ICIC Express Letters
Part B: Applications
Volume **8**, Number **3**, March **2017**

ICIC International ©2017 ISSN 2185-2766
pp. 449–457

SIMULATION OF SEA WATER BASED ON SOLUTION OF THE NAVIER-STOKES EQUATIONS ON THE GPU

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
fujinhuazz@qq.com

[2]Engineering Training Center
[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. *Simulation of sea water can substantially increase the fidelity of natural environment simulation, and its implementation in real-time graphics has been a great challenge in the interactive applications. In this paper, we present simulation of sea water based on solution of the Navier-Stokes equations on the GPU (Graphics Processing Unit). The algorithm for running water simulation is based on the Navier-Stokes equations for incompressible flow. We solve the velocity of a flow, the pressure in the fluid and the influence of external forces, which is implemented on the GPU. Considering the optical effects such as reflection and refraction, we also simulate interaction between light and sea water on the surface and underwater scene. The results show that we can acquire realistic simulation, while keeping fast speed of rendering at interactive rates.*
Keywords: Sea water, Navier-Stokes equations, Graphics Processing Unit, Optical effects

1. **Introduction.** The process of rendering sea water in real time is highly dependent on how much realism it needed in an application which can be a computer game, a movie or just some scientific setup being used to run some simulations. In the early days of computer graphics, water is used to be treated as a planar surface basically. However, with the drastic improvement in the rest of the computer graphics, the need for better animation of water scene grows strongly. This is one image of screenshots from simulation of sea water surface running on a Radeon 9700 as shown in Figure 1.

With the development of the computer technology, various numerical methods are applied in approximating sea water. Fluid simulation in computer graphics begins with lower dimensional techniques such as the particle system [1, 2, 3]. These lower quality techniques are used to create 2D shallow water models, and semi-random turbulent noise fields. Then, by improving particle-based system, Clavet et al. [4] present a new method for viscoelastic fluid simulation. They achieve realistic small-scale behavior of substances and also extend the technique to handle interaction between fluid and dynamic objects. It is important to understand how light behaves when it interacts with sea water in reality, because that is what mainly decides how it is going to view it. So using physical interaction between light and sea water, some rendering techniques [5, 6, 7] have also been introduced to simulate sea water. The simulation of fluids in real-time graphics can have a great impact on the visual appearance, but to simulate this on a GPU using a traditional rasterization-pipeline is not straightforward. Nowadays, with the increasing development of GPU, rendering of sea is not only to simulate realistic appearance, but also to develop sufficiently simple method to allow for feasible computation on the GPU. Some real-time

FIGURE 1. One image about simulation of sea water surface

rendering methods of sea water [8, 9, 10] have been acquired, and realistic ocean rendering results are got as well. However, they have to improve the rendering speed to meet high requirement of the interactive applications later, and some details on the surface could also be simulated further considering the interaction between water and illumination.

Our focus differs in that we design GPU parallelization of the Navier-Stokes to meet increasing requirement of interactive applications. Also we improve the realism of undersea water which people nearly seldom research considering refraction of optical effects. Based on making extensive use of GPU rendering pipeline, in this paper, we present a novel method for rapid rendering of sea including undersea water at interactive rates. We exploit the incremental numerical approach to simplify the Navier-Stokes equation [11] of classic fluid simulation, and use Jacobi iterations to solve the Poisson equations on the GPU. Also we improve realism of sea water via approximation of interaction between light and water. The results show that we can acquire realistic rendering of sea scene such as water color, reflection, caustics and details on surface, while achieving a relatively faster speed of rendering.

The rest of this paper is organized as follows. The efficient fluid simulation by solving the Navier-stokes equations is deduced in Section 2, which includes introduction and solution of the Navier-Stokes equations, solution of the Poisson equations on the GPU, and initial and boundary conditions. Implementation of water body simulation is designed in Section 3. Realism improvement of sea scene via optical effects is given in Section 4. The results are discussed in Section 5, and conclusions are drawn in Section 6.

2. **Efficient Fluid Simulation by Solving the Navier-Stokes Equations.**

The algorithm for running fluid simulation is based on the Navier-Stokes equations for incompressible flow. The equations describes how the velocity of a flow varies over time incident to the diffusion of the flow, the pressure in the fluid and the influence of external forces. A description of how to solve these equations and how to implement it on the GPU is presented in this section.

