ICIC Express Letters
Part B: Applications
Volume **7**, Number **9**, September **2016**

 ISSN 2185-2766
pp. 1975–1981

REAL-TIME RENDERING FOR SEA WATER BASED ON HEIGHT FIELD MODEL AND LIGHT INTERACTION

Jie Xu[1], Yang Zhang[2] and Jinhua Fu[3,4,*]

[1]School of Software
[2]College of Mathematics and Information Science
[4]School of Computer and Communication Engineering
Zhengzhou University of Light Industry
No. 5, Dongfeng Road, Zhengzhou 450002, P. R. China
*Corresponding author: fjhzzuli@qq.com

[3]State Key-Laboratory of Mathematical Engineering and Advanced Computing
Zhengzhou Information Science and Technology Institute
Zhengzhou 450000, P. R. China

Received February 2016; accepted May 2016

Abstract. *Real-time simulation of liquids like sea water is an important task nowadays in the field of computer graphics. In this paper, combined with simulation of light interaction, rendering of sea water at interactive rates is implemented based on the height field model. The focus of the sea rendering is a real-time and graphically realistic simulation, capable of handling different types of illumination such as refraction and reflection. In order to improve the realism of sea water, surface details are efficiently acquired by normal mapping. The effects of lighting and shading via our method are more obvious and clear in such as reflection, refraction, foam and soft shadow, which are in accord with physical interaction between light and water.*
Keywords: Sea water, Light interaction, Height field model, Refraction, Reflection

1. **Introduction.** Simulation of large-scale water is essential in virtual reality, with wide applications in 3D games, movie special effects, etc. As it involves much physical computation such as reflection, refraction and details simulation, how to achieve faster rendering while keeping visually plausible appearance is still a challenging task.

There have been many methods for simulation of sea water in the past. Particle-based fluid simulation methods [1, 2] can be easily realized, but these methods are used to create 2D shallow water models and only give lower quality of rendering appearance. In order to improve rendering quality, Clavet et al. [3] attempt to improve the particle-based simulation by using viscosity. However, they only achieve realistic small-scale behavior of substances such as paint or water as they splash on moving objects. For large-scale dynamic river motion phenomenon, Shi et al. [4] present a combination of modeling methods for flood routing process simulation. Their methods allow the user to interactively fly over a virtual valley, and present results of implementation of the 3D visualization for river flood routing simulation. Considering physical interaction between light and water, optical properties of water are simulated by Algan et al. [5] and Shi et al. [6]. They approximate water drops and polluted water respectively for especial lighting conditions. Recently, with the increasing requirement of modern 3D game scenes, the main challenge of water rendering is not only to simulate realistic appearance, but also develop sufficiently simple method to allow for feasible computation on the GPU (Graphics Processing Unit). Using the GPU rendering pipeline, Liu and Xiong [7] propose a fast and stable simulation of virtual water scenes, but the rendering speed is still a research direction in their future work. Oh [8] also introduces a novel practical single-phase particle simulation for trapped air bubbles in a turbulent water flow using GPU. More recently, in order to simulate

the transition between different types of flows, Lim et al. [9] model these transitions by constructing a very smooth fluid surface and a much rougher, splashy surface separately, and then blend them together in proportions that depend on the flow speed. Chlădek and Ďurikovič [10] propose a shallow water simulation using a Lagrangian technique and smoothed particle hydrodynamics are used to solve the shallow water equation. Motivated by meeting the 3D game requirements to rendering speed, our research focuses on faster GPU-based rendering of sea water. We simulate surface of sea water via height field model and improve realism of sea water surface via approximation of interaction between light and water. Since normal mapping is a way of adding high-resolution detail to low-polygon objects, it is especially useful for real-time display devices such as 3D game engines. We also acquire the details of sea surface via normal mapping on the GPU. The simulation results indicate that we can achieve rendering at interactive rates while keeping visually plausible appearance of water.

