An Efficient Slice-Based Water Simulation Approach for Breaking Waves

Chung-Nan Lee, Jyun-Ming Wu, Yan-Zheng Huang, Da-Jing Zhang-Jian
Department of Computer Science and Engineering National Sun Yat-sen University

ABSTRACT
The simulation of breaking wave has a computationally intensive application. To reduce the computation this paper presents a slice-based water simulation method for ocean breaking waves on natural simulation by generating the 2D simulations and then integrating these 2D simulation results into a 3D shape. We first simulate a 2D wave by a 2D Navier-Stokes (NS) solver to obtain the varying of ocean. Then, we propose a new reconstruction free surface method via Volume of Fluid (VOF) to further reduce the computation time and achieve better efficiency by using a linear interpolation with noise function to construct a complete 3D ocean simulation from these 2D simulations.

Keywords: *Slice Method, Volume of Fluid, Breaking Wave*

1. INTRODUCTION
Physics-based natural simulation of computer animation becomes popular in recent years. The 3D graphics hardware has been widely applied to various fields with the rapid progress of multimedia system and virtual reality. In natural simulation, it was difficult to render the ocean on the PC, due to the restriction of computing power in the past. Fortunately, after the first graphics processing unit introduced by NVIDIA in 1999 that integrating transform, lighting, triangle setup/clipping, and rendering engines is capable of processing a minimum of 10 million polygons per second, as a result, the natural simulation and many other 3D graphic applications become feasible. Speed and realism are two mainly considerations in breaking wave simulation. For the game purpose the speed is mainly concerned. For the physical research, "realism" is very important due to users hope to get precise image or results.

For ocean simulation one can use Navier-Stokes equations (NSE) to describe the motion of incompressible fluid. NSE is viscous Newtonian fluid, and can calculate the fluid motion accurately and simulate the effect effectively. The breaking wave needs much more calculation to simulate the fluid situation.

In addition, most of existing studies about real-time water simulation cannot simulate breaking wave. The breaking wave simulation spends a lot of time because it needs complex physical calculation. In order to simulate the breaking waves in real time, this paper proposes a slice-based fluid simulation, and combines linear interpolation with noise function to construct a complete 3D fluid simulation. The goal of this paper is to get a fast breaking wave simulation without loss of too much realism.

2. Related Work
Because the focus of computer animation is perceived by the human eye that can be fooled the visual on clever in the movies or games even if it is incorrect. Therefore, they often focus on real-time simulation on current research. There are two types of fluid frameworks Eulerian and Lagrangian for the ocean simulation. In [2] a hybrid method, called particle level set method that combines Eulerian with Lagrangian framework is proposed.

It is very important to get the free surface on the ocean simulation. According to the definition of Kinetics, the stress of the tangent direction sum is equal to zero. Assuming that the particle system based on the Lagrangian framework has enough number of particles, the particle will form automatically free surface. It needs to obtain the free surface on Eulerian framework, with the complicated calculation of boundary. Currently based on the grid method, there are some methods such as MAC grid, level set, particle level set, VOF, front tracking, ghost fluid, and height field.

2.1 Height field
Ocean wave is a sine wave alike. Breaking waves are traditionally modeled using sinusoidal and trochoidal functions [16]. It gets height of the surface by the height field function. The height function method doesn't consider the eddy varying under the water, and the simulation is fast and has good result in quality. Height field has better effect when sea is calm, but the result is worse for windy weather. There is one problem for height field [8], when simulating the shallow water, due to it cannot generate the breaking wave. Tessendorf [6] used statistical methods by the oceanographer observation and fast Fourier Transform (FFT) to obtain the results of breaking wave. Jensen and Golias [7] introduced a new real-time level-of-detail deepwater animation scheme, which uses many different proven water models. Johanson [1] used Perlin nosie to simulate the details of ocean, and presented an alternative technique called "projected grid". For the shallow water waves on a height-field, the wave equation in [17] can be used to describe the fluid flow by the 2D NSE and wave effects. In addition, a noise-based animation of detailed fluid structures further improves the realistic appearance.

2.2 Breaking wave
When the weather is stormy or windy, the wave will overturn and break. The breaking wave is very important for reality. The breaking wave includes the spilling breaking, plunging breaking and surging breaking.

If we can simulate the different kinds of breaking waves, the simulation results will become more real. The height field can get the real result and fast simulation, but the disadvantage is that it cannot simulate breaking waves. Existing methods for breaking wave use NSE to calculate accurately the varying of sea to implement breaking waves. Unfortunately, the NSE is a set of highly non-linear partial differential equations (PDE).