2.1. **Introduction of the Navier-Stokes equations.**

The fluid flow simulation is governed by the famous incompressible Navier-Stokes equations, a set of partial different equations that are supposed to hold throughout the fluid. They are usually written as:

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

$$u \cdot \nabla = 0 \tag{2}$$

where t is a time step. u is the velocity. v is the viscosity. ρ is the density. p is the pressure. f is the body force such as gravity which can be set an invariant constant in the program.

Equation (1) consists of four terms, and each term describes a factor that influences the flow of the fluid. These factors are called **advection**, **pressure**, **diffusion** and **external forces**. All of them are accelerations and they represent different physical phenomena. Advection describes how a flow transports other objects and itself. This term represents how the flow transports itself. Pressure builds up when molecules collide. This pressure then pushes other molecules away and the movement propagates though the fluid. This is represented by the pressure term. Fluids can have a resistance to movement. This resistance is described by the viscosity of a fluid. High viscosity leads to slower movement, e.g., honey, and low viscosity leads to more movement, e.g., smoke. Viscosity leads to diffusion of the flow and is represented by this term in the equation. External forces represent external forces such as gravity, pushing objects and wind, as follows. We can give a constant value to represent it.

2.2. **Solution of the Navier-Stokes equations.** In this paper, to solve the different terms of the Navier-Stokes equations we need to use some vector mathematics. The following formulas describe the different uses of the nabla operator ∇ that are used to solve the different terms between point x and point y.

Gradient:

$$\nabla p = \left(\frac{\delta p}{\delta x}, \frac{\delta p}{\delta y}\right) \Rightarrow \frac{p_{i+1,j} - p_{i-1,j}}{2\delta x}, \quad \frac{p_{i,j+1} - p_{i,j-1}}{2\delta y} \tag{3}$$

Divergence:

$$\nabla \cdot u = \left(\frac{\delta u}{\delta x}, \frac{\delta u}{\delta y}\right) \Rightarrow \frac{u_{i+1,j} - u_{i-1,j}}{2\delta x}, \quad \frac{u_{i,j+1} - u_{i,j-1}}{2\delta y} \tag{4}$$

Laplacian:

$$\nabla^2 p = \left(\frac{\delta^2 p}{\delta x^2}, \frac{\delta^2 p}{\delta y^2}\right) \Rightarrow \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2}, \quad \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2} \tag{5}$$

What worths noting is that the right side of the arrow in Equation (3), Equation (4) and Equation (5) is the finite difference form of the left side. It is only possible to solve the Navier-Stokes equations analytically in a few simple cases so we use an incremental numerical approach instead.

To compute the new velocity field at a time step we can simply take the previous velocity field, apply advection, calculate diffusion and add influence of external forces. However, this result will violate the second part of the Navier-Stokes equations, and the divergence will no longer be zero. This can be solved by using the Helmholtz-Hodge decomposition theorem [12]:

$$w = u + \nabla p \tag{6}$$

where w is the non-divergence free velocity field, u is the divergence free velocity field and p is the pressure field. From this theorem we get:

$$u = w - \nabla p \tag{7}$$

Basically we can get the divergence free field by subtracting the gradient of the pressure field. Now we just need a way to compute the pressure field which is given by applying the divergence to the same theorem as shown in Equation (6):

$$\nabla \cdot w = \nabla \cdot u + \nabla \cdot \nabla p = \nabla \cdot u + \nabla^2 p \tag{8}$$

Since Equation (2) demands that divergence of u is zero, this leads to:

$$\nabla^2 p \cdot w = \nabla \cdot w \tag{9}$$

This is an Poisson equation which can be solved with Jacobi iterations using w as input, which will be presented in Section 2.3. Since we now know how we can get pressure, we just need an exact way of calculating w. From the definition of the dot product we know that we can compute the projection of a vector on a unit vector by applying dot product between the two vectors. This operation can also be applied to vector field. We can take advantage of this and define a projection operation $\mathbb{P}$ that takes the field w and projects it onto the divergence free field u. Now we can apply this projection operation to Equation (6) and since $\mathbb{P}(w) = \mathbb{P}(u) = u$, we get:

$$\mathbb{P}(\nabla p) = 0 \tag{10}$$

Now we can apply the same projection operation to the first Navier-Stokes equation as shown in Equation (1):

$$\mathbb{P}\left(\frac{\delta u}{\delta t}\right) = \mathbb{P}\left[-(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + v\nabla^2 u + f\right] \tag{11}$$

