國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

應用虛擬實境平台於基於粒子組成的海面上之船舶模擬

Particle-based Ocean Waves for Ship Motion Simulator in VR Environment

林孟勳

Meng-Hsun Lin

指導教授：傅立成 博士

Advisor: Li-Chen Fu, Ph.D.

中華民國 98 年 7 月

July, 2009

誌謝

到了此刻，回頭看這兩年說長不長說短不短，實驗室的時光。回憶爭先恐後似的，一併湧上了心頭。在這裡獲得了許多寶貴的知識與經驗，卻也有很多東西還沒有經歷過，得到的同時似乎也失去了什麼。無論如何，首先最感謝敬愛的指導教授傅立成博士，老師諄諄教誨的親切以及孜孜不倦的精神不論是在身教或是言教上，都是我們一個親身體會並且學習的榜樣，猶記得老師在口試前更是為了指點我不眠不休而直至深夜，在這裡再次跟老師說聲：老師，您辛苦了，謝謝老師。祝老師身體安康，順心如意。

同時也非常感謝口試委員歐陽明、陳炳宇、郭振華以及范欽雄教授另外還有開設程式課程的陳俊杉教授，在我的修課以及口試時，非常親切的建議以及指導，讓我可以看到自己的不足，也看的更廣更深更遠。

立成盃的汗水與歡呼，出遊的笑談與義氣，demo 前的奮鬥與日出，考試前的討論與鼓勵，點點滴滴讓這兩年的時光顯得如此珍貴。不管是打球還是吃飯，研究還是揪團，實驗室夥伴鮮明的臉龐和交談的話語總是讓我又充滿了活力。特別感謝 VR 組聖化、恩暐、哲民、靖堂、顯真、嘉鳴、志鴻、平昇，每次的討論中聖化以及恩暐學長總是不厭其煩地指點我這個經驗值太低的學弟，讓我得以不斷地吸收知識及修正方向，聖化學長還特地在口試前逐字逐句校閱我的論文，恩暐學長也是不斷地提供他的經驗以及中肯的見解，也同時感謝平昇學弟熱情的幫忙。也感謝實驗室智富、銘全、明理、正民、定國、俊緯、威文、政昌等學長，不管是課業或是生活中，他們適時的照顧與關懷，總能讓我們感受到支持與溫暖。我們這一屆的志鵬、柏男、繕琮、柏徐、世勳、羿如，感謝你們，相互扶持的友誼，互相玩鬧的歡笑，一起討論的成長，一同奮鬥的努力，謝謝你們讓我學到了

很多也讓你們照顧了很多，恭喜我們也祝福我們。當然也要感謝冠霖、楨惇、又生、秉淳學弟，你們的幫忙與支持也是實驗室中不可或缺的笑聲及動力。

感謝栗兒的愛與包容，你的支持一路陪我度過了碩二下的時光，在我徬徨或是挫折時能夠再度充滿活力，面對挑戰。最後感謝親愛的家人，無條件地照顧和支持我，讓我得以順利完成兩年的學業，爸，媽，謝謝你們，謝謝。弟，開心長大吧，妹，快輪到你了，阿公阿嬤，不用擔心，你們的孫子要畢業了。

要感謝的實在太多了，套一句實驗室老梗，總之～謝啦。祝各位平安喜樂，順心如意。

摘要

本篇論文主要建構出基於粒子組成的海面波浪，這種波浪的特性是適合物理上與船體也就是剛體的互動模擬。我們的目標在於架構一個在虛擬實境中的即時動態模擬船舶駕駛系統，而我們提出了一些新的方法來達到這個目標。首先，為了模擬大自然中主要被風所吹動的海浪，我們試圖為風力的影響提出一個模型，而讓這風作用在粒子組成的海面上，自然地形成波浪的運動。同時波浪也會受到重力或是本身黏滯力的影響。我們對於流體的模擬主要是基於平滑粒子流體力學(SPH)方法，也會在論文中做相關的介紹。其次，我們必須繪製流體的表面，而我們介紹了一種使用軟性材質的網格來建構流體表面的新方法。接著我們計算跟處理剛體與流體、風力、海面間的互動碰撞來模擬船體的動態，如此可以獲得更真實的結果。最後，這個模擬系統可以整合到六自由度的史都華平台，而建構出一個虛擬實境中的船舶駕駛模擬器。

關鍵字: 虛擬實境、基於粒子組成的、海面波浪、風力、平滑粒子流體力(SPH)、流體表面、船舶模擬器

Abstract

This thesis focuses on the construction of particle-based ocean waves, which can be applied to interact with the boat in physical simulation. The goal is to construct a real-time physical dynamic system of boat motion simulator for virtual reality application. We introduce a new method to create ocean waves with particle-based fluid. First, in order to simulate wind-driven ocean surface, we use a wind force model to simulate the effect of actual wind, and let the force act on the particle-based fluid to create waves animation. Additionally, fluid dynamics of waves are also influenced with gravity force or viscous force. The particle-based fluid simulation is based on SPH (Smoothed Particle Hydrodynamics) method, which is introduced in this thesis. Second, we consider the issue of drawing fluid surface, and present a new method to construct the surface with physically simulated cloth. Then we can create the interaction between solid, fluid, force, and surface to simulate the dynamics of the boat, and we can get more realistic results. Finally, this simulation can also be integrated with the 6 degree-of-freedom motion platform to build a ship motion simulator.

KeyWords: virtual reality, particle-based, ocean waves, wind, smoothed particle hydrodynamics (SPH), fluid surface, ship motion simulator

Table of Contents

List of Figures

Chapter 1
Introduction

1.1 Background and Motivation

As time evolves, with progress of the science and technology, more and more virtual-reality applications have been developed. The technology of virtual reality is in fact widely used in modern life, including driver training, education, industry, entertainment, medical etc., and becomes more acceptable to people. Among kinds of applications, vehicle simulator is an important achievement in the field of virtual reality today. Usually, planes, cars, and ships are the most popular and common simulators that have been designed for mobile vehicles. The flight and the driving simulator models were developed in early stage, and have been commonly used even until now. Our work focuses on ship's motion simulator.

Since ancient time, ship is very important in transport, fisheries and wars. The history of ships is even older than which of ancient Egypt and China. There are several great navigators who appeared in history. Around fifteenth-century, Zheng He in Chinese Ming dynasty navigated to the Indian Ocean. After a century, Christopher Columbus discovered American continent. In the seventeenth-century, Portuguese navigator, Magellan, led the fleet around the world. Although Columbus didn't know

that he discovered a new continent and Magellan died in Philippines, but they both left remarkable legends in navigation pages.

In order to construct the ship simulator, we require developing the ocean simulation preliminarily. Ocean is unpredictable and changing at various locations and in various climates. Currently, the challenge of developing the ship simulator is the ocean waves modeling, and the behavior of ocean waves is still frequently discussed and continuously analyzed. Although the field of fluid simulation becomes more mature recently, and the realistic simulation results have been applied to many fields such as animation, games and movies, they still have some limitations. Some simulation results use more completed physical equation and complicated methods, and produce amazing rendering performance, but they have excessive computation load and have to do offline calculation. Some are well-rendered and real-time simulated, but cannot do the physical interaction between fluid and rigid body. Generally speaking, there is a trade-off between realistic behavior and real-time performance.

The properties of fluid flows are described with hydrodynamics, and CFD (Computational Fluid Dynamics) uses numerical methods and algorithms to solve these equations. In 1822 Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes equation which describes momentum conservation of fluid.

Besides, two additional equations required to be considered. One is continuity equation which describes mass conservation, and the other is state equation which describes energy conservation. The behavior of fluid is described with these partial differential equations. Traditionally, Eulerian (grid-based) method and Lagrangian (particle-based) method are two main approaches to solve these equations. In this thesis, we use Lagrangian method, since Eulerian method is unfortunately hard to achieve fine interactive frame rate. But in virtual reality or other similar applications, real-time performance is very important. In general, factors which influence researchers to select simulation methods include implementation of programming, computational cost, controllability and the realistic descriptions.

Lagrangian (particle-based) method uses particles system to simulate fluid, besides, particle system can be used to simulate other fuzzy phenomena, such as fire, explosions, and smoke. The property of Lagrangian method is that it makes mass conservation equations, and convection terms could be dispensable. Even though this reduces the complexity, the computational cost is still expensive. We adopt the GPU hardware acceleration to achieve real-time results. We use SPH (Smoothed Particle Hydrodynamics) method to calculate some required quantities of fluid particles for the advantage of hardware acceleration. For particle system, it is an interpolation method to determine field quantity.

There are also some approaches to simulate ocean waves, and the spectral method is one of them. But the drawback is homogeneity: cannot perform local properties and hard to interact with other objects. We use Lagrangian method to simulate ocean waves, two main challenges are how to create waves and how to construct ocean surface. This thesis attempts to create wind force acting on fluid and solves fluid dynamics equations, we use this method to create waves animation. The behaviors of fluids like surface tension, viscosity, wave reflection, can be achieved by particle-based approach. We use impulses to process solid-fluid interactions. Hence the interaction between the rigid body and fluid can be simulated, and the rigid body also interacts with our wind force.