There are many methods for discrete the PDE in the literature (e.g. finite difference [9], finite volume [18] and finite element [14]). Stam [5] proposed the semi-Lagrangian approach to solve the advection of NSE and simulate the varying of the 3D fluid. Foster and Fedkiw [10] presented a general method for modeling and animating liquids, and extended the object interaction mechanism to provide control over the liquid's 3D motion. Mihalef et al. [20] developed a novel fluid animation control approach to control breaking waves, and proposed the slice method to define the

shape of a breaking wave at a desired moment by the library evolution of breaking waves. Irving et al. [4] proposed an approach which optimizes the fluid simulations with tall and thin cells, and also reduces the computational complexity for large fluid volumes. Thürey et al. [12] presented a new method for enhancing shallow water simulations by the effect of overturning waves that make it possible to simulate scenes in real-time.

The wave simulation including the splashes and foam has an important role in the visual representation of the breaking waves. The particle system often specifies the splashes and foam movement in 3D environment. Takahashi et al. [19] proposed the control of the generation, vanishing and transition rule of splashes and foam. Based on the particle model Thürey et al. [11] presented a shallow water to achieve real-time by using a smoothed particle hydrodynamics simulation. Wang et al. [13] used moving particle semi-implicit (MPS) method, and using fractional Brownian motion (fBm) 2D simulation to expand into 3D representation by giving motion variation.

Due to simulating the effect of breaking waves needs a large amount of calculation and spends considerable time. Therefore, this paper proposes a slice-based method to speed up the simulation of breaking wave by using a new reconstructing algorithm keeps the simulation of volume fraction of fluid in VOF in 2D. Then it is expanded to 3D representation by combining linear interpolation with noise function. The proposed algorithm is effective in reducing the computation time.

3. THE PROPOSED METHOD

We proposed a slice-based method to get 2D simulation results first and then combined them using a linear interpolation with noise function to generate the new 2D simulation slices from previous two 2D simulations. Because the linear interpolation will cause unnatural result, the noise function is used to make the motion and behavior look more natural. For the 2D simulation we use NSE as the physical equation and solve free surface via VOF. In addition, about the reconstruction of volume fraction in VOF [15] one can improve by a moving interface reconstructing algorithm, and use the project grid method in reducing unnecessary computation in implementing the level of detail (LOD).

The system architecture for the proposed ocean simulation is given in Figure 1. It consists four parts including the grid generation, wave simulation, particle simulation, and render respectively. And the flowchart is given in Figure 2.

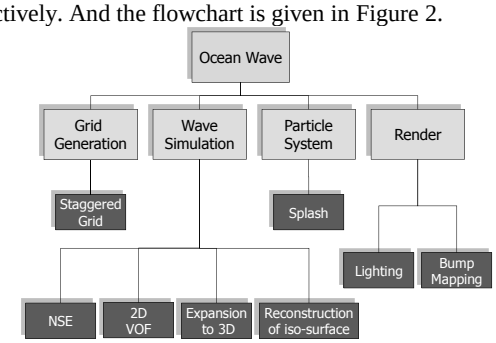

Figure 1：System architecture

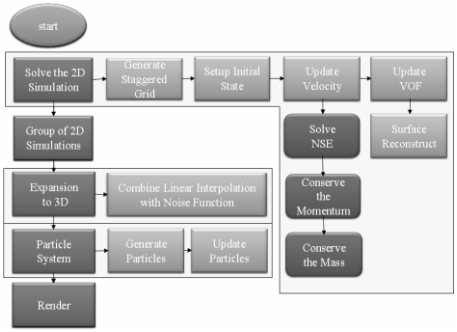

Figure 2：The flowchart of the proposed method

3.1 Grid generation

We use finite difference schemes that can be described as non-staggered or on staggered grids. In the same positions all the scalar variables for the non-staggered grid are stored, and the staggered grid the scalar variables (e.g. pressure and density) are stored in the cell centers of the control volumes, whereas the velocity or momentum variables are located at the cell faces. Using a staggered grid is a feasible way to avoid odd-even decoupling between the velocity and pressure. Odd-even decoupling is a discretization error that can occur on collocated grids and can lead to checkerboard patterns in the solutions.

3.2 Wave simulation

The main part we concern for the ocean simulation is the variation of water wave. We can use 2D NSE to calculate the fluid varying, and solve the problem of free surface using VOF. However, by combining the linear interpolation with noise function one can generate the new 2D simulation results between any two existing 2D simulations. Finally, we use an isosurface reconstruction approach to obtain the 3D surface.

3.2.1 Fluid equation

The incompressible NS equation is used for the ocean simulation.

