ICIC Express Letters
Part B: Applications
Volume **8**, Number **4**, April **2017**

ICIC International ©2017 ISSN 2185-2766
pp. **671–679**

REALISTIC RENDERING OF OCEAN WATER VIA FLUID SIMULATION AND VOLUME RENDERING

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

[1]State Key-Laboratory of Mathematical Engineering and Advanced Computing
No. 62, Science Avenue, Zhengzhou 450000, P. R. China
fjhzzuli@qq.com

[2]School of Computer and Communication Engineering
[3]School of Software
Zhengzhou University of Light Industry
No. 5, Dongfeng Road, Zhengzhou 450002, P. R. China
*Corresponding author: jiexuzz@tom.com

Received August 2016; accepted November 2016

Abstract. *Realistic approximation of ocean water has become a common tool in visual effects work at all levels of computer graphics, from print media to 3D video games. In order to make it possible to render this in real time with a realistic quality similar to real surface of ocean, in this paper, we present the realistic approximation of ocean water via fluid simulation and volume rendering. We deduce the fluid simulation for ocean water by Navier-Stokes equations, and improve the performance by mathematical discretization: advection step, Poisson solver step and projection step. Our method uses ray marching and distance sampling to evaluate the line integral appearing in the transmittance within ocean water and acquires realistic rendering of water surface. The results prove that our method is more effective in both quality and speed of rendering.*
Keywords: Realistic approximation, Ocean water, Fluid simulation, Volume rendering

1. **Introduction.** Realistic simulation of liquids such as rising smoke, clouds and water is a very demanding task in the field of computer graphics, but it has historically been the domain of high-quality offline rendering due to great computational cost. Nowadays, in virtual reality and 3D games what matters most is that the simulations look both convincing and fast. Figure 1 gives the realistic rendering of sea water in one computer 3D games, and shows lighting effects including Fresnel reflection and refraction with today's GPU (Graphics Processing Unit).

In early years of fluid simulation, procedural surface generation was used to represent waves and smoke as presented by authors in [1, 2, 3]. There has been a significant progress in field of fluid simulation over the past few years. The shallow water equation [4], which is derived from Navier-Stokes equation [5], is used to the visual simulation of water. The simulated cases involve dry-bed zones and nontrivial bottom topographies, which are very useful to water simulation. With the introduction of programmable GPUs, it makes it feasible to exploit the computational power of the GPU for nongraphical purposes. Realistic ocean waves [6] are rendered on the GPU and can be applied to generating real-time water in visual reality. However, this technology storages 250 pictures of the height map of ocean surface, and the rendering speed cannot be improved further when rendering large scale ocean scene. Other methods for employing GPUs to simulate liquids [7, 8] have been proposed to realize real-time rendering, which acquire realistic rendering effects and have a great practical value in virtual reality. However, these methods are complex to be implemented and have a less practical value in virtual reality. 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

FIGURE 1. Realistic sea water rendering in 3D game

surface separately, and then blending them together in proportions that depend on the flow speed. Also fast realistic rendering of ocean surface [10] has been acquired on the GPU, which shows different illumination effects on ocean surface. However, they have to improve the rendering speed to meet the increasing requirement of interactive applications such as 3D online games.

Nowadays, with the increasing requirement of rendering speed, the main challenge of rendering is not only to approximate the realistic effect, but also develop sufficiently fast method to acquire interactive rendering at higher speed. Motivated by meeting the increasing requirement of interactive applications, our research focuses on faster liquid simulation like water while keeping visually plausible appearance. In this paper, we present the realistic approximation of ocean water via fluid simulation and volume rendering. By Navier-Stokes equations, we deduce the fluid simulation for ocean water, and improve the performance by mathematical discretization. Using volume rendering: ray marching and distance sampling, we acquire realistic rendering of water surface. From results, the achievements of this proposed method are that both realistic appearance and details of ocean water are obvious and clear. Also FPS (Frames Per Second) is higher when rendering large scale ocean water, which has a practical value in some interactive applications such as 3D video games.

The rest of this paper is organized as follows. Fluid simulation by Navier-Stokes equations and mathematical discretization is deduced in Section 2. Volume rendering is given in Section 3. The results are discussed in Section 4, and conclusions are drawn in Section 5.