The construction of fluid surface is also an interesting problem in computer graphics, several techniques have been developed. In this thesis, we use two ways to construct surface. One is marching cubes algorithm, and the other is a new method which use a physically simulated cloth to fit the surface of particle-based fluid. When rendering the surface, reflection and refraction are main phenomena to consider. In order to achieve water visual effect, we use Cg language to do the shading.

Finally, we integrate the physical simulation and computer graphic system with the 6-degree-of freedom motion platform and driving cabin to build a ship motion simulator.

1.2 Related Works

Ocean waves are propagating disturbances of the water surface, principally generated by wind. One category for modeling and animating the waves is based on ocean wave models, and consists of three approaches. The first approach uses the Gerstner Swell model which is established in 1802. The model describes the motion of each water particle as a circle around a fixed point. Fourier and Reves [1] concentrated on shallow water waves. More complex parametric equations to present the propagation of water waves have been introduced by Gonzato and Le Saec [2].

Another group corresponds to spectral approaches, Mastin et al. [3] introduced an effective simulation of wave behavior using Fast Fourier Transform(FFT). The basic idea is to produce a height field having the same spectrum as the ocean waves, they used the Pierson-Moskowitz filter [4], obtained from a real ocean waves spectrum, for filtering a white noise image in the frequency domain. Tessendorf [5] showed that dispersive propagation could be managed in the frequency domain and that the resulting field could be modified to yield trochoid waves. However, the negative aspect of FFT based methods is homogeneity: cannot handle any local properties such as refraction, reflection, and others physical interaction.

The last one is the hybrid approach: The spectrum synthesized by a spectral approach is used to control trochoid waves generated by the Gerstner model [6].

Hinsinger [7] presented an adaptive scheme for the animation and display of ocean waves in real time. In these approaches, some smaller scale waves are obtained by directly tuning some extra Perlin noise [8].

Another family uses fluid dynamics equations to simulate waves. Claude Navier (1822) and George Stokes (1845) formulated the famous Navier-Stokes Equations that describe the dynamics of fluids, which related to conservation of momentum. Physical fluid simulation is adapted to these equations. There are two general approaches in analyzing fluid mechanics problem [9]. The first one is Eulerian method, which has become popular due to a series of papers by Foster and Metaxas [10, 11]. This is also called grid-based or mesh-based method. Stam [12] improved on this method by introducing a semi-Lagrangian technique and implicit solvers, allowing for large time steps. Recent efforts focused on improving the efficiency to enable more complex simulations. Some examples include the use of octree data structures [13], coupled 2D and 3D simulations [14] and dynamic non-uniform mesh refinement [15].

The second method, called Lagrangian method, decompose the fluid into particles or small fluid elements. It is also called particle-based method. Lucy and Gingold et al. [16, 17] introduced Smoothed Particle Hydrodynamics (SPH) to estimate each particle's properties such as pressure, viscosity and density. Miller and Pearce create solids, deformable objects and fluids by tuning the manner in which

particles interacted with one another [18]. Their particle forces are similar to Lennard-Jones forces: particles are very close together repel one another, but at moderate distances they are attracted to each other, with the attraction falling off with greater distances. Desbrun, Cani and Tonnesen use particles to animate soft objects [19, 20]. They extended the formalism to the animation of inelastic bodies with a wide range of stiffness and viscosity. They defined the object's surface as an iso-surface of the mass density function, and then showed smoothed particles.

Muller et al. [21] also considered surface tension in this approach which could simulate 5000 particles at interactive rates. The fluid surface was rendered by the marching cube method at the end [22]. Premoe et al. used a moving particle semi-implicit (MPS) technique to simulate the flow of distinct kinds of fluid [23]. The Navier-Stokes equation was recast to the interactive particle format. This method for fluid simulation and application has become popular and was used extensively [24, 25]. Our work is based on Muller et al. [21].

Davidson [26] is the first person presented the ship steering theory. Abkowitz [27] adopted the multivariable Taylor series expansion of the forces and torques about some initial equilibrium condition. It can be used in large ship simulator. Hirano and Inoue developed the compact mathematical model, which included related hydrodynamic effects. They demonstrated the sufficient accurate theoretical and

empirical formula based on experimental tests [28, 29]. Fossen presented marine control systems in his publication. Browning [31] modified the mathematical model so that it could be used in small boat. He also presented the simulation framework which could be expanded to other marine vehicles

The research of the relationship between waves and wind is developed and organized into physical books [32-33]. It is part of physical oceanography science. On the other hand, the Cg language provides the developers with a programming platform. It is easy to program GPU for real-time rendering. There are many techniques and tips have been developed [34-35].

The 6-DOF motion platform is usually used for the simulation of the vehicles. In the past researches of our lab, Wang designed a system of the wheeled vehicles with the platform [36]. Later, Lee developed a physical engine for rigid body simulation involving aerodynamic [37]. Then, Chou construct an ocean scene for ship simulation [38]. Most recently, Chen improved the two-way coupling of the ship simulation [39].

1.3 Thesis Organization

The organization of this thesis is as follows. In Chapter 2, we first introduce the fluid simulation by using smoothed particle hydrodynamics, and we present a wind force model to generate wind acting on particle-based fluid. Chapter 3 mentions that how we construct the ocean surface and render it. Chapter 4 describes the necessary mathematical preliminaries for the rigid body dynamics, and introduces impulse-based theory to process the collision, contact of rigid bodies. So the solid-fluid two-way coupling can be achieved. In Chapter 5, we present all of implementation details which contain program structure and algorithms, for particle-based fluid, wind force, ocean surface, rendering and the interactive ship. We describe methods to achieve equations in Chapter 2-4, and improve the work of predecessors. Chapter 6 shows the graphical and physical result. Discussion, conclusion and future work are described in Chapter 7.

Chapter 2
Particle-Based Waves Simulation

We use particle-based fluid to do our simulation, and it is a method to solve hydrodynamics equations. This kind of research belongs to Computational Fluid Dynamics (CFD) and has a long history. In 1822, Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes equation which describes the dynamics of fluid. Besides the conservation of momentum described in the Navier-Stokes equation, two additional equations, continuity equation describing mass conservation and state equation describing energy conservation, are required to consider. Since those equations are known and computers are available to solve them numerically, a large number of methods have been presented in CFD literature to simulate fluid on computers [21].

There are two general approaches in analyzing fluid mechanics problems [9]. The first method, called Eulerian method, uses the field concept. In this case, the fluid motion is given by completely describing the necessary properties (pressure, density, velocity, etc.) as functions of space and time. With this method, we obtain information about the flow at fixed points in space as the fluid flows pass those points, but it is not appropriate to our simulation. The reasons will be mentioned later.

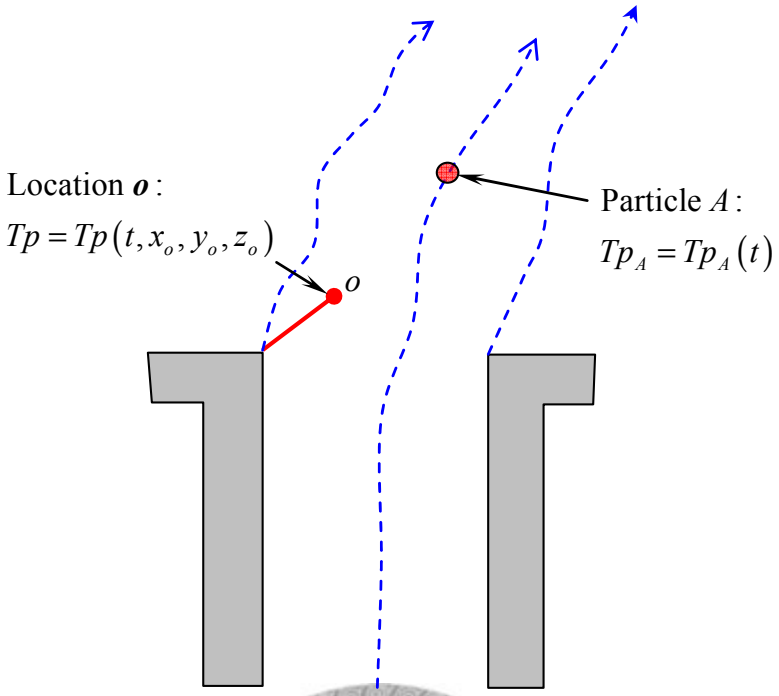


Figure 2-1: Eulerian and Lagrangian description of flowing fluid.

The second method, called Lagrangian method, involves fluid particles and determines how the fluid properties vary as a function of time. That is, the fluid particles are tagged or identified, and their properties determined as they move.

The difference between the two methods can be seen in the example of smoke discharging from a chimney, as is shown in Figure 2-1. In Eulerian method one may attach a temperature-measuring device to the top of the chimney (point ***o***) and record the temperature at that point as a function of time. Namely, $Tp = Tp(t, x_o, y_o, z_o)$. The use of numerous temperature-measuring devices fixed at various locations would provide the temperature field, $Tp = Tp(t, x, y, z)$.

