# A Frustum-Based Ocean Rendering Algorithm

Ho-Min Lee, Christian Anthony L. Go, and Won-Hyung Lee

Chung-Ang University,
Department of Image Engineering,
Graduate School of Advanced Imaging Science and Multimedia and Film,
221 Hukseok-Dong, Dongjak-Gu, Seoul, Korea
grancia@gmail.com, chipgo@gmail.com, whlee@cau.ac.kr

**Abstract.** Real-time water rendering has always posed a challenge to developers. Most algorithms concentrate on rendering small bodies of water such as pools and rivers. In this paper, we proposed a real-time rendering method for large water surfaces, such as oceans. This algorithm harnesses both the PC and GPU's processing power to deliver improved computing efficiency while, at the same time, realistically and efficiently simulating a large body of water. The frustum-based algorithm accomplishes this by representing a smooth water surface as a height value of the viewer, since surface size can be fluidly calculated given the camera frustum position. This algorithm has numerous potential applications in both the gaming and the movie industry. Experimental results show a marked improvement in computing power and increased realism in large surface areas.

## 1 Introduction

Together with the GPU's emergence in the last couple of years, real-time water rendering has likewise evolved. Early rendering techniques resulted in water surfaces that were not entirely realistic due to algorithm and processing power constraints. Today's GPU's however, with their increased computing power, afford developers added flexibility to render large bodies of water in real time.

In order to create realistic large water surfaces in real-time rendering, three components need to be addressed:

1. **Wave generation:** The height value of the water surface should be calculated in real-time, because waves cause the ocean to have a dynamic surface[1].

2. **Reflection and Refraction:** The overall look of water is based on reflected and refracted light. Both reflected and refracted light differ depending on the angle the surface viewed at.

3. **Water surface size:** A larger scale rendering must be done, taking into account height and point of view since the ocean represents a large body of water.

Z. Shi and R. Sadananda (Eds.): PRIMA 2006, LNAI 4088, pp. 584–589, 2006.

Among the three factors, for large-scale water rendering, we must primarily address surface size because the bigger the surface size, the more computing power needed to render it. Inefficient rendering will result in decreased performance and an overall decrease in speed when applied to games. This paper introduces an alternative, efficient approach for realistic large-scale water rendering.

This paper is organized as follows. Previous works of the real-time water rendering are described in Section 2. In Section 3, the proposed algorithm and the experimental results are shown. The conclusions of our research are stated in Section 4.

## 2 Water Surface Representation

### 2.1 Expressing Body of Water as Volume with Perlin Noise

The two most common algorithms employed for water surface generation are the Navier-Stokes and Perlin Noise Equations. While the Navier-Stokes Equation is the most commonly implemented of the two[7], it is, however, limited by its inability to efficiently render large bodies of water due to its high computational requirements[2]. Thus, to model large bodies of water, this paper utilizes the Perlin Noise Equation which creates a continuous noise. Two functions are necessary in order to create a Perlin Noise, a noise function, and an interpolation function[5]. A noise function is simply a seeded random number generator which takes an integer as a parameter, and returns random number based on that parameter. A continuous function can then be defined by interpolating the parameters. Fig. 1 graphically illustrates the processing[3]. Perlin noise can be

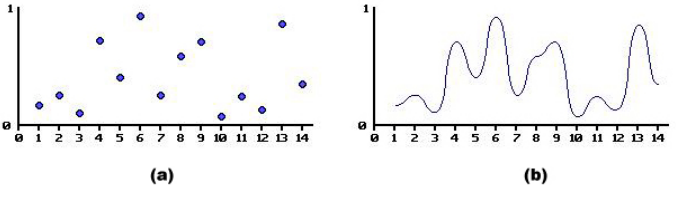


**Fig. 1.** (a)Demonstration of random numbers in coordinates (b)Demonstration of interpolated curve using random numbers

created using noise functions. When diverse noise functions are combined, a new noise patter can be achieved.

### 2.2 Reflection and Refraction

When light strikes a water surface, it is both reflected from and refracted into the surface. To realistically replicate his, we employ Shader programming HLSL (GPU). Fig. 2 Shows reflected light and refracted light on a water surface.