The NSE includes three types of forces, they are body forces (F_g), pressure forces (F_p) and viscous forces (F_v). The body forces act on the entire water element. We assume only the gravity, $F_g = \rho G$, where ρ is density and G is the gravitational force (9.81m/s^2). The pressure forces act inwards and normal to the water surface, so the pressure forces are defined as the negative ($F_p = -\nabla \cdot p$). The viscous forces are $F_v = \mu \nabla^2 V$ according to the definition of Newtonian fluid, where $\mu = 1/\rho VL$, ρ is density, V is velocity and L is dimension. Newton's second law is used ($F = mA$) to describe the motion. According to the $F = F_g + F_p + F_v = \rho A$, we can get the expansion as $\rho A = \rho G - \nabla \cdot p + \mu \nabla^2 V$. Assume that density is uniform, the acceleration is $A = G - \nabla p/\rho + \mu \nabla^2 V/\rho$. The varying velocity over time is $\partial V/\partial t + (\nabla \cdot V)V = G - (\nabla \cdot p)/\rho + \left(\mu \nabla^2 V\right)/\rho$.

This equation conserves the momentum. In addition, the equation $\nabla \cdot V = 0$ describes a conservation of mass. The last two equations together are referred to as the NSE. The NSE contains four terms: force, advect, diffuse, and pressure. The steps of the NSE solver are illustrated in Figure 3. We can solve the equation one by one.

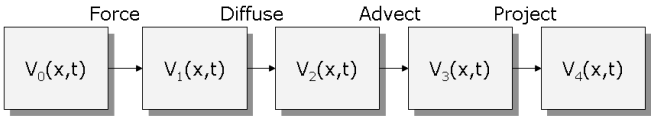

Figure 3：The steps of the NSE solver

First, the simplest term is the external force f. These forces may be either local forces (wind blow) or body forces (gravity). Here we assume that the force does not vary considerably during the time step. So, we can get the equation as $V_1(t+\Delta t)=V_0(t)+f\cdot\Delta t$ .The next step is the diffuse term. It is for the effect of viscosity and is equivalent to a diffusion equation. Viscosity is measure of how resistive a fluid is to flow. We can get the equation by the finite difference method as $V_2(t)=V_1(t)+(\Delta t\cdot\nu\nabla^2 V)/\rho$. The third step solves for the effect of advection (or convection) of the fluid on itself. The advection term is non-linear and cannot be approximated by the finite difference method. Here, we use the method of characteristics to solve partial differential equations. All the fluid particles are moved by the velocity of the fluid itself at each time step. It is necessary to calculate the velocity at a point x at the new time $t+\Delta t$. So, we backtrace the point x through the velocity field U_2 over a time Δt . This defines a path $p(x,s)$ corresponding to a partial streamline of the velocity field. The new velocity at the point x is set to the velocity at its previous location a time Δt ago. The equation is $V_3(t)=V_2(p(x,-\Delta t))$ .The final step is project term which is used to solve the pressure term. According to Helmholtz-Hodge decomposition theorem, any vector can be decomposed into a set of basis vector components. So, we can get the equation $w=u+\nabla p$. The new velocity field w with nonzero divergence to solve the NSE that involves three computations to update the velocity at each time step：force, diffusion, advection application. The velocity field u is free divergence velocity, and p is the pressure field. If we apply the divergence operator to both sides, then we obtain $\nabla w=\nabla\cdot(u+\nabla p)$. Because the velocity field u is free divergence ($\nabla\cdot u=0$), the equation can be simplified to $\nabla\cdot w=\nabla^2 p$. The equation is a Poisson equation for the pressure of the fluid. We can solve the equation for p, and then use w and p to compute the new divergence-free field u. So, the velocity equation of the final step is $V_4(t)=V_3(t)-\nabla p/\rho$.

3.2.2 Free surface

The free surface is the most influential factor for reality. So the classification of the free surface grid is used to distinguish the location of free surface. It is called full cell when the grid contains only liquid. The grid is located at the top of the free surface and the grid that contains no liquid is called empty cell. In addition, the grid that contains liquid and gas, and where at least one neighboring grid is empty is called the surface cell. We should determine whether the grid belongs to the above cells before the surface reconstruction. If the grid is a full cell, one can calculate the velocity according to the internal fluid field. On the other hand, it does not need to calculate when the grid is an empty cell. Besides, it should get the average density when the grid is surface cell, then calculate density by internal fluid field equation, $\rho=\rho_1+(\rho_2-\rho_1)f$, where ρ_1 and ρ_2 are the gas and liquid density respectively, and f is the fraction of fluid in the grid. We can get the free surface through the above description but the premise is that the grid type has already been determined. Therefore we have to determine the grid type using VOF, which defines an additional variable in the computing grid called fraction-of fluid function, f(x,z,t). The value of f is between zero and one. The value "0" expresses an empty grid, and we defined "1" as a full grid and "0~1" as a surface grid. Because the free surface moves over time, the value f will change with the free surface. The VOF gets the free surface through the definition of f for each grid, and the f in each cell is transported by the advection equation as $f_t+\mathrm{u}\cdot\nabla f=0$.