In Lagrangian method, one would attach the temperature-measuring device to the

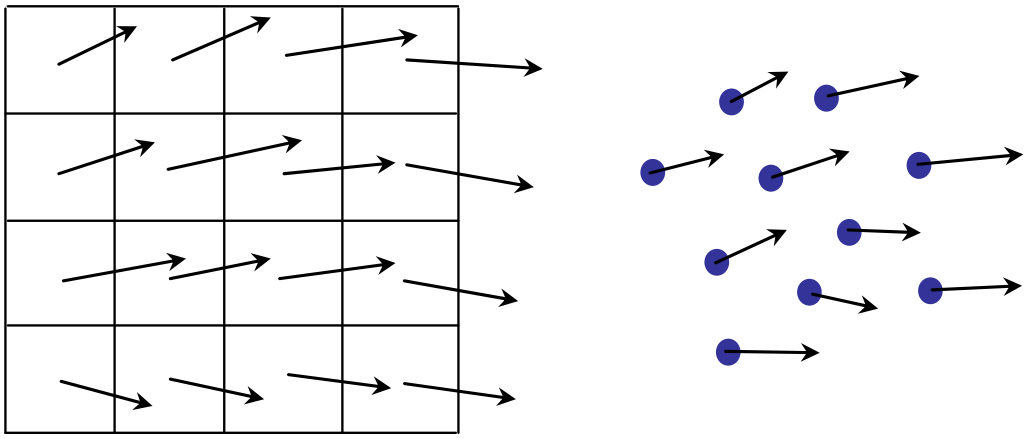


Figure 2-2: Eulerian grid-based description (left) and Lagrangian particle-based (right) description in CFD.

particular fluid particle (particle A) and record that particle's temperature as it moves. Thus, one would obtain that particle's temperature as a function of time, i.e., $Tp_A = Tp_A(t)$. The use of many such measuring devices moving with various fluid particles would provide the temperature of these fluid particles as a function of time.

In CFD, Eulerian method is typically referred to grid-based method, and the finite element method or finite difference method is the analysis tool. Lagrangian method is typically called particle-based methods, and the particle systems dynamics is the main tool. Figure 2-2 shows the Eulerian grid-based and Lagrangian particle-based descriptions in CFD.

Since we have to solve the equations, determining the properties of fluid such as density or viscosity becomes an important prerequisite. We can use Smoothed Particle Hydrodynamics (SPH) method to satisfy the requirement. The advantages of SPH over grid based approaches can be summarized as follows [40]. First, SPH is

conceptually both simple and elegant. All of the equations can be derived self-consistently from physical principles with a few basic assumptions. As a result, complex physics is relatively simple to incorporate. Its simplicity means for the user it is a very intuitive numerical method which makes itself easily to modify with various problem. Second, flexibility is built-in feature. If we want to modify density of fluid or flow morphology, it does not require to do mesh refinement or other complicated procedures. The nature of Lagrangian method is automatically accessed. As a result of its flexibility, SPH is also very efficient since it concentrates on regions of high density while computational effort is not wasted on empty regions of space. Third, free boundaries are simple and natural in SPH but typically suffer difficulties with grid-based method. SPH is also easy to integrate with physical system. Finally, visualization and analysis are also somewhat easier with particle-based methods, since it is a simple matter to track and visualize portions of the flow. These are the reasons we use this method, but the drawback is hard to form the mesh surface.

In this chapter, we will use particle systems to simulate the behavior of fluid. Particle systems refer to a computer graphics technique which uses particles to simulate certain fuzzy phenomena, such as fire, explosions, and smoke. It is more appropriate to do interactive simulation than grid-based systems. We will also present how to model wind force acting on these particles, and do waves simulation.

2.1 Smoothed Particle Hydrodynamics (SPH)

We use SPH method to determine the properties of fluid such as density or viscosity, and here we describe this method in detail. Smoothed Particle Hydrodynamics (SPH) was developed by Lucy [16], Gingold and Monaghan [17] for the simulation of astrophysical problems. Although it is not originally developed for CFD, it is general enough to be used in any kind of fluid simulation. SPH belongs to the class of Lagrangian approach. In simulations based on this approach, fluid is represented by a set of particles. SPH is an interpolation method for particle systems, and it allows one to compute the value of any field quantity such as density, at arbitrary positions in the fluid by smoothing over the set of nearby particles.

The basis of the SPH approach is given as follows [40-42]:

$$A(\mathbf{r}) = \int A(\mathbf{r}')\delta(\mathbf{r}-\mathbf{r}')d\mathbf{r}' \tag{2.1}$$

where A is any field quantity defined on the spatial coordinate $\mathbf{r}$, and δ refers to the delta function. This integral is then approximated by replacing the delta function with a smoothing kernel W with characteristic width h, such that

$$\lim_{h\to 0} W(\mathbf{r}-\mathbf{r}', h) = \delta(\mathbf{r}-\mathbf{r}') \tag{2.2}$$

giving

$$A(\mathbf{r}) = \int A(\mathbf{r}')W(\mathbf{r}-\mathbf{r}', h)d\mathbf{r}' + O(h^2) \tag{2.3}$$

The kernel function is normalized according to

$$\int W\left(\mathbf{r}-\mathbf{r}',h\right)d\mathbf{r}'=1 \tag{2.4}$$

The integral equation (2.3) can be discretized into a finite set of interpolation points (the particles). Replacing the integral with a summation and the mass element ρdV replaced with the particle mass *m*, i.e.,

$$\begin{aligned} A(\mathbf{r}) &= \int \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')} W\left(\mathbf{r}-\mathbf{r}',h\right)\rho(\mathbf{r}')\,d\mathbf{r}' + O\left(h^2\right) \\ &\cong \sum_j m_j \frac{A_j}{\rho_j} W\left(\mathbf{r}-\mathbf{r_j},h\right) \end{aligned} \tag{2.5}$$

From above equation, we can easily realize that any field quantity *A* at position **r**, can be obtained by the interpolation of quantities of its neighboring particles within the range of the smoothing kernel. The summation interpolation of (2.5) is the basis of all SPH formalisms. The gradient of *A* can be calculated by taking the analytic derivative of (2.5), namely,

$$\begin{aligned} \nabla A(\mathbf{r}) &= \frac{\partial}{\partial \mathbf{r}} \int \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')} W\left(\mathbf{r}-\mathbf{r}',h\right)\rho(\mathbf{r}')\,d\mathbf{r}' + O\left(h^2\right) \\ &\cong \sum_j m_j \frac{A_j}{\rho_j} \nabla W\left(\mathbf{r}-\mathbf{r_j},h\right) \end{aligned} \tag{2.6}$$

and the Laplacian of *A* can be calculated with the same procedure as follows:

$$\begin{aligned} \nabla^2 A(\mathbf{r}) &= \frac{\partial}{\partial \mathbf{r}^2} \int \frac{A(\mathbf{r}')}{\rho(\mathbf{r}')} W\left(\mathbf{r}-\mathbf{r}',h\right)\rho(\mathbf{r}')\,d\mathbf{r}' + O\left(h^2\right) \\ &\cong \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W\left(\mathbf{r}-\mathbf{r_j},h\right) \end{aligned} \tag{2.7}$$

As shown in the above two equations, with the SPH approach, derivatives of field quantities only influence the smoothing kernel. The errors introduced by the

approximation (2.5) can be estimated [40-42].

2.2 Field Quantity of Particle Fluid

In order to get more realistic results, we consider physical hydrodynamics to simulate fluid. The governing equation which describes the Newtonian fluid with viscosity is the famous Navier-Stokes equation. The equation formulates conservation of momentum:

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t}+\mathbf{v}\cdot\nabla\mathbf{v}\right)=-\nabla p+\rho\mathbf{g}+\mu\nabla^2\mathbf{v} \tag{2.8}$$

where **g** is gravity, or can be seen as an external force density field, μ is the viscosity of the fluid, ρ is the density field, p is the pressure field, and **v** is the velocity field. The continuity equation which describes the conservation of mass is expressed as:

$$\frac{\partial \rho}{\partial t}+\nabla\cdot(\rho\mathbf{v})=0 \tag{2.9}$$

For incompressible flow, the density is constant, and then the velocity field is divergence free, *i.e.*

$$\nabla\cdot\mathbf{v}=0 \tag{2.10}$$

The continuity equation (2.9) and the Navier-Stokes (2.8) equation are the governing equations for the Newtonian and isothermal fluid. The use of particles instead of a

stationary grid simplifies these two equations substantially. First, because the mass of each particle is constant and the number of particles is also constant, the conservation of mass is guaranteed and (2.9) can be omitted completely. Furthermore, the substantial derivative operator [9] on the left hand side of (2.8), $\frac{D}{Dt}=\left(\frac{\partial}{\partial t}+\mathbf{v}\cdot\nabla\right)$, can be replaced by $\frac{d}{dt}$. This is because the particles move with the fluid, the substantial derivative of the velocity field is simply the time derivative of the velocity of the particles. Thus, the convection term $\mathbf{v}\cdot\nabla\mathbf{v}$ is not needed for particle systems [21]. Therefore, the particle dynamics becomes

$$\rho\frac{d\mathbf{v}}{dt}=-\nabla p+\rho\mathbf{g}+\mu\nabla^2\mathbf{v} \tag{2.11}$$

There are three forces acting on each particle, and the sum of these forces $\mathbf{F}=-\nabla p+\rho\mathbf{g}+\mu\nabla^2\mathbf{v}$ (pressure force, external force, and viscous force) determines the change of momentum of the particle. For the time rate of change of velocity (*i.e.* acceleration) of particle *i*, we have

$$\frac{d\mathbf{v_i}}{dt}=\frac{\mathbf{F_i}}{\rho_i} \tag{2.12}$$

We will add wind force on external force term in next section.