2. Fluid Simulation by Navier-Stokes Equations and Mathematical Discretization.

Many fluids are simulated with a set of differential equations known as the Navier-Stokes equations [5]. These equations describe the fluid as a function of pressure, velocity, and time, and are shown below.

$$\rho\left(\frac{\delta v}{\delta t} + v \cdot \nabla v\right) = -\nabla p + \mu \nabla^2 v + f \tag{1}$$

where ρ is the density. v is the velocity. p is pressure. μ is the viscosity, and f is the sum of external forces.

The above Equation (1) needs some additional constraints to limit the degrees of freedom, which vary on the type of the simulation. The divergence of a velocity field v can be represented in vector notation as $\nabla \cdot v$. Adding the divergence-free restriction to the system of equations allows us to simplify Equation (1). The simplified equation is known as the incompressible Navier-Stokes equations as shown below.

$$\frac{\delta v}{\delta t} = -v \cdot \nabla v - \nabla p + f \tag{2}$$

$$\nabla \cdot v = 0 \tag{3}$$

The summary of the motion of a fluid can be described by the terms of the above equation. The first term is known as the advection term, and describes the propagation of the velocity field through time. The second term describes how the pressure acts as a counterbalance to the flow of the fluid to keep the fluid incompressible. It defines a projection of the velocity field onto the space of divergence-free velocity fields. To understand how this projection method works, we must use the mathematical property known as the Helmholtz-Hodge decomposition [11] (as shown in Figure 2).

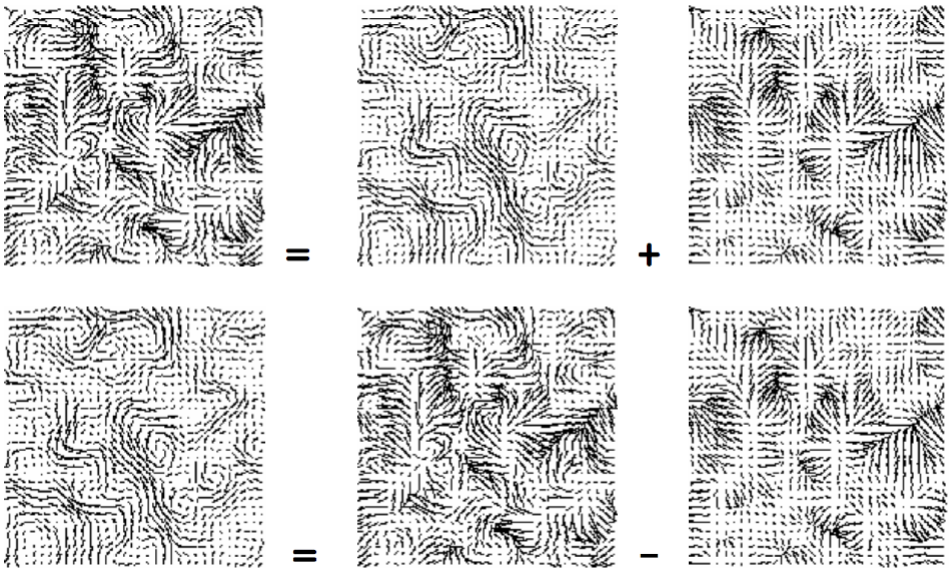

FIGURE 2. Helmholtz-Hodge decomposition

From Figure 2, it states that any vector field can be broken into the sum of a divergence-free vector field and a curl-free, or gradient, vector field. In the incompressible fluids case, the velocity field v can be represented as

$$v = v^* + \nabla p \tag{4}$$

where v^* is the divergence-free velocity field and ∇p is the curl-free, or gradient, vector field. Therefore, given the velocity v after applying the other terms of Equation (2), we can solve the final velocity field v^*. Taking the divergence of both sides of Equation (4), we get

$$\nabla \cdot v = \nabla^2 p \tag{5}$$

Relaxation solvers are a class of methods that iteratively solve systems of equations. One widely applicable use of them is that it can be used to solve the Poisson equation $\nabla p^2 = f$. Poisson relaxation solvers work by solving the equation for each cell using values from the neighboring cells. Taking the earlier Laplacian Equation (6) and solving p produces Equation (7):

$$\sum_d \frac{p_d - p}{h} = h\nabla \cdot v \tag{6}$$

$$p = \frac{\sum_d p_d - h^2(\nabla \cdot v)}{4} \tag{7}$$

Using Equation (7), we can solve p by applying this equation at each cell until convergence.