Since u is divergence free and the projection disappears from the left hand side of the equation, combining this with what we know from Equation (10) we arrive at the following equation which is the final equation that our implementation will solve.

$$\frac{\delta u}{\delta t} = \mathbb{P}\left[-(u \cdot \nabla)u - \frac{1}{\rho} + v\nabla^2 u + f\right] \tag{12}$$

This is an equation we can compute at each time step. We have access to the velocity field u and can simply apply the three inner operations, add them together to produce w and apply the projection operation to arriving at the final result which is the new velocity field.

Typically, the different components are not computed as in Equation (12) but instead calculated using state transformations. Each component takes a vector field as input, performs calculations and produces a new vector field that is used as input to the next component. We therefore define $\mathbb{S}$ as the solution to Equation (12) for a single time step. $\mathbb{S}$ can be split into operations (applied right to left):

$$\mathbb{S} = \mathbb{P} \circ \mathbb{F} \circ \mathbb{D} \circ \mathbb{A}(u) \tag{13}$$

where $\mathbb{A}$ is advection, $\mathbb{D}$ is diffusion, $\mathbb{F}$ is external forces, $\mathbb{P}$ is projection.

To compute the advection at the current time we need to access the velocity at the previous time step. Since we intend to implement this in the GPU fragment shaders and we cannot change the position of the current pixel, instead we back trace the velocity for each grid cell and copy the contents from this position into the current pixel. This can be applied on velocity, density, temperature, etcetera.

To compute the diffusion, we use the following diffusion equation

$$\left(I - v\delta t\nabla^2\right) u(x, t + \delta t) = u(x, t) \tag{14}$$

where I is the identity matrix. This is a Poisson equation that we solve using Jacobi iterations.

2.3. **Solution of the Poisson equations on the GPU.** To solve the diffusion equation (14) as well as the Poisson pressure equation we use several Jacobi iterations. Both equations can be expressed on the form:

$$x_{i,j}^{(k+1)} = \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} + \alpha b_{i,j}}{\beta} \tag{15}$$

where α and β are two coefficients when Jacobi iterations are performed. b is the convergence rate.

This equation is evaluated once per iteration and the result is given as input for the next iteration. The initial input has to be guessed and in our implementation the initial

guess is 0. After an amount of iterations 20-40 the result starts to converge towards the desired value. There are other solutions than Jacobi that require less iterations, but they are far more complicated to be implemented on the GPU.

2.4. **Initial and boundary conditions.** In order to have a fully functioning model, we need to know what the initial state is for our velocity and pressure. We set the initial velocity and pressure to zero throughout the grid. We also need to know what happens at the boundaries and specify our boundaries and boundary conditions. We imagine our simulation takes place in a big box that is positioned so that its entire contents are visible on the screen. To get the effect of our water being contained in this box, we want the fluid to "bounce" in contact with the hills in the water. Therefore, our velocity boundary condition is to take the negative value of the nearest cell. For pressure we set the boundary to have the same pressure as the nearest cell to get a realistic simulation.

3. **Implementation of Water Body Simulation.** Simulating fluid physics is a very demanding task for a processor to run, and there are many computations that have to run for every element in the fluid. However this is also a very parallel task, since every single element has to run the same operation every loop cycle, which fits the GPU very well. Our implementation utilizes the computational power of the GPU by running shader programs for every pixel and storing intermediate results into textures.

From solution of the Navier-Stokes equations we get Equation (13) which we can implement. Every time step is the same, and this pseudocode represents a single time step respectively:

- u1 = advection(u1);
- u2 = diffusion(u2);
- u3 = addforces(u3);
- u4 = projection(u4);

where the projection operation consists of two operations as follows:

- p = computePressure(u);
- u = subtractPressureGradient(u, p).