We reconstruct the free surface according to a moving interface reconstructing algorithm [15], but the reconstructing algorithm is only implemented on the closed boundary grid. Hence, we modify the algorithm for the ocean simulation (non-closed boundary).The grids whose value is between zero and one are first located. The intersection of the surface of the boundary line and the grid is called spot. The surface grids shown in Figure 4 have three types. Each type includes four patterns according to the relationship of the location of the surface grid and the adjacent grid.

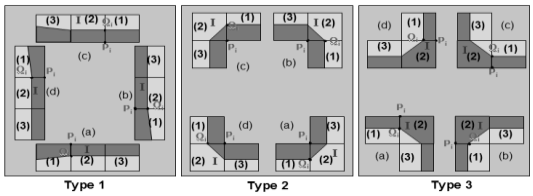

Figure 4：Three types for the free surface grid

The grid (1)(2)(3) is the order index for surface grid as shown in Figure 4. The grid I is the current boundary grid. Type 1 is characterized by "linear" that the current grid and the adjacent grid is a straight line parallel to the x axis or y axis. Type 2 is characterized by "convex" that the current grid and the adjacent grid are not straight but convex. Type 3 is characterized by "concave" that the current grid and the adjacent grid are not straight but concave. We define four different cases according to the different types. In order to obtain the spot, we define Q_i as the boundary spot. P is the grid node that locates at the adjacency of Q_i. The variable t_i is the reference value and is defined as follows.

$$t_i=\begin{cases}|P_iQ_i|/\Delta x & \text{, if } P_iQ_i \text{ is parallel to the x axis}\\ |P_iQ_i|/\Delta y & \text{, if } P_iQ_i \text{ is parallel to the y axis}\end{cases} \quad (1)$$

where Δx and Δy are the length of the grid x axis and y axis respectively. We record the related information of each Q_i, including the number of spot, type and the coordinate of $P_i\ (X_{P_i},Y_{P_i})$. Assuming the volume fraction of fluid of the I surface grid is f_i, we can build volume fraction of fluid as listed in Table 1 according to the type of the surface grid. After obtaining t_i one can calculate the coordinate of spot (X_{Q_i},Y_{Q_i}) according to the spot Q_i as shown in Table 2.

Table 1：Volume fraction of fluid

The I grid type	Keeping volume fraction of fluid
Type 1：Linear	$\frac{1}{2}(t_i+t_{i+1})=f_i$
Type 2：Convex	$\frac{1}{2}t_it_{i+1}=f_i$
Type 3：Concave	$1-\frac{1}{2}(1-t_i)(1-t_{i+1})=f_i$

Table 2：Formula of spot Q

The pattern of the *I* type	Spot *Q*	
	X_{Q_i}	Y_{Q_i}
(a)	X_{P_i}	$Y_{P_i}-t_i$
(b)	$X_{P_i}+t_i$	Y_{P_i}
(c)	X_{P_i}	$Y_{P_i}+t_i$
(d)	$X_{P_i}-t_i$	Y_{P_i}

Assuming the boundary grid, $\dot{i}$, is the volume fraction of fluid f_i. If the boundary grid is closed and the boundary is composed of the boundary grids, we can obtain *N* equations. By solving these equations one can obtain $t_i(i=1,2,\cdots,N)$ and the coordinate of Q_i by $t_{N+1} = t_1$. When the simulation has non-closed boundary, the equation would be $t_{N+1} \neq t_1$. Hence, we should obtain the spot of start gird before the operation. Here, we divide the start grid into three types and seven cases. And we can obtain the t_1 as shown in Figure 5 by the equations of the different cases. We divide the start grid into three types. Type 1 is "leftward", and the adjacent grid is located on the left start grid. Type 2 is located below the start grid, called "downward". Type 3 is located above the start grid, called "upward". The different types have different cases. According to the different cases one can calculate t_1 as shown in Table 3.