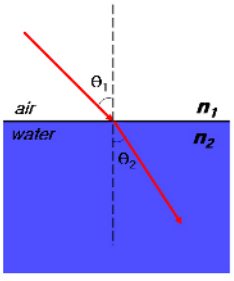


**Fig. 2.** Illustration of reflected and refracted light by Snell'law

Equation (1) demonstrates Snell's law which is used to calculate for reflection and refraction on water surface. We get $\theta_1$ and $\theta_2$ from the equation, and we can get realistic water surface which is applied reflected light and refracted light.

$$\frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1} \tag{1}$$

## 3 Frustum-Based Algorithm

### 3.1 Water Surface Size

In real-time water rendering, a large water surface requires high processing power[8]. To reduce processing requirements and increase rendering efficiency for such large surfaces, we use shader programming and propose a new algortihm. This algorithm improves the projected grid algorithm by Claes Johanson[4]. The projected grid algorithm is sufficient when the water surface is viewed from a low viewpoint, however the water surface is not correctly displayed when the viewpoint (i.e. height value) of a viewer is increases. As a result, the water surface appears cut and looks smaller than it should. This is a serious problem when expressing a large water surface. Moreover, speed is an issue as it is not feasible to apply it to games which use about 30 fps.

### 3.2 Render Water Surface

The water surface is rendered within the rendering frustum and surface size is defined by viewer's frustum which is a function of the rendering frustum. Equations (2) and (3) describe how the water surface is defined using viewer height value.

$$W_{range} = V_{HF} \cdot sin\frac{3}{2}\theta \cdot \min(W_{range}), \qquad (\theta < 60) \tag{2}$$

$$W_{range} = \max(W_{range}). \qquad (\theta > 60) \tag{3}$$

Where $V_{HF}$ is orthogonal height value from the water surface and $\theta$ represents the angle between the surface and the user viewpoint. When viewing the

ocean from a beach, minimal water surface can be seen on the horizon. The $min(W_{range})$ value represents this water surface size. However, as we increase viewpoint height from a beach to that of an airplane, the ocean surface consequently increases on the horizon. After a certain vertical threshold, increases in height will no longer have any effect on surface area on the horizon. $Max(W_{range})$ value denotes this limit on surface size. Fig 3. illustrates the water surface as height value, Fig. 3(a) is screenshot of the water surface with a height value of 60, while Fig. 3(b) is screenshot with a height value parameter of 800. Although,

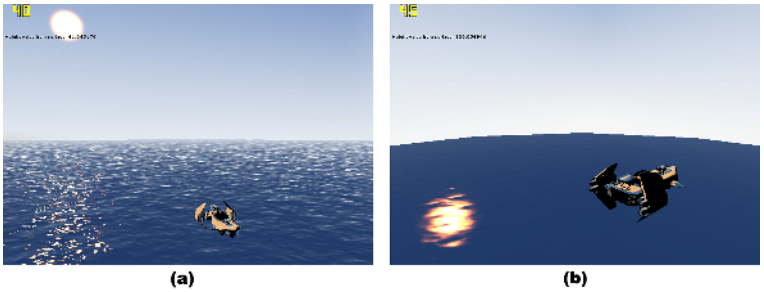


**Fig. 3.** Water surface rendering as height value

rendering size is substantially increased, our algorithm demonstrates efficient rendering with a refresh rate of approximate 40fps. This is true for both micro and macro views of the rendered ocean. An increased efficiency in grid calculation results in a significant improvement in rendering speed. The traditional projected grid algorithm creates a grid size of 128 by 256[4], conversely, our improved algorithm calculates grid size as a binding of neighboring grid coordinates, thus resulting in improved speed. Equation (4) shows how the grid coordinates bind with neighboring grid coordinates.

$$G_{bind} = \sum_{i=m} \sum_{j=n} \frac{\overrightarrow{c}_{i,j} + \overrightarrow{c}_{i+1,j+1}}{2} \tag{4}$$

### 3.3 Definition of Water Rendering Size

In our algorithm, we utilize two frustums for water rendering, one for the viewer and the other for the water body. Rendering of the body of water is done in detail within the frustum which is defined in post-perspective space. Areas outside of the camera frustum are not rendered. This is to minimize processing overhead. Fig. 4 shows how a surface can be presented in post-perspective. As described in Fig. 4, the water surface is rendered by calculating the transposition area of the camera frustum and surface. $\theta$ which is described in Equation (2), (3) is the angle of the water surface and the camera viewpoint. And although the size of rendering surface is getting bigger as viewpoint and $\theta$, this algorithm shows efficient computing power because we utilize the LOD algorithm[9].