This code could be implemented both on the CPU and the GPU; however, as stated above, the parallel nature of the GPU makes it a much better match. To implement this kind of algorithm on the GPU, we need to learn how to do this kind of computation on the GPU. In our system the algorithm looks something like this.

RCInit():

- Create quad that spans the entire grid.
- Setup textures to store all the different values at the grid cells (velocity, temperature, pressure, etc).
- Initiate the fragment shaders that will perform the necessary calculations.

RCUpdate():

- Render the quad into different buffers using the different shader programs that perform the operations (advection, diffusion, divergence, etc).
- Iterate Jacobi renders to compute the pressure.
- Subtract the pressure gradient to arrive at the final velocity field.

To run the simulation we need at least two buffers that we render to, velocity and color. Running only this simulation is however not very interesting. To get something interesting out of it we need to put something such as fishes and hills into the sea water that is carried through the flow and can be visualized. Since we focus on making water simulation we chose to visualize this by adding a buffer for the density.

4. **Realism Improvement of Sea Scene via Optical Effects.** For realistic rendering of sea scene it is essential to handle the physical interaction between water and incoming light correctly. This surface and underwater realism can be achieved if reflection and

refraction effects are computed. In this paper, we have implemented a visualization that is based on the physical laws of reflection and refraction.

4.1. **Reflection effect.** Light is known to behave in a very predictable manner. If a ray of light could be observed approaching and reflecting off a flat mirror like water surface, then the behavior of the light as it reflects would follow a predictable law known as the law of reflection. The diagram below illustrates the law of reflection as shown in Figure 2.

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:

$$R = I - 2N(N \cdot I) \tag{16}$$

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.

4.2. **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:

$$\eta_2 \sin(\theta_1) = \eta_1 \sin(\theta_2) \tag{17}$$

where η_1 and η_2 are the refractive indices for two materials, θ_1 is the angle of incidence between the camera ray I and the surface normal N, and θ_2 is the angle of refraction between the surface normal below the surface $-N$ and the refracted vector T, also called *transmitted*, as depicted in Figure 3.

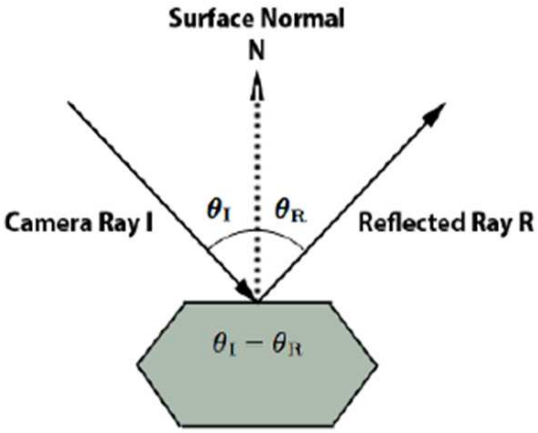


FIGURE 2. Reflection of light at media surface

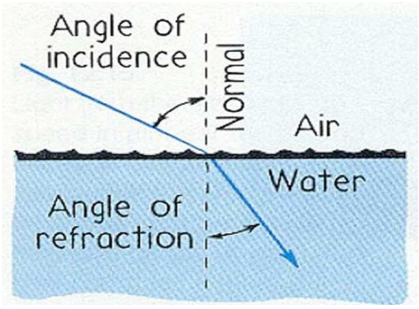


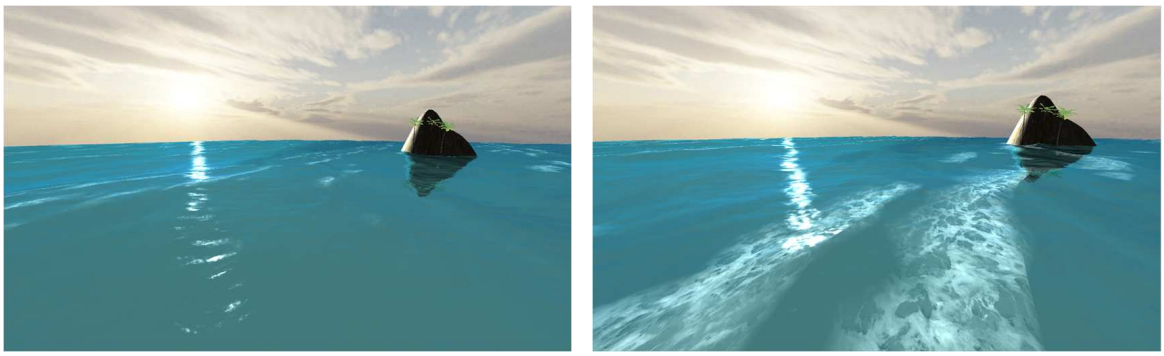
FIGURE 3. Refraction of light at the interface between two media