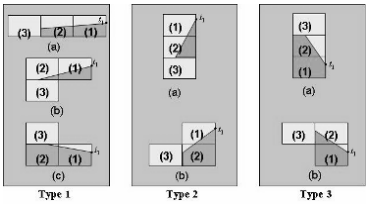

Figure 5：The status of the start grid

Table 3：Formula of t_1

	Type 1
(a)	$t_1=(3f_1-f_2)/2$
(b)	$t_1=2(f_1+f_2)\pm 2\sqrt{f_2^2+f_1f_2}$
(c)	$t_1=(2f_1+2f_2-3)+2\sqrt{f_2^2+f_1f_2-f_1-3f_2+2}$
	Type 2
(a)	$t_1=\left(f_1+\sqrt{f_1^2+f_1f_2}\right)/f_2$
(b)	$t_1=2f_1\pm 2\sqrt{f_1-f_1f_2}$
	Type 3
(a)	$t_1=\left((f_1-f_2)\pm\sqrt{f_1^2+f_1f_2-3f_1-f_2+2}\right)/(1-f_2)$
(b)	$t_1=(2f_1-1)\pm 2\sqrt{f_2-f_1f_2}$

3.2.3 Expansion to 3D

The slice method can generate the different 2D simulations, one can further integrate them to get a 3D shape. We can use the level of detail(LOD) scheme because LOD involves decreasing the complexity of a 3D object representation as it moves away from the viewer or according other metrics such as object importance, eye-space speed or position. Based on the LOD each patch will be determined by the distance to the viewer, one can use the larger grid on the 2D simulation when grid is farther away from the camera.

In contrast to the slice method that needs to process many slices, the proposed method only process some key slices. We apply linear interpolation with noise function to generate the new 2D simulations between two key slices. The formula is shown as follows.

$$P_{i+t,j}=P_{i,j}+\frac{(P_{k,j}-P_{i,j})}{k-i}t\ ,1\le j\le n \tag{2}$$

$$P'_{i,j}=P_{i,j-1}+(P_{i,j}-P_{i,j-1})Noise(j) \tag{3}$$

where $P_{i,j}$ is the i-th of 2D simulation result (outline), and the *j* is the order of the boundary. $P_{i,j}$ and $P_{k,j}$ are the two 2D simulations by VOF, and they need to interpolate 2D new slice $P_{i+t,j}$ between the i-th and the k-th simulation. By Eq.(2) which is the linear interpolation one can obtain the value of the interpolation. But the quality is insufficient, when only use the value of the interpolation to generate the 2D simulation, so a noise function is integrated into Eq.(3). The noise function makes the simulation more real.

The reconstructed surface may have aliasing and not smooth. Therefore, we should add some techniques to make the surface more smooth. Here, we use the Chaikin's algorithm to smooth the reconstructed surface.

3.3 Particle simulation

The particle system is used to simulate certain fuzzy phenomena on computer graphics. The particle status of splashes and foam is very important for the ocean simulation. In this paper, we only render the splash. We generate the particles according to the value of curvature, and the new splash particle has its lifespan. Then we update position over time. And the splash particles are rendered in white. In this way, we can effectively visualize splashes.

3.4 Render

In this step, we use the lighting and texture operations for the ocean simulation. The texture technique that we used is bump mapping. Bump mapping perturbs the surface normal of the object at each pixel, and is applied before the illumination calculation. In addition, it has been proved an effective way to improve the appearance of reality and not increase the complexity of the geometric objects. The technique has already implemented in the field of real-time graphics with the graphics hardware progresses and the application of shading language. In order to render the water, we should implement the reflective and the refractive effects. Environment mapping can help us to render the effect. One can calculate the normal vector to get the corresponding environment texture. Then one can mix the texture with water color and get the final color. The technique is easy to implement by GLSL, and the result is more realism.

4. IMPLEMENTATION

The simulation of breaking wave has a lot of formulas that are very complicated to implement for hydrodynamics. In this paper, we use the finite difference method to describe the formula of NES. Then, we can get the velocity at the center of each face of a grid and the pressure at the center of a grid. According to the correctness of velocity and pressure one can construct a flow field and do the ocean simulation.

4.1 2D simulations

4.1.1 Initialization

Before the simulation, the computation domain can be represented via a fixed rectangular grid aligned with a Cartesian coordinate system. To initial configurations of space such as mesh size, velocity field and pressure field, and prescribe initial conditions for the 2D NSE, one can create an ocean wave by [20]. At first, we denote the period, T , wavelength, λ , and amplitude, A . If

the frequency, $w = 2\pi/\lambda$, and the phase-constant $\varphi = 2\pi/T$ which expresses speed, the surface displacement for a wave is given as $W(x,t) = A\cos(wx - \varphi t)$. And the velocity components are given by

$$\begin{aligned} u(x,t) &= A\varphi e^{-wz}\cos(wx - \varphi t) \\ v(x,t) &= A\varphi e^{-wz}\sin(wx - \varphi t) \end{aligned} \quad (4)$$

where z is the depth, and we can obtain the equation when time is zero.

$$W(x,0) = d + \frac{1}{2\pi}\left(wA\cos(2\pi x) + \frac{1}{2}(wA)^2\cos(2\pi x) + \frac{3}{8}(wA)^3\cos(6\pi x)\right) \quad (5)$$

4.1.2 Discretization

We expand the equation via the finite difference method because the 2D simulation is driven by NSE, The NSE contains three different applications of the symbol ∇. The three applications are gradient, the divergence, and the Laplacian operators respectively, as shown in Table 4.