2.2.1 Density

The density at position of particle *i*, can be computed from the SPH summation (2.5). We denote the position of particle *i* as $\mathbf{r_i}$, then its density is given by

$$\begin{aligned}\rho_i = \rho(\mathbf{r_i}) &= \sum_j m_j \frac{\rho_j}{\rho_j} W(\mathbf{r_i} - \mathbf{r_j}, h) \\ &= \sum_j m_j W(\mathbf{r_i} - \mathbf{r_j}, h)\end{aligned} \tag{2.13}$$

From equation (2.13), the density can be obtained by the weighed sum of the neighboring particle's mass, within the radius range *h*.

2.2.2 Pressure

The pressure term $-\nabla p$ can be described with the SPH gradient (2.6), so that we can obtain the pressure force acting on the particle *i* :

$$\mathbf{F_i^{pressure}} = -\nabla p(\mathbf{r_i}) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r_i} - \mathbf{r_j}, h) \tag{2.14}$$

Since particles only carry the information of mass, position, and velocity, the pressure at positions of particles has to be evaluated first. We use equation (2.13) to yield the density at those positions. Then the pressure p_i can be computed with the modified ideal gas state equation [43] :

$$p_i = k(\rho_i - \rho_0) \tag{2.15}$$

where ρ_0 is the rest density of the fluid, and *k* is a gas constant which depends on the temperature.

The force is not symmetric in (2.14), as can be seen when only two particles interact. The pressure at locations of the two particles is not equal and cause action

force and reaction force are not equal. This is not consistent with Newton's 3rd Law, namely law of reciprocal actions. However, we observe it can be symmetrized as [21] describes:

$$\mathbf{F}_{\mathbf{i}}^{\text{pressure}} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W\left(\mathbf{r}_{\mathbf{i}} - \mathbf{r}_{\mathbf{j}}, h\right) \tag{2.16}$$

2.2.3 Viscosity

The viscosity term $\mu\nabla^2 \boldsymbol{v}$ can be described with the SPH Laplacian (2.7), so that we can obtain the viscous force acting on particle *i* :

$$\mathbf{F}_{\mathbf{i}}^{\text{viscosity}} = \mu\nabla^2 \mathbf{v}\left(\mathbf{r}_{\mathbf{i}}\right) = \mu\sum_j m_j \frac{\mathbf{v}_{\mathbf{j}}}{\rho_j} \nabla^2 W\left(\mathbf{r}_{\mathbf{i}} - \mathbf{r}_{\mathbf{j}}, h\right) \tag{2.17}$$

Likewise, the force is also asymmetric. When only two particles interact with each other, the viscosity forces at locations of the two particles are not equal. This is because velocities of the two particles are typically not the same. On the other hand, since viscosity forces are only dependent on velocity difference and not on absolute velocity, the natural way to symmetrize the viscous force is using velocity difference, *i.e.*

$$\mathbf{F}_{\mathbf{i}}^{\text{viscosity}} = \mu\sum_j m_j \frac{\mathbf{v}_{\mathbf{j}} - \mathbf{v}_{\mathbf{i}}}{\rho_j} \nabla^2 W\left(\mathbf{r}_{\mathbf{i}} - \mathbf{r}_{\mathbf{j}}, h\right). \tag{2.16}$$

2.3 Wind-Driven Waves

Constructing realistic ocean waves is an important stage in our work, so in the beginning, we find out how the ocean waves are generated in the nature. Hydrodynamics of ocean waves has been researched and analyzed for a long time, and this research belongs to physical oceanography science [32-33]. The factors which are assumed to influence waves, in addition to wind, are fetch (*f,* distance to the upwind coastlines), duration (*t*, no wind or waves for $t<0$) and gravitational acceleration (g). Other parameters that may be relevant, such as the viscosity of water, turbulence in the airflow, gustiness and atmospheric stability, are usually ignored. These introduce errors in estimating the significant wave height of up to 20% even in such idealized situations. In our simulation, we consider wind forces, duration, gravity, viscosity and turbulence, and these factors influence the motion of waves.

From the hint of nature, we also attempt to "construct" our wind force blowing on particle-based fluid, just as it does in the real world. In the ideal situation of waves generation described above, the wind is assumed to be constant. The wind is a result of forces acting on the atmosphere [44]:

$$\mathbf{F}_{\text{net}} = \mathbf{F}_{\text{pg}} + \mathbf{F}_{\text{g}} + \mathbf{F}_{\text{co}} + \mathbf{F}_{\text{fr}} + \mathbf{F}_{\text{ce}}. \tag{2.19}$$

$\mathbf{F}_{\text{pg}}$ is pressure gradient force (PGF), which causes horizontal pressure differences and winds. $\mathbf{F}_{\text{g}}$ is gravity, which causes vertical pressure differences and

winds. $\mathbf{F_{co}}$ is coriolis force, which causes all moving objects, such as air, to diverge, or veer, to the right in the Northern Hemisphere and to the left in the Southern Hemisphere. $\mathbf{F_{fr}}$ is friction, which has very little effect on air high in the atmosphere, but more important closer to the ground. $\mathbf{F_{ce}}$ is centrifugal force, which describes objects in motion tend to travel in straight lines, unless acted upon by an external force. Here we can ignore coriolis forcen and centrifugal force, since the velocity of wind is too slow and the effect is not apparent in the simulation.

The wind can be defined as a three-dimensional vector that varies randomly in three space dimensions and time mathematically [32]. However, this vector is usually described by only horizontal component, averaged over some time interval (typically 10min) at a fixed elevation (typically 10m) above the mean sea surface, and the wind is accordingly denoted as $\vec{U}_{10} = \vec{U}_{10}(x, y, t)$. Sometimes the wind is characterized by an alternative, purely fictitious wind speed, which is directly related to the shear stress τ of wind. It is called the friction velocity and it is denoted by u_*. The relationship is

$$\tau = \rho_{air} u_*^2 = \rho_{air} C_d U_{10}^2 \tag{2.20}$$

where ρ_{air} is the density of air and C_d is the drag coefficient. The value C_d is determined with an expression:

$$C_d = \begin{cases} 1.2875 \times 10^{-3} & \text{for } U_{10} < 7.5\text{m/s} \\ (0.8 + 0.065 U_{10}) \times 10^{-3} & \text{for } U_{10} \geq 7.5\text{m/s} \end{cases} \tag{2.21}$$

In fluid dynamics, drag force refers to force that opposes the relative motion of an object through a fluid (a liquid or gas). Drag force acts in a direction opposite to the oncoming flow velocity. The drag equation calculates the force experienced by an object moving through a fluid at relatively large velocity and can be represented as below:

$$\mathbf{F_d} = \tau \cdot A = -\frac{1}{2}\rho \mathbf{v}^2 A C_d \hat{\mathbf{v}} \tag{2.22}$$

where ρ is the density of air, $\mathbf{v}$ is the speed of the object relative to the fluid, $\hat{\mathbf{v}}$ is the normalized relative velocity, A is the projection area, C_d is the drag coefficient. Drag force can be considered to the resistant force of object moving through wind, and it is parallel to the object's motion.

On the other hand, Lift is defined to be the component of the force that is perpendicular to the oncoming flow direction, namely, it contrasts with the drag force. If the lift coefficient for a object at a specified angle of attack is known (or estimated using a method such as thin-airfoil theory), then the lift produced for specific flow conditions can be determined using the following equation:

$$\mathbf{F_L} = -\frac{1}{2}\rho \mathbf{v}^2 A C_L \tag{2.23}$$

where C_L is the lift coefficient. Lift force is related to flow direction and is not necessarily opposed to gravity. Moreover, in order to simulate turbulence in the airflow, we also add random term on wind forces, which is noise of wind force:

$$\mathbf{F}_{\mathrm{N}} = n_x\mathbf{i} + n_y\mathbf{j} + n_z\mathbf{k} \tag{2.24}$$

where n_x, n_y, n_z are three random variables generated from uniform distribution $U(\min,\ \max)$. Accumulating these forces is our method to model wind forces. In our developing tools, there is a concept called force field. While objects in the range of field, they are influenced with external forces which are determined in the force field. We use force field to model the friction forces of wind. With varying directions and magnitudes of $\mathbf{v}$, we can control our wind velocity to do simulation and our force field will update its position according this wind velocity. Since the velocity is much larger than velocities of objects, we can regard it as relative velocity. Once we decide our wind velocity, we can estimate external forces by the equation (2.22) and (2.23). We can defined a vector is horizontal and perpendicular to the wind velocity vector, and let wind velocity vector cross to this vector to get the direction of lift force. Finally we add random term in our force field to model the friction forces of wind, and we can also determine the influence range of the wind force.