In this paper we use Poisson relaxation solvers by solving the equation for each cell using values from the neighboring cells as shown in Figure 3.

The Poisson relaxation method is an iterative method, requiring many grid traversals to converge. Since advection and projection require a single traversal, the pressure solver

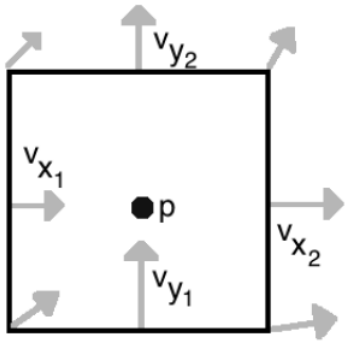

FIGURE 3. Computation of each cell using values from the neighboring cells

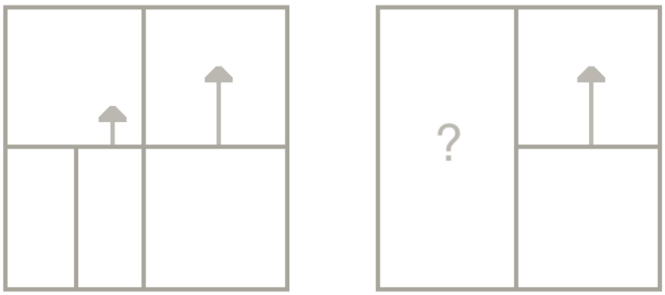

FIGURE 4. Problematic nodal velocity cases in a non-uniform grid

is definitely the bottleneck of these computations. As such, the majority of our research in this paper has focused on improving the performance of fluids simulation by mathematical discretization: advection step, Poisson solver step and projection step.

2.1. **Advection step.** To perform the Semi-Lagrangian advection, we use the same nodal velocity algorithm for the interpolation and other facial velocity component. However, this method becomes more complicated when the cell sizes are not uniform, and even worse when the cells themselves are not uniform. The simple case is when the two adjacent facial velocities are on faces of differing size. This means there are multiple valid adjacent face velocities to choose from on the other side. To reduce error, the smallest, and thus the closest, face is used (Figure 4, left), and a simple weighted average scheme suffices in this case. The much more complicated case is when a node has only one adjacent facial velocity for a given component (Figure 4, right).

Traditionally, it solves this problem by restricting the face velocity at T-junctions to be the average of the face's nodal velocities. However, this introduces error into the simulation and restricts the utility of the adaptation. In our approach, we interpolate the value of the face at the larger cell and average it with the existing value at the smaller cell. This decreases error because it does not ignore the facial velocity adjacent to the node. However, doing so introduces dependencies into the nodal velocity computation, because now nodal velocities are being used to compute nodal velocities. In quadtrees, this is not a problem because the nodal velocities at the T-junction can only depend on nodal velocities at a higher level in the tree, so there is a valid topological ordering of nodal velocities. For K-d trees, this is not the case, resulting in recursive dependencies in the nodal velocity calculation if interpolation is used.

To alleviate this, we use the nodal velocities on the adjacent face and the two facial velocity values (as shown in Figure 5). This does not have any recursive dependencies since these nodal velocities are on the parent of the smaller node, and the adjacent node, and thus cannot be at a T-junction.