Table 4：Three finite difference forms

Operator	Definition	Finite Difference Form
Gradient	$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}\right)$	$\left(\frac{p_{i+1,j} - p_{i-1,j}}{2\Delta x}, \frac{p_{i,j+1} - p_{i,j-1}}{2\Delta y}\right)$
Divergence	$\nabla \cdot V = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$	$\frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x} + \frac{v_{i,j+1/2} - v_{i,j-1/2}}{\Delta y}$
Laplacian	$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$	$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\Delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\Delta y)^2}$

The subscripts i and j used in the expressions in Table 4 is used to discrete locations on a Cartesian grid, Δx and Δy are the size of the grid in the x and y dimensions, respectively, and $u_{i+1/2,j}, v_{i,j+1/2}$ are the location of staggered velocity components on a grid. This discretization leads to an explicit finite difference approximation of Eq.(3).

4.1.3 Boundary conditions and surface grid

Boundary condition is set on the boundary between the obstacle and the water. Assume the obstacle is non-slip, one can make the gradient of pressure equal zero on the boundary and the velocity on the boundary is zero.

The pressure of the empty and surface grid are set to atmospheric pressure, and the velocity on each of its faces shared with another empty grid, which is set to zero and the velocity should satisfy the conservation of mass. As shown in Figure 6, Type 1 shows the velocity on four sides of the two dimensional surface grid. Three of them are surrounded three sides by grids containing fluid. In order to keep the conservation of mass, the velocity of air v_2 set to $v_2 = v_1 - (u_2 - u_1)$. When the grid has two sides which face an empty grid, it is type 2. Each open side velocity equals the velocity of the side of the grid opposite it. And when the grid has three open sides, it is type 3. The side velocity equals the velocity of the side that opposites it

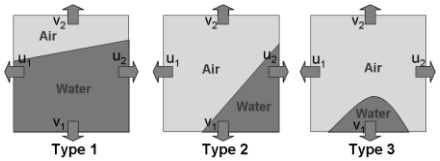

Figure 6：Setting the velocity on the free surface

4.2 GPU implementation

The efficiency is lower when only using CPU, because of the complex operations for NSE. General-purpose computing on graphics processing units (GPGPU) is the technique of using a GPU, which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the CPU.

We utilize GPU to improve for the complicated operations. In the GPU pipeline, the traditional end point of every rendering operation is the frame buffer, a special chunk of graphics memory from which the image that appears on the display is read. The frame buffer can get is 32 bits of color depth, shared among the red, green, blue and alpha channels. The data will always be clamped to the range of [0;1]. Fortunately, OpenGL extension provides 32-bit floating point values on GPUs. Each RGBA channel of offscreen buffer supports 32-bit floating point values for vector calculations. We commonly call the technique FBO.

Arrays in GPU memory are called textures or texture samplers. Texture dimensions are limited on GPU, the maximum value in each dimension is 2048 or 4096. The data elements which are accessed from texture memory, one needs to choose a special projection from the 3D world to the 2D screen. Pixels and texels are one to one mapping. To choose an orthogonal projection and a proper viewport, one will enable a one to one mapping between geometry coordinates and texture coordinates and pixel coordinates. To use a texture as render target to attach the texture to the FBO, the FBO extension provides a very narrow interface to render to a texture. Then we transfer data to a texture from CPU to GPU. The operations that are calculated on GPU can be achieved by shading language. The shading language divides into two parts(vertex and fragment shader), and we can code according to the requirement and transfer from GPU textures to CPU arrays or render.

GPGPU can help us to do the complicated arithmetic, but one thing should be noted. It is expensive to transfer data from GPU to CPU. So, we must consider very carefully before doing GPGPU. We calculate the two dimensional NSE solver on CPU and GPU respectively. Table 5 lists the comparison of computation time between CPU and GPU. Here we divide the GPU process into three major items, the NSE fragment, texture transmission and glReadPixels function. GPU spends most of time on glReadPixels function. The computation time on CPU is about 15.9 milliseconds and GPU is about 2 milliseconds. The result is calculated on the grid size 100 × 100. It shows the excellence floating-point operation of GPU.