5. **Experiments and Results.** We have implemented the fast rendering for sea water using fluid simulation and optical effects 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 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 rendering effects of sea scene and compare the rendering speed of our method with the previous method. Our final rendering clearly shows the effective implementation of the above mentioned technique.

We firstly implement realistic rendering of sea surface under sunlight illumination, and show different scenes to express realistic rendering effects in different views. The images from sea simulation via our method are shown in Figure 4.

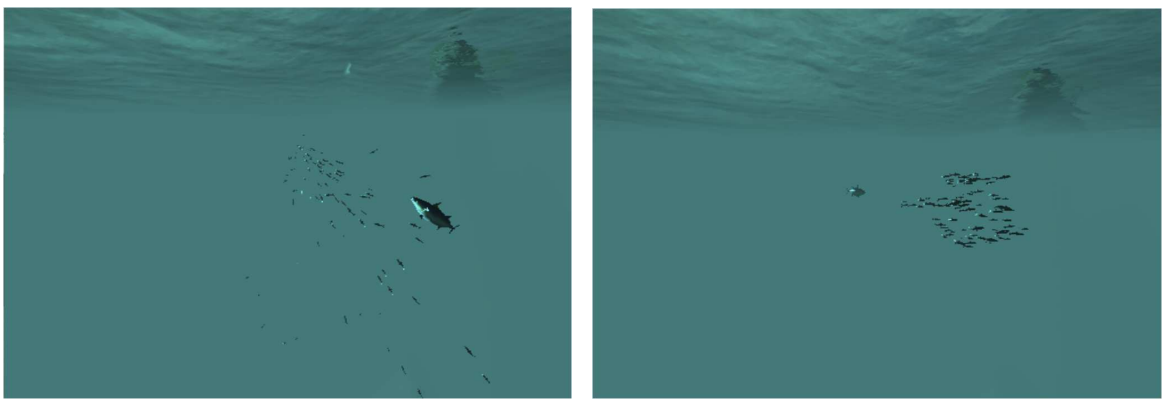
Then, we render two underwater scenes containing many swimming fishes and show different rendering effects in different views as shown in Figure 5.

From Figure 4, we can observe that the rendering appearance of sea surface using our method is realistic in such as water color, light reflection and caustics, which is in accord with physical phenomena. Although Liu and Xiong's method [9] can also show this physical phenomenon, the effects in details such as soft shadow and foams cannot be simulated very well, and our surface details in Figure 4(b) are very obvious and clear. Especially, we can simulate underwater scenes in Figure 5 but Liu and Xiong's method [9] can only approximate surface. Meanwhile, Liu and Xiong's method [9] cannot well deal with the break waves, and may generate implausible results for objects with non-symmetrical geometry. This is because pre-rigid body method in their rendering is actually a simplified



(a) (b)

FIGURE 4. Rendering results of sea surface in different views



(a) (b)