The rest of this paper is organized as follows. The surface representation of sea water is introduced in Section 2. Simulation for optical properties of sea water is given in Section 3. Detailed surface simulation by normal mapping is provided in Section 4. The results are discussed in Section 5, and conclusions are drawn in Section 6.

2. **Surface Representation via Height Field Model.** To simulate the realistic surface of sea water, the normal method is using the 3D grids, which can be precise but also involves quite heavy-weight computation as well. In this paper, we use a 2D grid to represent the sea surface by a height field function. The use of 2D grids is not as accurate as the use of 3D grids but it does provide us with a much simpler way to represent the water surface.

As mentioned earlier that initially in 3D games, water is just rendered as a plane surface which results in very low realism. To depict the complex movements at the surface of the water, $h_{surface}(x, y)$, a model of height-field fluid is introduced here. The height of a fluid is represented as a function of two parameters x and y as follows.

$$h_{surface}(x, y) = h_{plane}(x, y) + H(x, y) \cdot N_{plane} \tag{1}$$

where h_{plane} is the base xy-plane. The function $H(x, y)$ is called the height-field function which decides how the surface will vary with x and y. N_{plane} is a unit vector in the direction perpendicular to the xy-plane. In this paper, the fluid surface is represented by a 2D function $H(x, y)$ as shown in Figure 1.

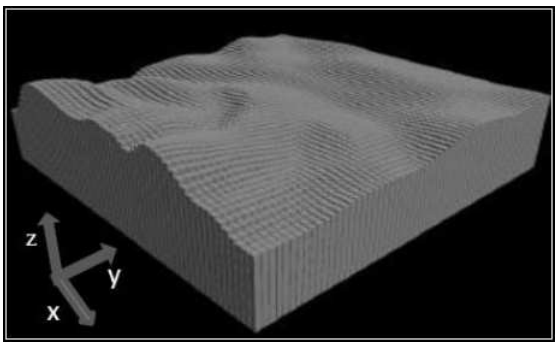


FIGURE 1. Representing a fluid surface as a 2D function $H(x, y)$

3. **Improving Realism of Sea Water Surface via Light Interaction.** For realistic water rendering it is essential to handle the physical interaction between the water surface and light correctly. This realism can be achieved if reflections, refractions and diffusion effects are computed, and if the Fresnel equation is used to calculate the intensity of the reflected and refracted light rays.

3.1. **Reflection effect.** When looking at the surface of water, the surface will reflect either the sky or other objects. In computer graphics this is given by the formula:

$$R = I - 2N(N \cdot I) \tag{2}$$

which dictates that the angle between the camera ray, I, and the surface normal, N, is equal to the angle between the surface normal and the reflected vector, R. This is also depicted in Figure 2(a).

3.2. **Refraction effect.** Refraction is what happens when a ray of light moves through translucent materials with different densities (e.g., sea water and glass). A popular explanation is that the light travels slower in materials with large densities and vice versa, and so the direction of the ray is changed. The equation for refraction is given by Snell's Law in the following equation:

$$\eta_1 \sin(\theta_I) = \eta_2 \sin(\theta_T) \tag{3}$$

where η_1 and η_2 are the refractive indices for two materials, θ_I is the angle between the camera ray I and the surface normal N, and θ_T is the angle between the surface normal below the surface $-N$ and the refracted vector T, also called *transmitted*, as depicted in Figure 2(b).