Applying our external wind force on the particle-based fluid to generate waves is our main approach. It also means we add the wind force term to the right side of the equation (2.11) and solve the velocity of fluid. Then, for each fluid particle, we have

$$\rho\frac{d\mathbf{v}}{dt} = -\nabla p + \rho\mathbf{g} + \rho\frac{\mathbf{F}_{\mathrm{w}}}{m} + \mu\nabla^2\mathbf{v} \tag{2.25}$$

and $\mathbf{F}_{\mathrm{w}}$ is wind force:

$$\begin{aligned}\mathbf{F_w} &= \mathbf{F_d} + \mathbf{F_L} + \mathbf{F_n} \\ &= -\frac{1}{2}\rho \mathbf{v}^2 A C_d \hat{\mathbf{v}} - \frac{1}{2}\rho \mathbf{v}^2 A C_L + n_x \mathbf{i} + n_y \mathbf{j} + n_z \mathbf{k}\end{aligned} \tag{2.26}$$

with SPH method to determine the properties of fluid, we can rewrite equation (2.25) then, for each fluid particle:

$$\begin{aligned}\sum_j m_j W\left(\mathbf{r_i} - \mathbf{r_j}, h\right) \cdot \frac{d\mathbf{v_i}}{dt} = &-\sum_j m_j \frac{p_j}{\rho_j} \nabla W\left(\mathbf{r_i} - \mathbf{r_j}, h\right) + \rho_i \mathbf{g} \\ &+ \rho_i \frac{\mathbf{F_w}}{m_i} + \mu \sum_j m_j \frac{\mathbf{v_j} - \mathbf{v_i}}{\rho_j} \nabla^2 W\left(\mathbf{r_i} - \mathbf{r_j}, h\right)\end{aligned} \tag{2.27}$$

With this equation, we can solve velocity of each fluid particle. Once we get the velocity, we can update its position and observe motion of waves. Generally speaking, wind-generated gravity waves are considered as elliptic motions of the ocean surface, and occur in the oceans and along the shores of the world. There is little actual forward motion of individual water particles in a wave, despite the large amount of energy it may carry forward. When the wind blows, because of the pressure and the viscosity force of the wind, ocean surface is disturbed and some energy transfer from the wind to the waves. Due to the combination of transverse wave and longitudinal wave, water particles of the ocean surface have elliptic motions.

To estimate wave conditions visually, observer tends to concentrate his attention on the highest waves in the wave field. These average wave characteristics are called the significant wave height and the significant wave period, denoted as H_s and T_s, respectively. However, two wave parameters give only a limited description of the

wave conditions and more parameters are required for detailed descriptions. This is sometimes done and it may be appropriate in some cases, but any small number of parameters would not, in general, completely characterize the wave conditions. For a complete description in a statistical sense, the spectral technique is required. It is based on the notion that the random motion of the ocean surface can be treated as the summation of a large number of harmonic wave components.

Chapter 3
Ocean Surface Construction and Rendering

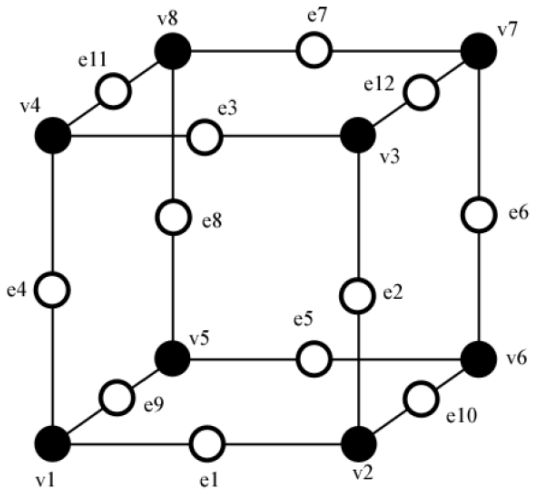
For interactive applications, particle-based fluid is commonly preferred to grid-based fluid representation. This is because the fluid is able to flow everywhere in the scene without the need to define finite grids, which have expensive cost of memory and computation. Particle-based fluid is also more convenient to integrate into physics systems as particles can collide against the scene geometry such as other rigid objects, without the need to voxelize the scene geometry. The drawback is that it is difficult to extract a surface for rendering.

Generally, the fluid surface is constructed in world-space, either directly as a mesh [45], or as implicit surface and then polygonized using marching cubes [22] or similar methods [46, 47]. Level-sets [48] have been used extensively in fluid simulations to track the interface between fluid and air as they provide a deformable surface representation that allows for topology changes. But even efficient level-set methods involve too much computation to be used in real-time. In [49] the authors present an approach for generating the boundary of a three-dimensional point cloud as a mesh in screen-space, generating the surface only where it is visible. It first

computes the depth to the surface at each pixel on the screen, smoothes this depth map with using a binomial filter, then polygonizes the depth buffer. This thesis uses marching cubes algorithm and a new method to render fluid, and in next section we introduce these two methods.

3.1 Marching Cubes

The marching cubes method [22] is an algorithm of surface construction used for viewing 3D data, and it is also a popular technique for generating triangle meshes along isosurfaces of scalar fields. The basic notion is that we can define a voxel(cube) by the pixel values at the eight corners of the cube. If one or more pixels of a cube have values less than the user-specified isovalue, and one or more have values greater than this value, then we know the voxel must contribute some component of the isosurface. By determining which edges of the cube are intersected by the isosurface, we can create triangular patches which divide the cube between regions within the isosurface and regions outside. By connecting the patches from all cubes on the isosurface boundary, we get a surface representation. This algorithm is originally used to extract the surface of medical organs. It provides a way to get from serial sections to a complete 3D object. One important application is the reconstruction of the liquid-air interface in fluid simulations.



Case = v8|v7|v6|v5|v4|v3|v2|v1

Figure 3-1: Cube created from eight pixels.
(http://users.polytech.unice.fr/~lingrand/MarchingCubes/algo.html)

The marching cubes algorithm consists of three basic steps. First, locate the surface corresponding to a user-specified value. Second, create triangles. Last, calculate normal vectors to the surface at each vertex. To locate the surface, it uses a cube created from eight (data) pixels. The indexing convention for vertices and edges used in the algorithm are shown as Figure 3-1. The algorithm determines how the surface intersects this cube, then moves (or marches) to the next cube. To find the surface intersection in a cube, we assign a one to a cube's vertex if the data value at that vertex exceeds (or equals) the value of the surface we are constructing. These vertices are inside (or on) the surface. Cube vertices with values below the surface receive a zero and are outside the surface. The surface intersects those cube edges where one vertex is outside the surface (one) and the other is inside the surface (zero).

With this assumption, it can determine the topology of the surface within a cube, finding the location of intersection.

Since there are eight vertices in each cube and two states, inside and outside, surface can intersect the cube in $2^8 = 256$ ways. By enumerating these 256 cases, we can create a table to look up surface-edge intersections, given the labeling of cube vertices. The table contains the edges intersected for each case. Two different symmetries of the cube reduce the problem from 256 cases to 14 patterns. First, the topology of the triangulated surface is unchanged if the relationship of the surface values to the cubes is reversed. Complementary cases, where vertices greater than the surface value are interchanged with those less than the value, are equivalent. Thus, only cases with zero to four vertices greater than the surface value need be considered, reducing the number of cases to 128. Using the second symmetry property, rotational symmetry, we reduced the problem to 15 patterns by inspection. (Figure 3-2.)

We create an index for each case, based on the state of the vertex. The 8-bit index contains one bit for each vertex. This index serves as a pointer into an edge table that gives all edge intersections for a given cube configuration. Using the index to tell which edge the surface intersects, we can interpolate the surface intersection along the edge. The position that it cuts the edge will be linearly interpolated, and the