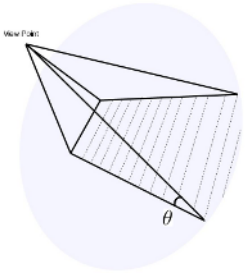


**Fig. 4.** Define a range for water rendering in post-perspective space

### 3.4 Experimental Results

We use common factors to achieve a realistic, believable water surface, these are reflection, refraction and waves. We, at the same time, minimize processing overhead to facilitate the rendering of a large mass of water. Fig 5. shows an ocean rendering using our algorithm. A 40% improvement in computing efficiency was achieved. Experiments were performed on a Geforce 6600 GPU with Pentium 4 processor running at 3.0 GHz. In this environment, we demonstrate enough efficiency to apply this method of water simulation to a game.

Comparing the two algorithms, Fig. 5(a) presents the projected grid algorithm and Fig. 5(b) is our algorithm. Both of these use the same height value of 2130, but the projected grid algorithm cuts the water surface and displays a part of earth surface, because it can no longer be rendered. We can clearly distinguish the improvement over the traditional method.

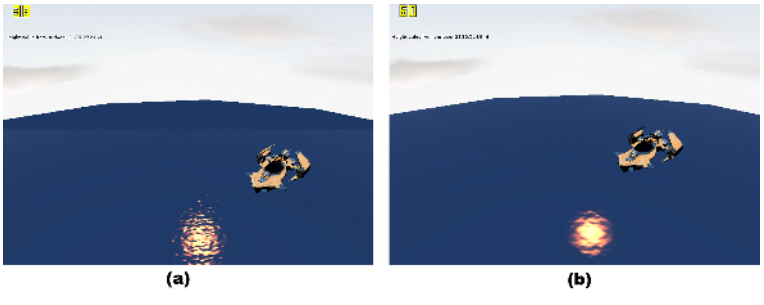


**Fig. 5.** (a) Rendering result by the projected grid algorithm (approximate height value = 2130) (b) Rendering result by our algorithm (approximate height value= 2130)

## 4 Conclusion

We have presented an efficient method for large water surface rendering using a frustum-based algorithm. Realistic water rendering has always posed a challenge

to developers. The tradeoff was always a compromise between computing power and size of the body of water. Realistic simulations are no longer the exception but have become the norm with the continuing advances in technology. Our algorithm addresses these demands by giving the developer the freedom to model a large body of water such as an ocean without having to worry about impossibly high computational demands.

## Acknowledgment

This research was supported by the Korea Culture and Contents Agency at the Culture Contents Technology Venture Center.

## References

1. Simon Premoze, Michael Ashikhmin.: Rendering Natural Waters. IEEE Computer Society. (2000) 23
2. Jim X. Chen , Niels da Vitoria Lobo.: Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations. Graphical Models and Image Processing. **57** (1995) 107–116
3. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley.: Texturing & Modeling - A Procedural Approach Second Edition. AP Professional. (1998)
4. Claes Johanson.: Real-time water rendering. Lund University. (2004)
5. K. Perlin.: An image synthesizer. In B. A. Barsky (ed). Computer Graphics (SIGGRAPH '85 Proceedings). **19** (1985) 287–296
6. John C. hart.: Perlin noise pixel shaders. ACM Trans Graphics(SIGGRAPH Proc) (2001) 87–94
7. James F. O'Brien, Jessica K, Hodgins.: Dynamic Simulation of Splaching Fluids. Proceedings of computer Animation. (1995) 198–205
8. Ping Wang, Daniel S. Katz, Yi Chao.: Optimization of a parallel ocean general circulation model. Proceedings of ACM on Supercomputing. (1997) 1–11
9. BLOW, J.: Terrain rendering at high levels of detail. Game Developers Conference. (2000)