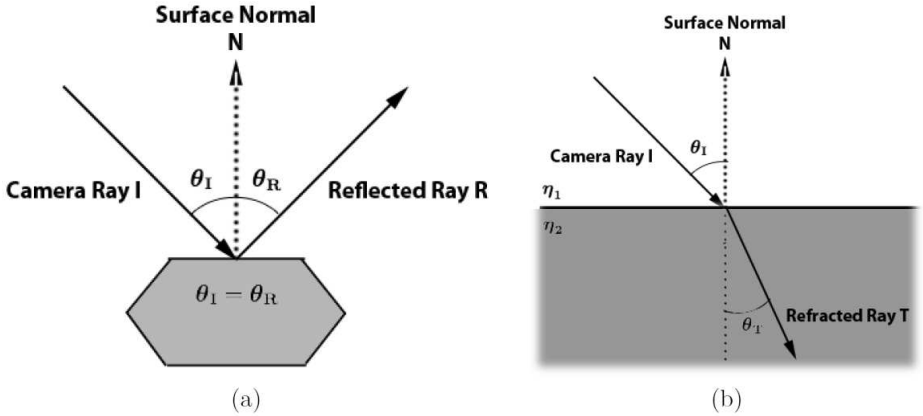


FIGURE 2. The description of reflection process (a) and refraction process (b)

3.3. **Fresnel term.** The Fresnel term F describes the reflective or refractive behavior of the light when it reaches water surface. In this paper, we use the following equation

$$F(\beta) = F_0 + (1 - F_0)(1 - \cos\beta)^5 = F_0 + (1 - F_0)(1 - (V \cdot H))^5 \tag{4}$$

where F_0 is the specular reflectance when light arrives perpendicularly to the surface. β is the angle between the normal N and the half vector H, which can be evaluated as $H = (L + V)/\|(L + V)\|$. L is the incoming light direction. V is the viewing direction.

3.4. **Diffusion effect.** To compute diffuse lighting efficiently, our shading model starts with the same Lambertian diffuse shading as Blinn-Phong model, but it models the surface's microstructure. The microfacet model used in the physical shading model assumes that the surface consists of randomly aligned small smooth planar facets. The diffuse term also takes the Fresnel refraction-reflection term and the shadowing effect of the microfacets into account as follows.

$$I_{diffuse} = \left(k_s \frac{F(\beta)}{\pi} \frac{D(\alpha_h)G}{(N \cdot V)(N \cdot L)} \right) \times \max(N \cdot L, 0) \times I/r^2 \tag{5}$$

where F is the Fresnel term. D is the microfacet distribution. G is the geometric attenuation. k_s is specular reflectance of the surface. α_h is the angle between the viewing direction V and the half vector H.

To evaluate microfacet term D, we use a physically based model of microfacet distribution called Beckmann distribution function which can be defined as

$$D(\alpha_h) = \frac{e^{\lambda}}{\pi m^2 \cos^4 \alpha_h} \tag{6}$$

where m is the RMS (Root Mean Square) slope of the surface microfacets (the roughness of the material). λ is the wavelength of incoming light.

4. **Surface Details Simulation by Normal Mapping on the GPU.** Creating detailed models with thousands of polygons is a time consuming job and rendering them in real-time can be also problematic. However, with normal mapping, we can make simple, low resolution models look like highly detailed ones. In addition to the polygon model, we provide a high resolution normal map which we can use in the GPU fragment shader instead of the interpolated normals, adding detail to the low resolution model. Normal maps can be generated procedurally in the GPU shaders or read from an image and used as a texture.

In this paper, we implement a normal map texture generator which generates normals for a sphere which has smaller flat disc on its surface. This texture will be used in the GPU fragment shader to retrieve normals instead of using the interpolated normals. The normal map texture generator should have two parameters, and two normal map textures with different *bumpRadius* and *resolution* are shown in Figure 3 and Figure 4 as follows.

• **resolution**: The number of flat discs for each row and column on the generated normal map.

• **bumpRadius**: The radius of the flat discs. If $bumpRadius = 0.5$, the discs are tangent to each other. If it is larger, they intersect with each other. If $bumpRadius \geq 1.0$, the sphere looks like it has not smooth but rectangular surface.