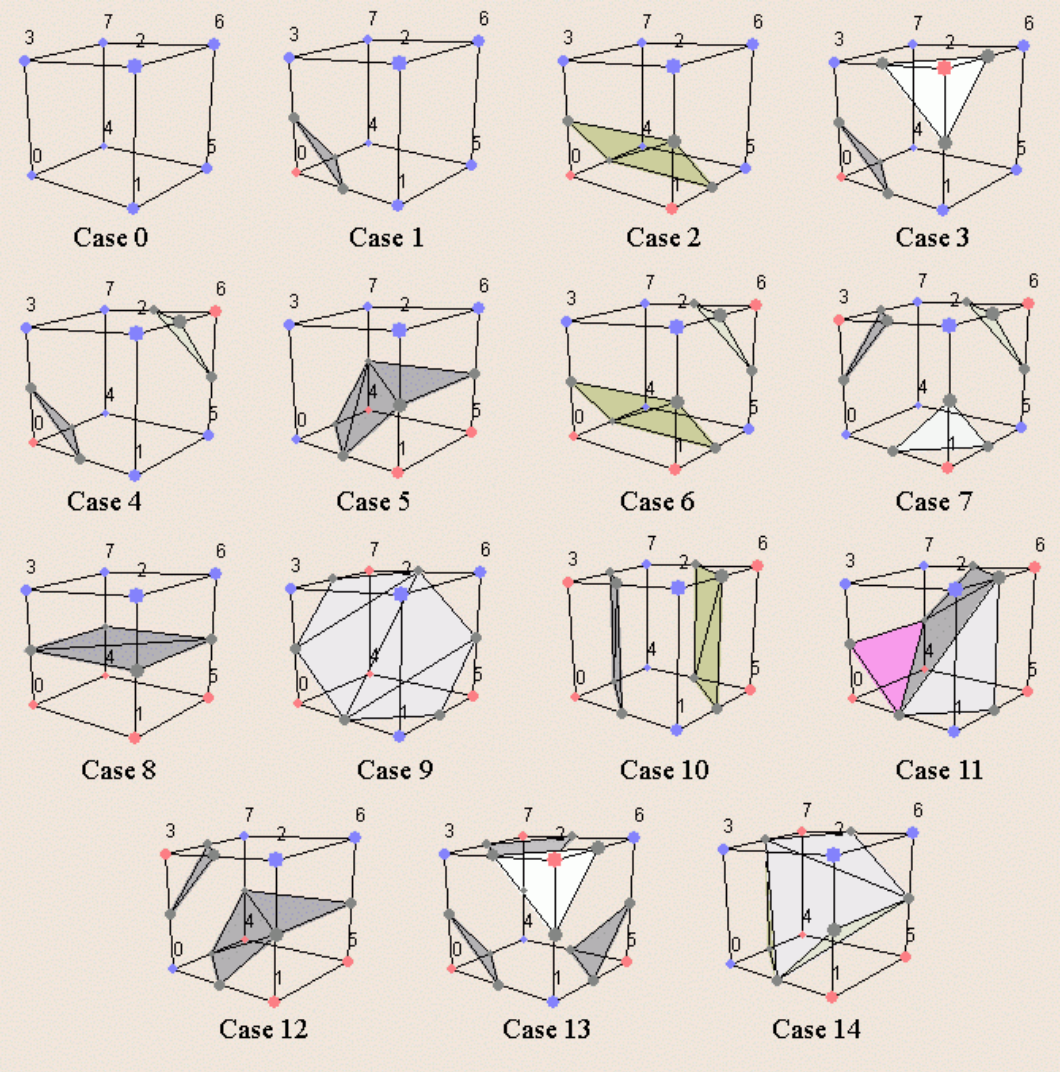


Figure3-2: The triangulation for the 15 patterns.

(http://users.polytech.unice.fr/~lingrand/MarchingCubes/algo.html)

ratio of the length between the two vertices will be the same as the ratio of the isosurface value to the values at the vertices of the cube.

The final step in marching cubes calculates a unit normal vector for each triangle vertex. The rendering algorithms use this normal to produce images. A surface of constant density has a zero gradient component along the surface tangential direction. Consequently, the direction of the gradient vector is normal to the surface. We can use

this fact to determine surface normal vector, $\vec{\mathbf{n}}$ if the magnitude of the gradient, $\left|\vec{G}\right|$, is nonzero. The gradient vector, $\vec{G}$, is the derivative of the density function.

$$\vec{G}(x, y, z) = \nabla \vec{D}(x, y, z) \tag{3.1}$$

To estimate the gradient vector at the surface of interest, we first estimate the gradient vectors at the cube vertices and linearly interpolate the gradient at the point of intersection. The gradient at cube vertex *(i, j, k)*, is estimated using central differences along the three coordinate axes by:

$$G_x(i, j, k) = \frac{\vec{D}(i+1, j, k) - \vec{D}(i-1, j, k)}{\Delta x} \tag{3.2}$$

$$G_y(i, j, k) = \frac{\vec{D}(i, j+1, k) - \vec{D}(i, j-1, k)}{\Delta y} \tag{3.3}$$

$$G_z(i, j, k) = \frac{\vec{D}(i, j, k+1) - \vec{D}(i, j, k-1)}{\Delta z} \tag{3.4}$$

3.2 Fitting with Cloth

Although Marching Cubes is certainly the popular and useful algorithm to generate 3D triangle meshes for isosurfaces of scalar fields, there are some drawbacks when used for the visualization of surfaces of liquids. First, since the standard approach is camera-independent, many invisible triangles and surface details are generated. Second, the algorithm operates in three dimensions although the front 2D surface is sought. This surface is to be found by marching through a 3D data set, and the cost of calculation is relative expensive. In order to achieve higher frame rate, we

present a new method to construct the fluid surface. Since our goal is rendering a surface mesh which fits top fluid particles, we attempt to use physically simulated cloth (soft body) to do this task. We will introduce physical model for cloth in following paragraph.

Physical cloth animation has been a problem to the graphics community for more than a decade. Early work by Terzopoulos et al. [50], Terzopoulos and Fleischer [51, 52] on deformable models correctly characterized cloth simulation as a problem in deformable surfaces, and applied techniques from the mechanical engineering and finite element communities to the problem. Since then, other research groups have taken up the challenge of cloth. One of the models achieves real-time performances by using the mass-spring system [53, 54, 55]. This model can be seen as a discrete approximation of a finite-element method for integrating the Lagrange partial derivative equation of motion [56].

A continuous cloth surface is discretized into a finite number of particles, and similar to a sphere is divided into a group of vertices and triangles for drawing. The particles are then connected in an orderly fashion with springs (Figure 3-3). Each particle is connected with springs to its four neighbors along both the horizontal and vertical axes. These springs are called stretch springs because they prevent the cloth from stretching too much. Additional springs are added from each particle to its four

neighbors along the diagonal directions. These shear springs resist any shearing

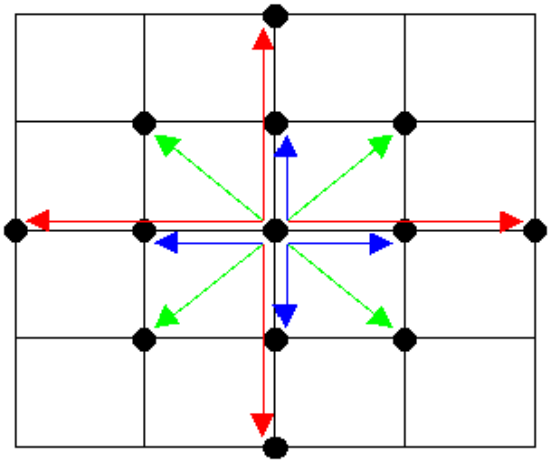


Figure 3-3 Stretch, Shear, and Bend springs
(http://www.cppblog.com/zmj/archive/2009/01/13/71893.html)

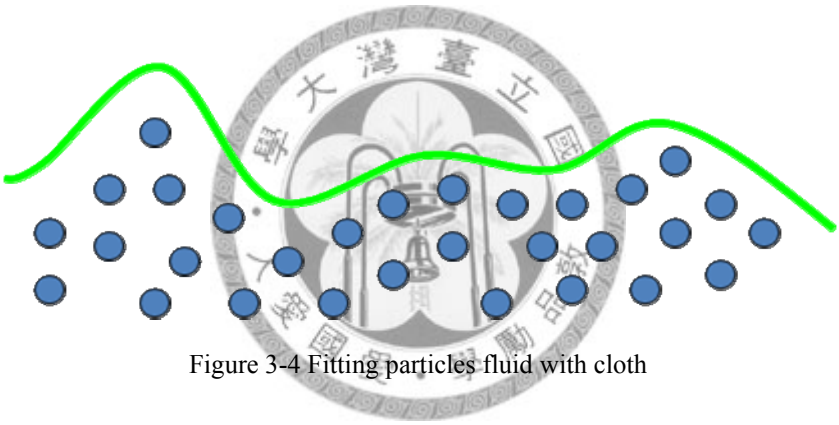


Figure 3-4 Fitting particles fluid with cloth

movement of the cloth. Finally, each spring is connected to the four neighbors along both the horizontal and vertical axes but skipping over the closest particles. These springs are called bend springs and prevent the cloth from folding itself too easily.

Now we understand the cloth model, the basic idea is using the cloth as a surface to fit particles fluid. So we put the cloth on fluid (Figure 3-4), and let particles fluid collide with cloth particles. Notice that we want momentum can be transfered from fluid to cloth, but not vice versa. Since we do not want to alter the motion of fluid, in

other words, it is one-way interaction between cloth and fluid. We have to modify several parameters else, such as bending stiffness and stretching stiffness, to affect the strength of the constraint along horizontal and vertical edges. We also decrease the gravity effect on cloth and attach particles of four verices to fixed points. Finally, we set friction and how large the damping is applied to the motion of cloth particles.

3.3 Ocean Surface Rendering

As the physical simulation loop is completed, if we do not render it properly, the user would not be immersive into the virtual reality system. In this section, we would introduce how we render the ocean surface.

The rendering equation presents a general algorithm for computer graphics [57]. It presents that the light intensity is simply the sum of the light that an object emits and other scattered light from other objects, *i.e.*,

$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_{\Omega} \rho(x,x',x'')\, I(x',x'')\, dx''\right] \tag{3.5}$$

where

$I(x,x')$ is related to the intensity of light (radiance) passing from point x' to x,

$g(x,x')$ is a geometry term, equals to $1/r^2$ if x' is visible from x, else 0,

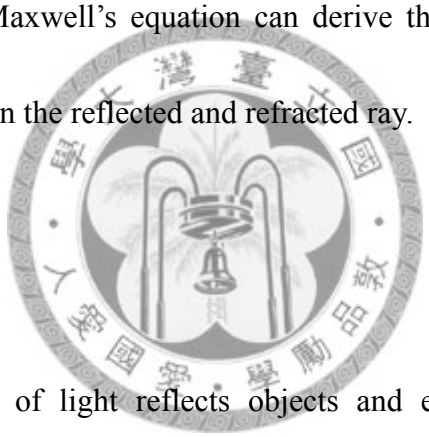
$\varepsilon(x,x')$ is related to the intensity of emitted light from x' to x, and

$\rho(x,x',x'')$ is related to the intensity of light scattered from x'' to x by a patch of surface at x'.