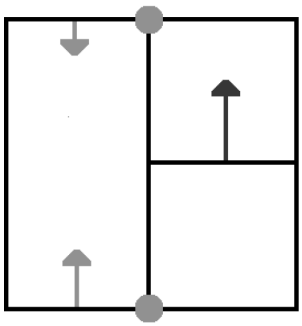

FIGURE 5. Mixed interpolation scheme in K-d trees

2.2. **Poisson solver step.** We use the relaxation solver to solve a pressure correction. Since the pressure p^* from the previous time step is already known, we can compute a pressure correction dp in order to compute the new pressure p.

$$p = p^* + dp \tag{8}$$

Plugging this into $\nabla \cdot v = \nabla^2 p$ we get

$$L(p) = L(p^*) + L(dp) = \nabla \cdot v \tag{9}$$

$$L(dp) = \nabla \cdot v - L(p^*) = R \tag{10}$$

Given the residual to our system of equations R, we repeatedly relax the pressure correction until convergence, using the following equation.

$$dp = \frac{R - A\beta_t}{\alpha_t} \tag{11}$$

where A is the area of the cell. β_t and α_t are the sum of β and α values over all of the cell's *face gradients*. Each gradient is defined to be

$$\nabla dp = \beta - \alpha p \tag{12}$$

where p is the pressure at the center of the cell. In the uniform case β is $\frac{pd}{h}$, and α is $\frac{1}{h}$. pd is the pressure value at the neighboring cell in direction d, and h is the width of the cell.

To compute the velocity divergence, the divergence theorem is used on the definition of velocity divergence to obtain

$$\int_c \nabla \cdot v = \oint_{\partial c} v \cdot n \tag{13}$$

where n is the face normal, c is the cell, and ∂c is the cell boundary.

Thus, since we are storing the normal components of the velocity at the faces, the velocity divergence is just the sum of the velocity components weighted by the length of the face (as shown in Figure 3). In the uniform case, this equation is simply

$$\nabla \cdot v = \frac{vx_2 + vy_2 - vx_1 - vy_1}{h} \tag{14}$$

2.3. **Projection step.** To compute the *face gradient*, we use the assertion that perturbing a value by $O(\triangle x)$ from the center of the cell results in a convergent approximation. We compute the *face gradient* using a weighted average of the approximate gradients of each neighbor n in direction d, according to the area of their intersecting face (as shown in Figure 6).

$$\nabla dp = \frac{1}{h} \sum w_n \frac{p_n - p}{h_n} \tag{15}$$

where h is the width of the cell. w is the weighted average of the residual in each cell.

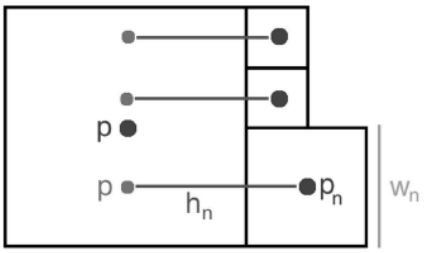

FIGURE 6. Face gradient discretization

Then, we use the gradient and the velocity at each face to compute the projected velocity at each face. This means that once this approximate pressure gradient is constructed, it is subtracted from the velocity at each face to obtain the projected velocity.

3. Volume Rendering.

3.1. **Model construction.** The density distribution ρ is used for constructing the fluid model in Section 2. The fluid is characterized by the distribution of its extinction coefficient σ_t and scattering coefficient σ_s. The relationship between ρ and σ_t is given so that ρ serves as a scaling factor of σ_t:

$$\sigma_t = \rho/800 \tag{16}$$

Given σ_t, we give σ_s as

$$\sigma_s = \sigma_t/2 \tag{17}$$

By definition, the relationship $\sigma_t > \sigma_s$ must hold. This relationship states that the albedo of the fluid medium is 0.5 for all wavelengths, giving the medium a physical color when illuminated with sun light.

3.2. **Distance sampling.** For evaluating the transmittance of a given line segment within the fluid medias such as clouds and water, our method uses ray marching to evaluate the line integral in the equation of the transmittance.

To take account of scattering, distance sampling is required to determine which point the ray is scattered inside the media. Distance sampling is designed as shown in Algorithm 1.

Algorithm 1 Distance Sampling Algorithm

```
Intersect medium bounding box and ray;
segment ← intersection line segment of medium bounding box and ray;
prob ← random([0, 1]);
S ← 0;
for p ← points along segment do
    S ← S + σ_t(p)ds
    if prob < exp(−S) then
        return p as the scattering points;
    endif
endfor
Ray has passed through medium;
```

The main idea of this sampling algorithm is to focus on the ratio of radiance that arrives at a given point, and regard that ratio as the probability of the light being able to pass that point. To sample distance, we first sample a random number $prob \in [0, 1]$ from a uniform distribution. Then, we calculate the transmittance T along the ray segment that intersects with the medium. Our distance sampling method then finds the point where

$T = prob$. If $T < prob$ at all points, we regard that the ray has passed through the medium.