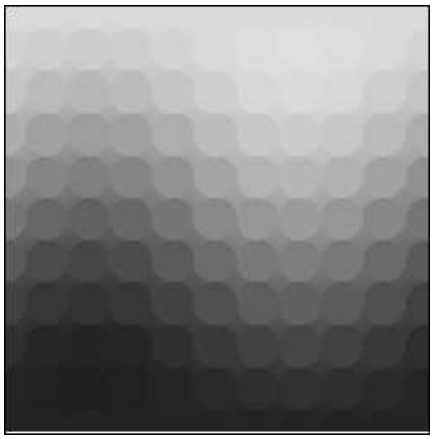


FIGURE 3. The normal map texture with $bumpRadius = 0.5$ and $resolution = 10$

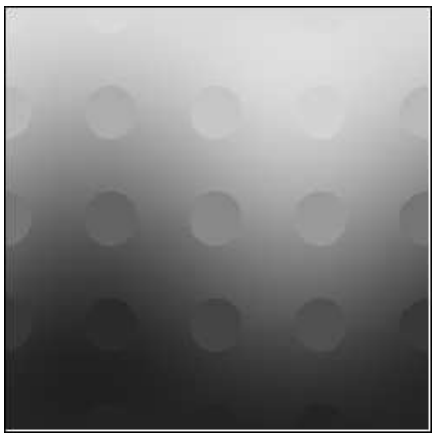


FIGURE 4. The normal map texture with $bumpRadius = 0.25$ and $resolution = 4$

We can implement the *getColor* function of the $TexGenSphereNormalMap$ class. This function will be called with u and v sweeping from 0 to 1 and the returned colors will be saved to a texture, which will be passed later to the GPU shader.

5. **Rendering Results.** We have implemented the fast rendering of sea water on the GPU. All experiments are run on a PC with AMD Athlon II X4 Four Cores, and NVIDIA GeForce GT430. The software platform is based on MS Visual Studio 2012 and OpenGL as the programming language. We also employ GLSL (OpenGL Shading Language) as vertex and fragment shader language on the GPU. We have given rendering effects of sea

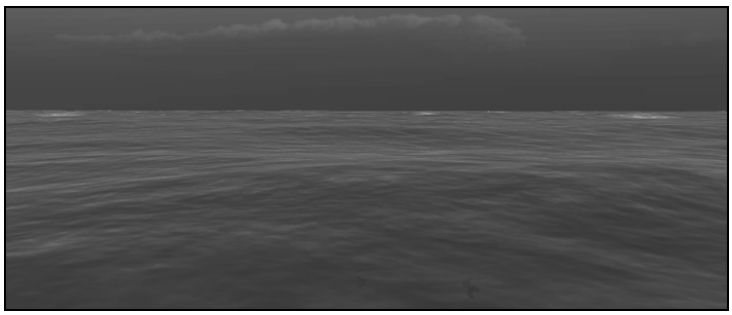
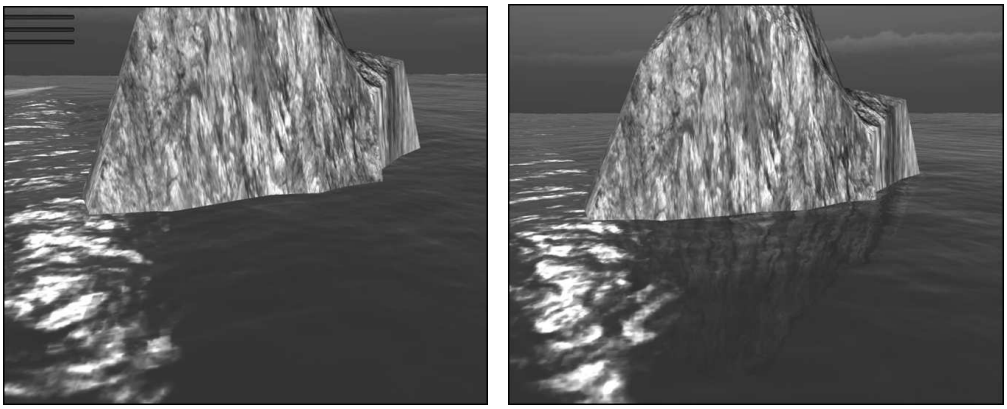
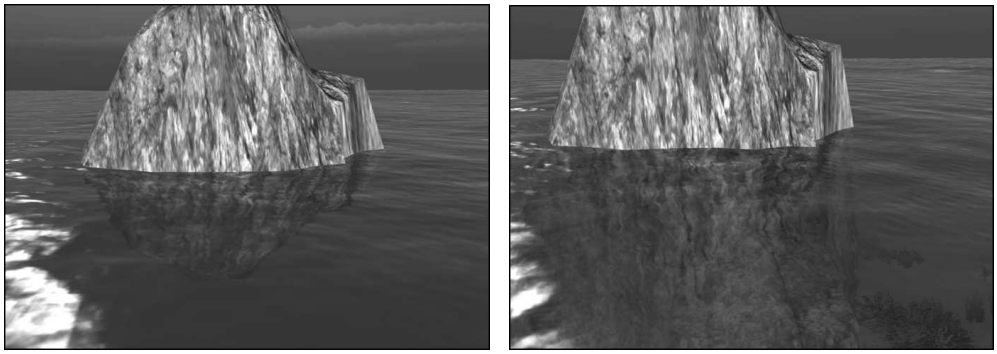