The rendering equation gives a base of rendering. Computer graphics progresses unceasingly as time evolves, and more complicate and realistic methods are brought up to the world. Therefore, realistic ocean simulation is implemented with the simplified and faster method described below.

3.3.1 Reflection and Refraction

The main visual phenomenon of the water is reflection and refraction. As an incident ray hitting water surface, part of the ray reflect to air, and other part transmit into water. Solving the Maxwell's equation can derive the Fresnel equation which calculates the ratio between the reflected and refracted ray.



Reflection

The main reflection of light reflects objects and environment above ocean surface. Factors which influence the reflection of light are not only related to the radiance but also the shape and material of object's surface. In Figure 3-5, the reflection part of light can be calculated with

$$\vec{\mathbf{R}} = 2(\vec{\mathbf{E}} \cdot \vec{\mathbf{N}})\vec{\mathbf{N}} - \vec{\mathbf{E}} \tag{3.6}$$

where $\vec{\mathbf{E}}$ is the eye vector from object surface *O* to eye, $\vec{\mathbf{N}}$ is the object surface normal vector, and $\vec{\mathbf{R}}$ is the original incident ray. The law of reflection states that when a ray hit on a surface, the angle of incidence is equal to the angle of reflection.

With this equation, we can get which part of ray emits from the background environment and reflects into the camera. Once we obtain the vector $\vec{\mathbf{R}}$, then we can know which part of environment scenes would be reflected.

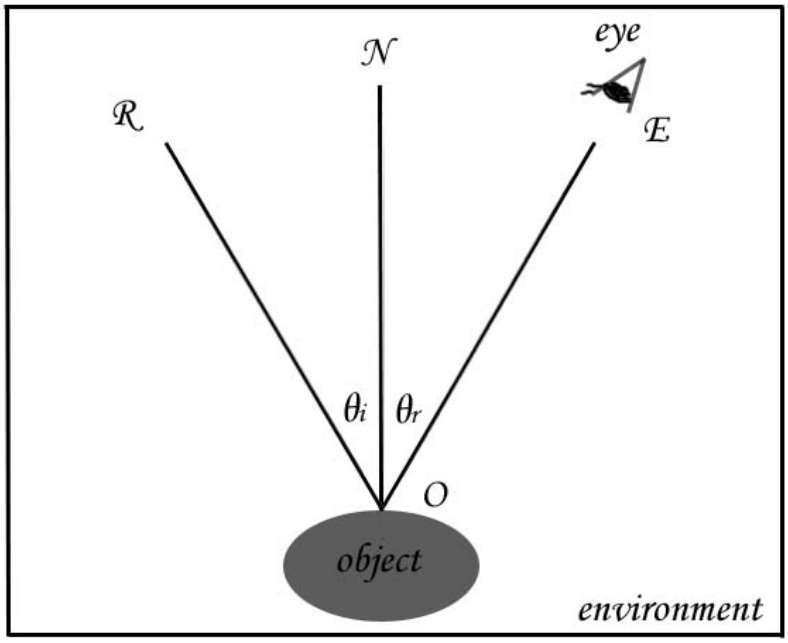


Figure 3-5: The reflection law diagram

Refraction

Refraction of light transmitting the scene and object below the water is also an important phenomenon of fluid. It dues to waves change their speed in different mediums, and change their direction. When refraction occurs, it would obey the Snell's law:

$$n_1 \sin\theta_i = n_2 \sin\theta_t \tag{3.7}$$

where θ_i is the incidence angle (*i.e.* angle between the surface normal and the incident ray), θ_t is the transmitted angle (*i.e.* angle between the transmitted ray and the negated normal), and n_1 and n_2 are refractive indices of incident and refraction mediums respectively. The refractive index of the air is near to 1, and the refractive index of the water is 1.333.

However we assume the ocean is very deep, and we could not see objects and terrain through the ocean. The only color we can see from refraction is color of ocean.

3.3.2 The Fresnel Effect

When light reaches to the interface between two materials, part of it reflects and the other refracts through the surface (Figure 3-6). The effect which describes the ratio between the reflection and refraction is called Fresnel effect. Figure 3-7 is the relationship of the incident and transmitted angle. The limit of the transmitted angle is 48.59 degree, as the incident angle here is 90 degree and the light is parallel to the water surface. In other words, the object in the water could not be seen if it is outside this range. This is the reason that when we are standing on a side of a clear lake, we can see the objects below the water surface close to our eyes. However, it is hard to see through the water when we are looking toward the horizon. Since the water surface acts as a mirror around the horizon.

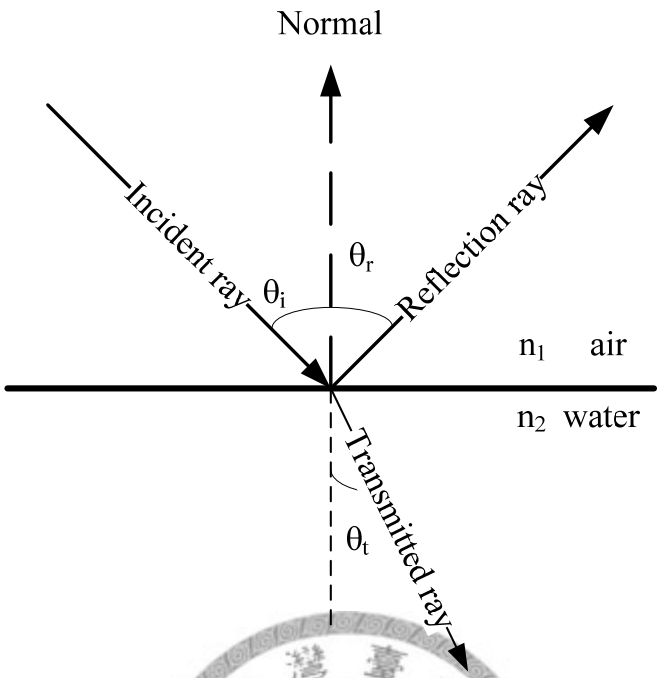


Figure 3-6: The relationship between rays path and their angles in two different mediums.

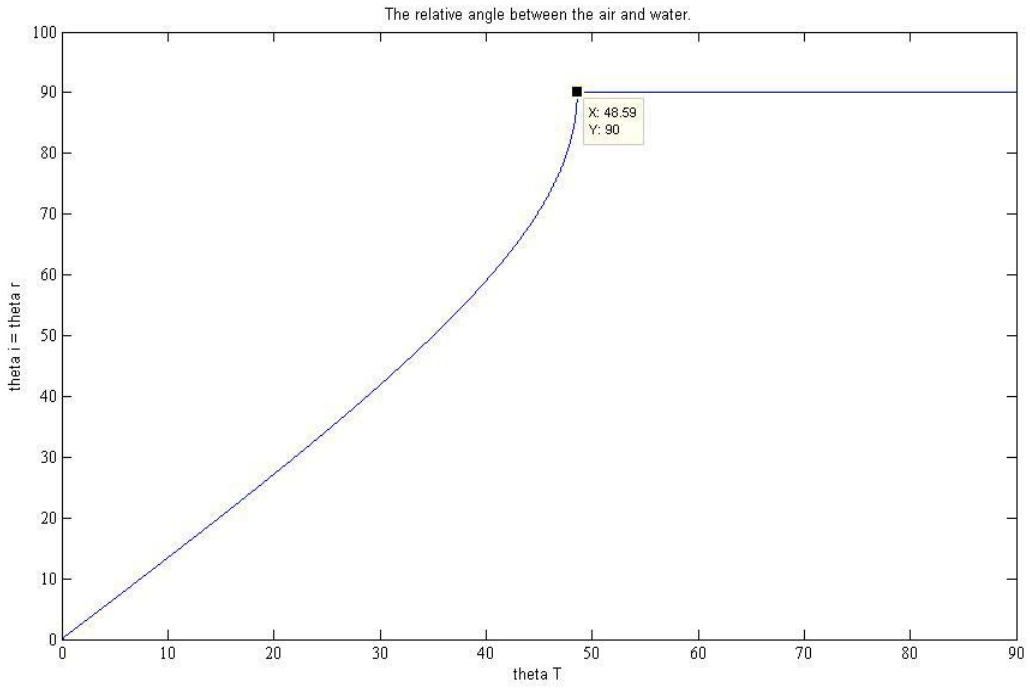


Figure 3-7: The angles relationship between the incident angle (reflected angle) and transmitted angle.