Table 5： Computation time between CPU and GPU

Operation	Run NSE fragment	Texture transmission	glReadPixels
GPU	Less than 0.5ms	0.25ms	0.7ms
CPU	15.9ms		

4.3 Results

We generate a 2D initial wave and the parameters $\varphi = 2$ and $w = 3$. The grid size is 100×100. Figure 7 shows the 2D wave. The left picture is at time 2.8 and the right picture is at time 9.5. The experiments were run in PC with 3.4GHz Pentium IV CPU and NVIDIA GeForce 8800GTS display card. The performance is about 44.15 frames per second.

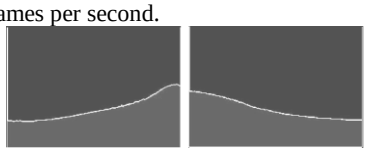

Figure 7：Snapshots of the 2D wave

The 3D simulation result is through combining with the different 2D results of the proposed algorithm. Figure 8 shows the result. The advantage of the proposed algorithm is that users can choose

the 2D results to determine the variation of wave. Figure 9 shows the FPS measurements for the different slice counts. These slices would be uniformly distributed on the interval of z-axis grid size. And the grid size is $100 \times 100 \times 50$ and the number of triangles is about 148000.

Figure 8：Snapshots of the 3D wave

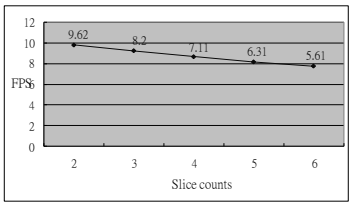

Figure 9：FPS measurements for the different slice counts

NSE can be divided into four steps with the time complexity is listed in **Table 6**. Project step uses the PCG method to reduce both the space complexity and time complexity per iteration from $O(n^2)$ to $O(m)$, where n is the grid size, m is the number of nonzero entries of A, and κ is the iteration times. Our algorithm efficiency is competitive compared with the previous method of the slice method [20]. In [20], the free surface is calculated by CLSVOF. The result is more realism but needs more computational time. In the proposed algorithm, we use VOF to solve the free surface. The VOF method is known for its ability to conserve the "mass" of the traced fluid. Although the reality is less, the computational time is twice less than that of CLSVOF.

Table 7 lists comparisons of time complexity for the proposed and the slice methods. We assume the time complexity of 2D NSE and the surface reconstruction is $O(K)$, so the time complexity of the proposed method and the slice method is $tO(K)$ and $hO(K)$ respectively, where t is the number of key slice and h is the length of z axis ($t \le h$). Here we assume the length of z-axis on 3D space is fifty. The slice method needs fifty slices and the proposed method would use 2~6 slices to expand 3D environment. The computation time of the proposed method is faster than the slice method. Although fewer slices would reduce the reality, the work efficiency would increase. And the proposed method is more flexible due to users can decide to control slices count. In order to reduce aliasing which producing by linear interpolation, we add the noise function to make the free surface more natural.

Table 6：Time complexity for different solutions of NSE

NSE	Force	Diffuse	Advect	Project
Method	Finite difference	Finite difference	Method of characteristics	PCG
Time complexity	$O(n)$	$O(n)$	$O(n)$	$O(\kappa m)$

Table 7：Time complexity of the proposed method and the slice method

Method	The proposed method	The slice method
Time complexity	$tO(K)$	$hO(K)$
Computation time	5~9 frames/s	1 frame/s

We render the environment mapping, the reflective and the refractive effect of the water by GLSL. The reflective and the refractive effect of the water are mix environment texture with water color. And we use bump mapping to increase the reality. The technique is simple and yields good results. Figure 10 shows the render effect. And Figure 11 increases the splash effect by the particle system.

Figure 10：Render effect

Figure 11：Snapshots of the breaking wave

5. CONCLUSIONS

In this paper, we have presented a slice-based water simulation method for ocean breaking wave on natural simulation. By combining linear interpolation with noise function to construct a complete three dimensional ocean simulation, one can reduce computational time and achieve better efficiency. The proposed algorithm can improve the computational time of the slice method. And we also avoid the aliasing using linear interpolation with noise function. The proposed method is more flexible to users, because users can determine how many slices are used. Users can generate simple ocean waves by a few slices, or complicated ocean waves by more slices. No matter what users choose, the ocean wave is still realism.

People can design the different ocean waves by the slice method, but it would spend much time and resources. The proposed method can speed up the computation time as fast as 9 times and make the simulation efficiency. And this method still can reserve the reality because it uses the noise function to increase the variation of water.