FIGURE 5. Realistic appearance of sea water



(a) (b)

FIGURE 6. Rendering results comparison between no reflection (a) and adding reflection (b)



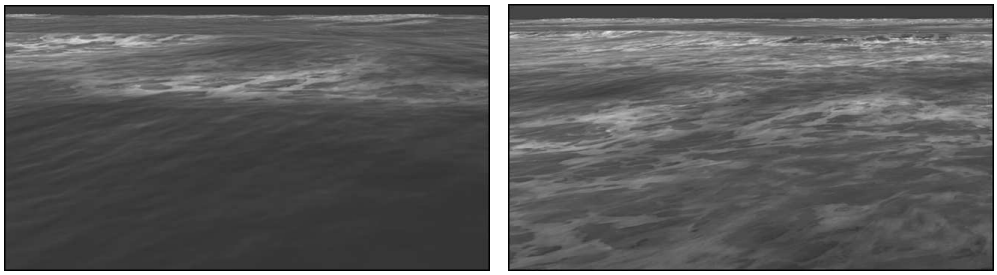
(a) (b)

FIGURE 7. Rendering results comparison between no refraction (a) and adding refraction (b)

water and compare the rendering effects under different illumination conditions. Our final rendering clearly shows the effective implementation of our technique.

We firstly implement realistic rendering of sea water using height field model, and acquire a realistic appearance as shown in Figure 5.

It is important to simulate how light behaves when it interacts with water in reality, because that is what mainly decides how it is going to look to sea water. In order to give clear effects of reflection and refraction, we simulate rendering scenes at different time and illumination conditions separately.



(a) (b)

FIGURE 8. Rendering results comparison between no normal mapping (a) and adding normal mapping (b)

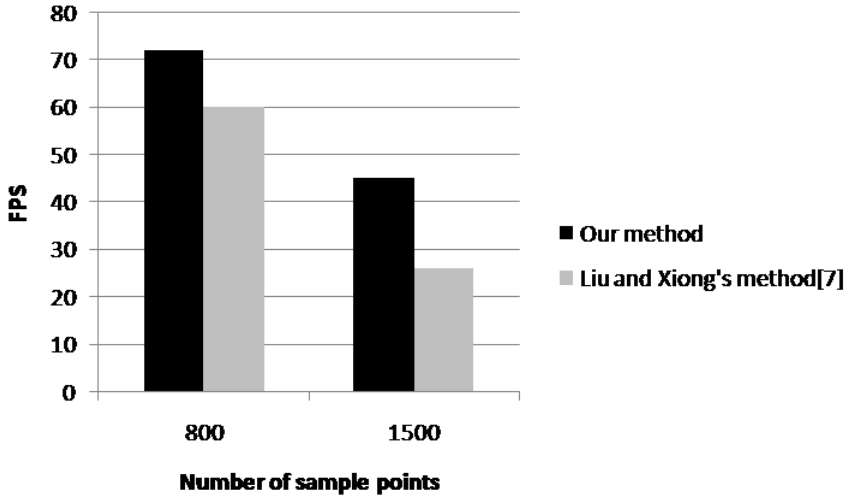


FIGURE 9. Comparison of rendering speed

Firstly, we show different scenes of reflection and refraction to express more realistic effects using our method as shown in Figure 6 and Figure 7.