4. **Experiment and Results.** We have implemented realistic approximation of ocean water via fluid simulation and volume rendering. 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 2015 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 fluid simulation such as effects of water scene and compare the rendering speed of our method with the previous method. Our final rendering clearly shows our method is more effective in both quality and speed of rendering, which is suited to interactive applications such as 3D games.

Firstly, we have to keep in mind that the look of the water depends mostly on how it interacts with the environment. We have implemented two scenes under different interactions. One scene represents the surface of the fluid in a small box which can show dynamic simulation. The other scene is more realistic surface of ocean considering light reflection and refraction.

To visualize water, a box scene containing one sphere is used, which can represent the collision process between water and sphere. This is very useful for the debugging and development process. The realistic appearance from water simulation via our method is shown in Figure 7.

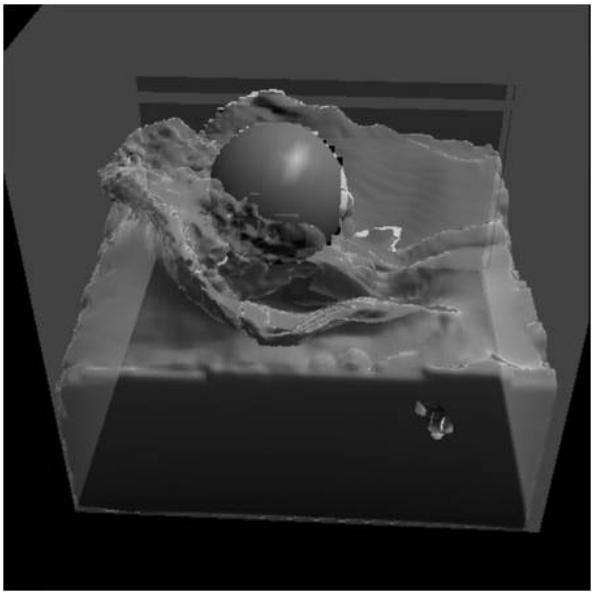

FIGURE 7. Dynamic simulation of water in a small box

Then, in order to testify effectiveness of our method, we render a large scale ocean water. The rendering of ocean water is done by ray marching and distance sampling. We first map the screen to the 3-D space, and get rays that start from the eye position and direct at each pixels in the resulting image. Then the ray is pushed forward in the level-set field and returns the thickness of the water as well as the reflection and refraction parameters. Combining fluid simulation, the whole scene of ocean water can be shown in Figure 9, which is compared to Figure 8 given by Zhang et al.'s method [6].

From Figure 8 and Figure 9, we can observe that the rendering appearance of water using our method is more realistic in such as water color, light reflection, caustics and soft shadow, which is in accord with physical phenomena. The effects in details such as waves and foams are very obvious and clear on the water surface, which shows our technique is more effective.

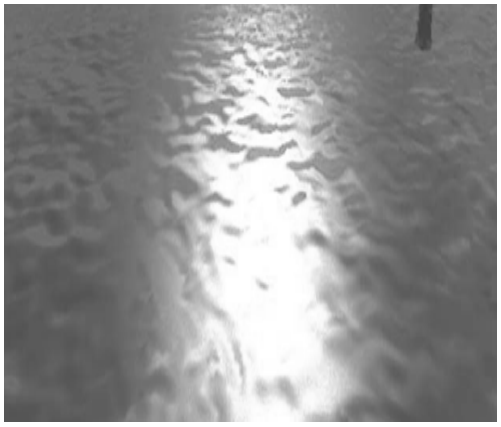

FIGURE 8. Ocean water rendering using reference method [6]