6. REFERENCES

[1]. C. Johanson, "Real-time water rendering," *Master of Science paper*, Lund University, 2004.

[2]. D. Enright, S. Marschner, and R. Fedkiw, "Animation and Rendering of Complex Water Surfaces," *Proc. of the ACM Transactions on Graphics*, vol. 21, no. 3, pp. 736–744, 2002.

[3]. D. Kim, O.Y. Song, and H.S. Ko, "A Semi-Lagrangian CIP Fluid Solver without Dimensional Splitting," *EUROGRAPHICS 2008*, vol. 27, no. 2, 2008.

[4]. G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw, "Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques," *Proc. of the ACM Transactions on Graphics*, vol. 25, 2006.

[5]. J. Stam, "Stable Fluids," *Proc. of ACM SIGGRAPH '99*, pp. 121–128, 1999.

[6]. J. Tessendorf, "Simulating Ocean Water," *Proc. of ACM SIGGRAPH 2002 Course Notes 9* (Simulating Nature: Realistic and Interactive Techniques), 2002.

[7]. L.S. Jensen, and R. Golias, **"**Deep-Water Animation and Rendering," *In Gamasutra*, 2001.
[8]. M. Kass, and G. Miller, "Rapid, Stable Fluid Dynamics for Computer Graphics," *Proc. of the ACM Transactions on Graphics*, vol. 24, no. 4, pp. 49–55, 1990.
[9]. N. Foster, and D. Metaxas, "Modeling the motion of a hot, turbulent gas," *In Proc. of SIGGRAPH '97*, pp. 181–188, 1997.
[10]. N. Foster, and R. Fedkiw, "Practical animation of liquids," *Proc. of ACM SIGGRPAH*, pp. 23–30, 2001.
[11]. N. Thürey, F. Sadlo, S. Schirm, M. Müller-Fischer, and M. Gross, "Real-time Simulations of Bubbles and Foam within a Shallow Water Framework," *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007.
[12]. N. Thürey, M. Müller-Fischer, S. Schirm, M. Gross, and ETH. Zurich, „Real-time Breaking Waves for Shallow Water Simulations," *15th Pacific Conference on Computer Graphics and Applications*, 2007.
[13]. Q. Wang, Y. Zheng, C. Chen, T. Fujimoto, and N. Chiba, "Efficient rendering of breaking waves using mps method," *Journal of Zhejiang University SCIENCE A*, 2006.
[14]. R. LeVeque, "Finite volume methods for hyperbolic problems." *Cambridge Texts in Applied Mathematics*, *Cambridge University Press, Cambridge*, 2002.
[15]. R. Wang, P. Chen, and C. Ban, "Keeping Volume Fraction of Fluid in Reconstructing Moving-interfaces of VOF on Rectangular Meshes," *Chinese Journal of Computational Physic*, vol. 25, no. 4, 2008.
[16]. S. Jeschke, H. Birkholz, and H. Schmann, "A procedural model for interactive animation of breaking ocean waves," *Proc. WSCG2003 POSTERS*, 2003.
[17]. T. Klein, M. Eissele, D. Weiskopf, and T. Ertl, "Simulation, modelling and rendering of incompressible fluids in real time," *Workshop on Vision, Modelling, and Visualization VMV '03* pp. 365-373, 2003.
[18]. T. Nakayama, and M. Mori, "An Eulerian finite element method for time dependent free surface problems in hydrodynamics," *Int. j. numer. methods fluids*, vol. 22, pp. 175–194, 1996.
[19]. T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki, "Realistic animation of fluid with splash and foam," *Computer Graphics Forum*, vol. 22, no. 3, 2003.
[20]. V. Mihalef, D. Metaxas, and M. Sussman, "Animation and Control of Breaking Waves," *Proc. of the ACM SIGGRAPH/ Eurographics Symposium on Computer Animation*, pp. 315–324, 2004.

About the author

Chung-Nan Lee is a professor at National Sun Yat-sen University, Department of Computer Science and Engineering. His contact email is cnlee@cse.nsysu.edu.tw.

Jyun-Ming Wu is a Master student at National Sun Yat-sen University, Department of Computer Science and Engineering. His contact email is piiwu2002@yahoo.com.tw.

Yan-Zheng Huang is a Master student at National Sun Yat-sen University, Department of Computer Science and Engineering. His contact email is lewis005299@hotmail.com

Da-Jing Zhang-Jian is a Ph.D. student at National Sun Yat-sen University, Department of Computer Science and Engineering. His contact email is salmoner.tw@yahoo.com.tw.