Normal mapping is only used to fake high-resolution details on a low-resolution model to simulate surface details, which has no relation with reflection and refraction. Each pixel of the map stores the surface slope of the original high-res mesh at that point. This can create the illusion of more surface details of sea water as shown in Figure 8.

From Figure 6, Figure 7 and Figure 8, we can observe that the rendering appearance of sea water using our method is more realistic in such as reflection, refraction, caustics, foam and soft shadow, which are in accord with physical interaction between light and water. So we can simulate the realistic appearance that when a ray hits the water surface. Also, surface details of sea water such as foam are achieved by normal mapping. The effect of lighting and shading via our method is more obvious and clear, which shows our method is very effective.

In the performance of rendering speed, using rendering method on the GPU we have obtained about 72 frames per second (FPS) when rendering 800 points, which is a relatively high speed and suitable for 3D game scenes at interactive rates (>60 FPS). However, the real-time rendering method such as Liu and Xiong's method [7] only acquires 60 FPS. The difference of rendering speed between our method and the method [7] is more obvious when increasing rendering points greatly as shown in Figure 9.

6. **Conclusions.** We have implemented the fast rendering of sea water on the GPU. The rendering results show that our rendering method can obtain realistic effects of sea water in such as reflection, refraction, caustics, foam and soft shadow, which are in accord

with physical phenomena. Also we can acquire more surface details of sea water. In the performance of rendering speed, we are able to achieve rendering at interactive rates (>60 FPS) while keeping visually plausible appearance of water, which makes it possible to use the simulation as part of a computer 3D game.

The present simulation is a height field and therefore cannot handle breaking waves. This may be improved in various ways such as 3D Eulerian simulation on top of the height field, which is the further research which we are working on.

REFERENCES

[1] S. Premžoe, T. Tasdizen, J. Bigler, A. Lefohn and R. T. Whitaker, Particle-based simulation of fluids, *Computer Graphics Forum*, no.10, pp.401-410, 2003.

[2] M. Müller, D. Charypar and M. Gross, Particle-based fluid simulation for interactive applications, *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, pp.154-159, 2003.

[3] S. Clavet, P. Beaudoin and P. Poulin, Particle-based viscoelastic fluid simulation, *Proc. of the ACM Siggraph/Eurographics Symposium on Computer Animation*, pp.219-228, 2005.

[4] S. Shi, X. Ye, Z. Dong and Y. Zhang, Real-time simulation of large-scale dynamic river water, *Simulation Modeling Practice & Theory*, vol.15, no.6, pp.635-646, 2007.

[5] E. Algan, M. Kabak, B. Ozguc and T. Capin, Simulation of water drops on a surface, *3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video*, pp.361-364, 2008.

[6] J. Shi, D. Zhu, Y. Zhang and Z. Wang, Realistically rendering polluted water, *Visual Computer*, vol.28, nos.6-8, pp.647-656, 2012.

[7] S. Liu and Y. Xiong, Fast and stable simulation of virtual water scenes with interactions, *Virtual Reality*, vol.17, no.1, pp.77-88, 2013.

[8] S. Oh, Single-phase trapped air simulation in water flow, *Proc. of WSCG*, 2014.

[9] J. G. Lim, B. J. Kim and J. M. Hong, Water simulation using a responsive surface tracking for flow-type changes, *Visual Computer*, pp.1-11, 2015.

[10] M. Chlădek and R. Ďurikovič, Particle-based shallow water simulation for irregular and sparse simulation domains, *Computers & Graphics*, 2015.