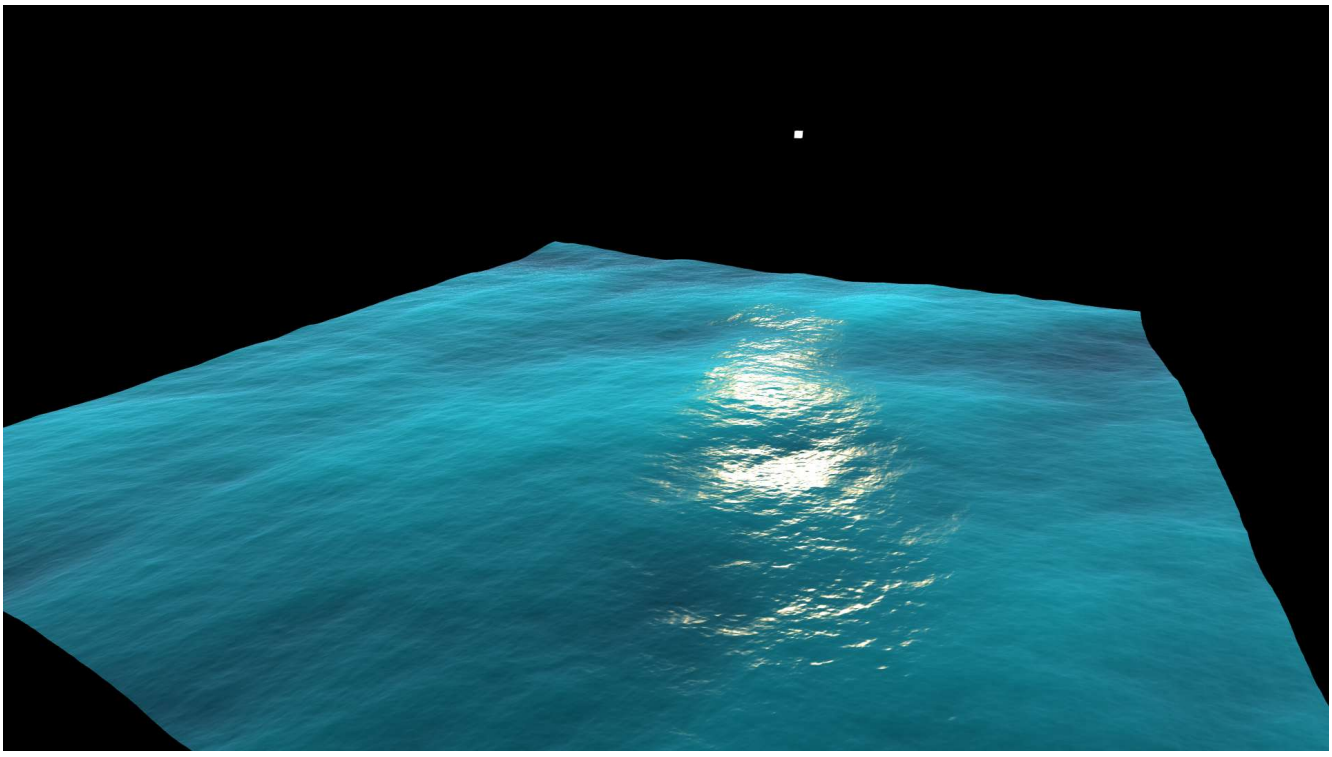
[9] Schilick C., *An inexpensive BRDF model for physically-based rendering*, Computer Graphics Forum 13 (1994), 233-246.

[10] FFTW documentation, `http://www.fftw.org/fftw3_doc/The-1d-Discrete-Fourier-Transform-_0028DFT_0029.html`

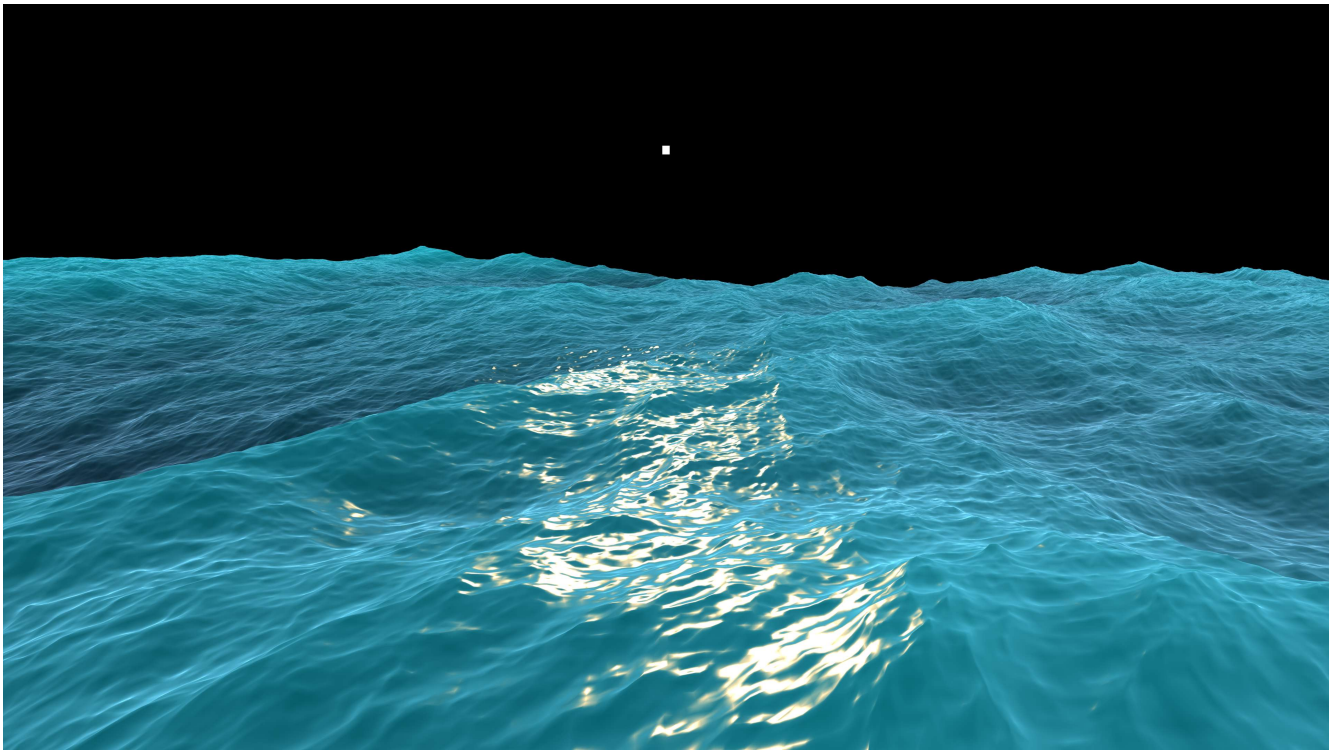
[11] Effective Water Simulation from Physical Models, `http://http.developer.nvidia.com/GPUGems/gpugems_ch01.html`



Wavy Surface



Calm Surface



Local View (3840 x 2160 resolution!)