FIGURE 5. Rendering results of underwater scene in different views

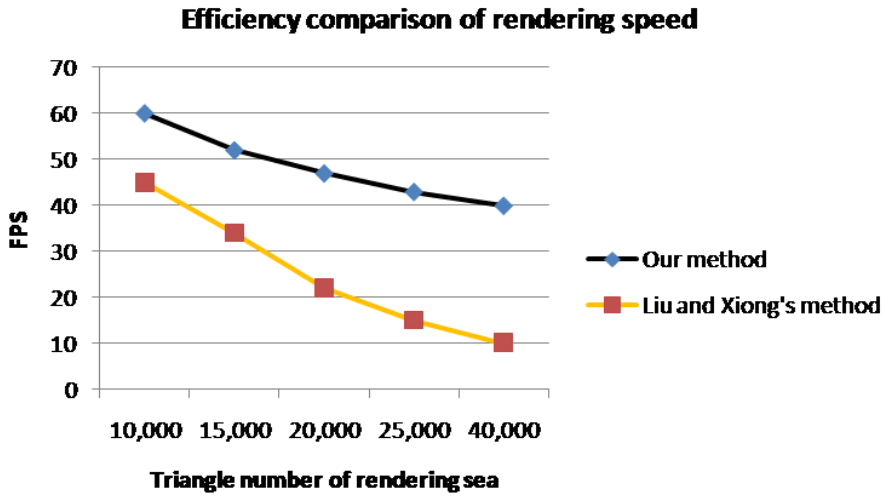


FIGURE 6. Efficiency comparison between our method and the literature method [9]

method for dynamic objects with approximately symmetrical geometry. Above all, rendering appearance of our technique is more effective over Liu and Xiong's method [9].

Figure 6 shows the comparison of rendering speed between Liu and Xiong's method [9], and our method, from which we can see that our method is more efficient. Our method is based on the discretization of Navier-Stokes equations which can accelerate the computation of sea water, and the rendering is implemented directly in graphics memory of GeForce GT430 via GLSL shader programming, so the speed is faster on the whole. When the triangle number of the simulation sea is ten thousand, we have obtained about 60 frames per second (FPS). However, the real-time rendering method such as Liu and Xiong's method [9] only acquires 45 FPS. The difference is more obvious when increasing triangle number of rendering. From Figure 6, our method is about 1.33-4 times faster than Liu and Xiong's method [9].

6. **Conclusions.** We have implemented simulation for sea water with fluid simulation and optical effects on the GPU. Through rendering results, we can observe that the rendering of sea scene using our method is realistic both in sea surface and underwater scene. We can acquire realistic effects such as water color, light reflection, caustics, foams and soft shadow, which is in accord with physical phenomena. By comparing the efficiency between Liu and Xiong's method [9], and our method, we can obtain a faster rendering speed and the difference is more obvious when increasing triangle number of rendering. Hence our method has a great practical value in interactive applications such as 3D online games.

We have demonstrated some of the useful properties of our method, but many exciting avenues remain to be explored. In future work, we will improve the following subjects. We want to simplify the proposed method in order to make it combined with practical applications easily. We also plan to investigate its use for the interactive control of sea water motion to improve practical value.

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

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] T. Tsunemi, F. Hiroko, K. Atsushi, H. Kazuhiro and S. Takahiro, Realistic animation of fluid with splash and foam, *Computer Graphics Forum*, vol.22, no.3, pp.391-400, 2003.
[4] 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.
[5] J. Shi, D. Zhu, Y. Zhang and Z. Wang, Realistically rendering polluted water, *The Visual Computer*, vol.28, no.6, pp.647-656, 2012.
[6] X. Cai, B. Qian, H. Sun and J. Li, Rendering realistic ocean scenes on GPU, *Lecture Notes in Computer Science*, vol.7145, pp.230-238, 2012.
[7] B. Ouyang, F. Dalgleish, A. Vuorenkoski, W. Britton, B. Ramos and B. Metzger, Visualization and image enhancement for multi-static underwater laser line scan system using image-based rendering, *IEEE Journal of Oceanic Engineering*, vol.38, no.3, pp.566-580, 2013.
[8] A. Semmo, J. E. Kyprianidis, M. Trapp and J. Llner, Real-time rendering of water surfaces with cartography-oriented design, *Proc. of International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging*, pp.5-14, 2013.
[9] 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.
[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] A. Quarteroni, Navier-Stokes equations, in *Numerical Models for Differential Problems*, Springer Milan, 2014.
[12] H. Bhatia, G. Norgard, V. Pascucci and P.-T. Bremer, The Helmholtz-Hodge decomposition: A survey, *IEEE Trans. Visualization & Computer Graphics*, vol.19, no.8, pp.1386-1404, 2013.