FIGURE 9. Ocean water rendering using our method

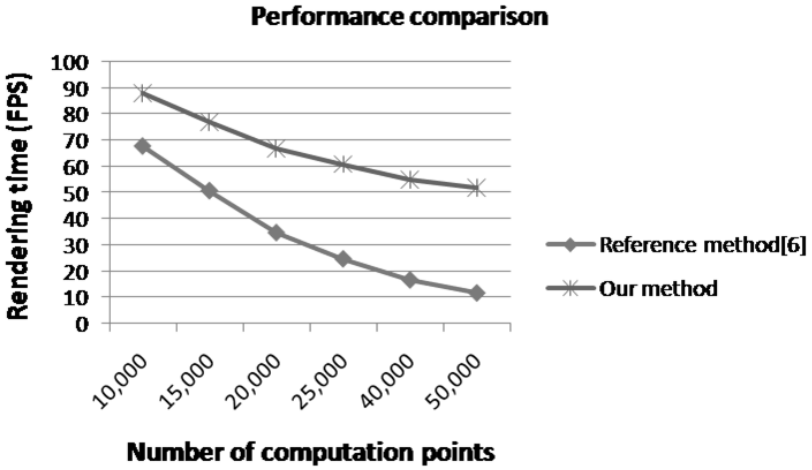

FIGURE 10. Performance comparison between our method and the reference method [6]

Performance comparison between Zhang et al.'s method [6] and our method is also shown in Figure 10. We have obtained about 88 FPS when rendering 1000 points; however, Zhang et al.'s method only acquires 67 FPS. From Figure 10, it is clear that our method performs much better than the reference method, even with pressure interpolation. The reason is that we improve the performance of the Poisson solver and implement a more efficient sampling algorithm. When rendering the larger scene of fluid flows, the difference of rendering speed is more obvious as shown in Figure 10.

5. **Conclusions.** We have implemented the fast approximation of ocean water via fluid simulation and volume rendering. The rendering results show that we can acquire realistic appearance of water such as water color, light reflection, caustics and soft shadow. The effects in details such as waves and foams are got as well. Our method performs much better than reference method [6] in rendering speed. Above all, our rendering results are in accord with physical phenomena and our technique is more effective.

Later we plan on including more real time factors in our future project and present a more real-time rendering of water waves. We would also like to show the behavior of waves when it comes in contact with other rigid objects like rocks.

Acknowledgment. This work was supported by the National Natural Science Foundation of China under Grant No. 61672470.

REFERENCES

[1] C. T. Pozzer, Procedural solid-space techniques for modeling and animating waves, *Computers & Graphics*, vol.26, no.6, pp.877-885, 2002.

[2] D. Hinsinger, F. Neyret and M. P. Cani, Interactive animation of ocean waves, *Symposium on Computer Animation*, pp.161-166, 2002.

[3] A. Selle, A. Mohr and S. Chenney, Cartoon rendering of smoke animations, *Proc. of the 3rd International Symposium on Non-photorealistic Animation and Rendering*, pp.57-60, 2004.

[4] T. R. Hagen, J. M. Hjelmervik, K. A. Lie, J. R. Natvig and M. O. Henriksen, Visual simulation of shallow-water waves, *Simulation Modeling Practice & Theory*, vol.13, no.8, pp.716-726, 2005.

[5] G. Chen, C. Kharif, S. Zaleski and J. Li, Two-dimensional Navier-Stokes simulation of breaking waves, *Physics of Fluids*, vol.11, no.1, pp.121-133, 1999.

[6] W. Zhang, H. Zhou, L. Tang and X. Zhou, Realistic real-time rendering for ocean waves on GPU, *Proc. of IEEE International Conference on Progress in Informatics and Computing*, pp.743-747, 2010.

[7] X. Cai, B. Qian, H. Sun and J. Li, Rendering realistic ocean scenes on GPU, *Trans. Edutainment VII*, pp.230-238, 2012.

[8] T. Imai, Y. Kanamori, Y. Fukui and J. Mitani, Real-time screen-space liquid rendering with two-sided refractions, *ITE Technical Report*, 2014.

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

[10] M. Yang and H. Yuan, GPU-based fast realistic rendering of ocean surface, *ICIC Express Letters, Part B: Applications*, vol.6, no.10, pp.2851-2856, 2015.

[11] J. Stam, Real-time fluid dynamics for games, *Proc. of the Game Developer Conference*, vol.18, 2003.