The Fresnel equation determines the reflection coefficient, which is the proportion of the reflection light. We assume the light is non-polarized here. The reflection coefficient is

$$R_c = \frac{1}{2}\left(\left(\frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)}\right)^2 + \left(\frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)}\right)^2\right) \tag{3.8}$$

where the incident angle θ_i can be calculated with the eye vector $\vec{\mathbf{E}}$ and surface normal $\vec{\mathbf{N}}$, we can obtain θ_t from the Snell's Law. θ_r is equals to θ_i, therefore we can derive

$$\cos\theta_i = \cos\theta_r = \frac{\vec{\mathbf{E}} \cdot \vec{\mathbf{N}}}{\left|\vec{\mathbf{E}}\right|\left|\vec{\mathbf{N}}\right|} \tag{3.9}$$

The transmission coefficient T_c (the proportion of the refraction light transmits across the interface to the new medium) can be derived by:

$$T_c = 1 - R_c \tag{3.10}$$

3.3.3 Shading for Ocean

A "Shader" is a set of software instructions in the field of computer graphics, which is used primarily to calculate rendering effects on graphics hardware with a high degree of flexibility. It allows programmers to program GPU (Graphic Processing Unit) programmable rendering pipeline. It is more flexibility than the older "fixed-function pipeline". We combine OpenGL (Open Graphics Library) and Cg (C

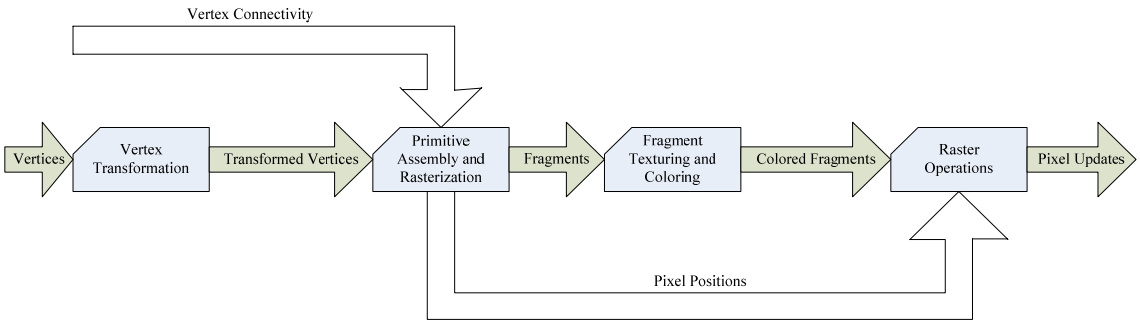


Figure 3-8: The graphics hardware pipeline

for Graphics) which is a high-level shading language (HLSL) developed by NVIDIA, for programming vertex and pixel shaders to render our result.

Figure 3-8 shows the pipeline of the graphics hardware. OpenGL (3D API) sends a sequence of vertices which have some other attributes such as normal, color, texture coordinate and other parameters to GPU. Then, entering the first processing stage called vertex transformation. It performs mathematic operations on each vertex to the windows spaces. The second stage is called primitive assembly and rasterization. Those vertices are assembled into specific geometric primitives. They would require clipping to the view frustum (the visible region in the 3D space) and culling (discard the polygons that are backward to us). After those primitives are rasterized, it determines which fragment is covered by a geometric primitive, and pixels are decided. The third stage, interpolation, texturing and coloring stage determines final colors, texture look up and mapping. The fourth stage, raster operation stage, does

several tests such as scissor, alpha, stencil and depth test. Finally, the frame buffer updates the pixel's color with the blended color.

After introducing the GPU pipeline, we would introduce the pixel color we render. In general, the color is the sum of emissive, ambient, diffuse and specular colors [35].

$$surfaceColor = emissive + ambient + diffuse + specular \tag{3.11}$$

The emissive term is the surface emits light itself. It is independent of the other light.

$$emissive = K_e \tag{3.12}$$

where K_e is the material's emissive color. The ambient term means the light exists everywhere and is independent of position.

$$ambient = K_a \cdot globalAmbient \tag{3.13}$$

where K_a is the material's ambient reflectance, and *globalAmbient* is the ambient light color. The diffuse term accounts for reflected light.

$$diffuse = K_d \cdot lightColor \cdot \max(N \cdot L, 0) \tag{3.14}$$

where K_d is the material's diffuse coefficient, *lightColor* is the color of the diffuse light, N is the unit surface normal vector and L is the normalized vector from surface toward the light source.

The specular term is the light scattered from a surface. It is used on the smooth and shiny surfaces, such as the metal. The specular term is dependent of the viewer's position.

$$specular = K_s \cdot lightColor \cdot facing \cdot \max(\vec{N} \cdot \vec{H}, 0)^{shininess} \quad (3.15)$$

Where K_s is the material's specular coefficient, *lightColor* is the color of the specular light, N is the unit surface normal vector, L is the normalized vector toward the light source, H is the normalized vector that is halfway between vector from surface toward the viewer and L, *facing* is 1 if $\vec{N} \cdot \vec{L} > 0$, and 0 otherwise. *shininess* is the factor which determines the range of the shining part.

We consider environment map of skybox and water color for diffuse and emissive parts, and use Fresnel effect to determine the ratio. However, the sunshine reflection does not look very clear from the surface. The image of the sunshine in the environment is too bright to be mapped. The imitative high-dynamic-range (HDR) is used here to render the shining reflection of sunshine on the water:

$$Sun_{reflectC} = Sun_{reflectC}\left(1 + L_f\left(\mathrm{C}_{\mathrm{curr}} - \mathrm{C}_{\mathrm{threshold}}\right)\right) \quad (3.16)$$

where $Sun_{reflectC}$ is output color, $\mathrm{C}_{\mathrm{curr}}$ is the current pixel color we see, and $\mathrm{C}_{\mathrm{threshold}}$ is the threshold color. If the current pixel color exceeds this value, it will be multiplied by L_f. This formula can create the better result for sunshine reflection.

Chapter 4
Ship Modeling and Dynamics

4.1 Preliminary Rigid Body Dynamics

We will describe the rigid body's modeling and dynamics in this chapter. We start from introducing the rigid body theory of classical mechanics, which relates to the position, velocity and acceleration.

4.1.1 Position and Orientation

An unconstrained rigid body in the world coordinate system can be expressed in six degrees of freedom. We usually divide it into two main groups: three degrees represent for translational motions and others represent for rotational motions. Therefore, these two groups of motions are described as translation and rotation.

Position represents where a point is in the space. Typically, it considers a rigid body as a point, which is at the center of mass of the rigid body. This point is also the origin of the body coordinate system position. We can use position vector to describe the relationship between the point and the origin of the world coordinate system, as shown in Figure 4-1. Let $\mathbf{x}$ denotes the position vector $[p_x, p_y, p_z]^T$ in R^3.

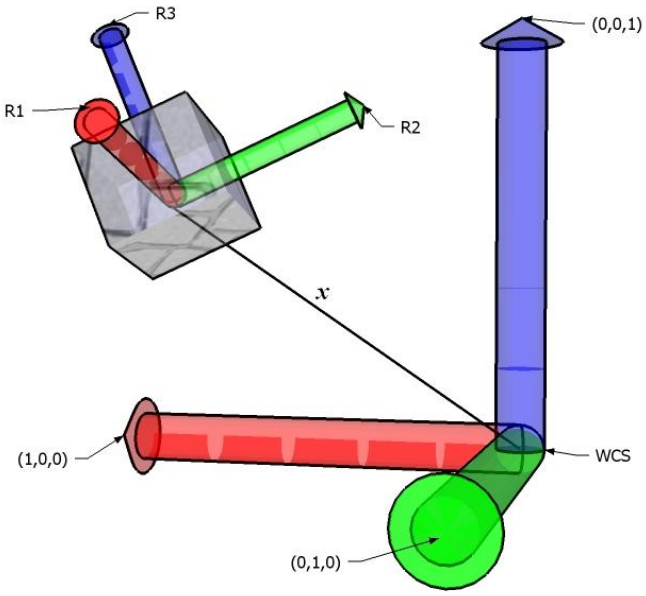


Figure 4-1 Represent the position and orientation of rigid body in world coordinate system.

When an object translates or rotates, its position vector would be transformed and could be described as:

$$T(\mathbf{x}) = \mathbf{A}\mathbf{x}. \tag{4.1}$$

where $T(\cdot)$ stands for translation or rotation, and $\mathbf{A}$ denotes the corresponding matrix. Translation is the process that changes the object's position. We use a 4×1 vector to represent position. If we want to translate the position, it is equivalent to multiply it by a translation matrix as below:

$$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.2}$$

where tx, ty, tz are the translation amount along x-axis, y-axis and z-axis respectively.

Orientation represents a rigid body's posture in a space, and we can rotate the rigid body to change its orientation. There are two methods describing rotation, one is using Euler angles, and the other is using quaternion. We introduce both of them here.

In Euler angles method, it uses a 3×3 matrix to represent rotation. The fundamental rotation matrices are:

$$Q_x(\theta)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix}$$
$$Q_y(\theta)=\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}. \qquad (4.3)$$
$$Q_z(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

where Q_x, Q_y, Q_z represent rotation around x-axis, y-axis, z-axis respectively. If we want to rotate a object around an arbitrary axis $\mathbf{u}$, then the rotation matrix $Q_u(\theta)$ subject to angular displacement θ is found to be:

$$\begin{aligned}Q_u(\theta)&=\begin{bmatrix}0 & -z & y\\ z & 0 & -x\\ -y & x & 0\end{bmatrix}\sin\theta+(I-\mathbf{u}\mathbf{u}^T)\cos\theta+\mathbf{u}\mathbf{u}^T\\ &=\begin{bmatrix}(1-x^2)c_\theta+x^2 & -zs_\theta-xyc_\theta+xy & ys_\theta-xzc_\theta+xz\\ zs_\theta-xyc_\theta+xy & (1-y^2)c_\theta+y^2 & -xs_\theta-yzc_\theta+yz\\ -ys_\theta-xzc_\theta+xz & xs_\theta-yzc_\theta+yz & (1-z^2)c_\theta+z^2\end{bmatrix}. \qquad (4.4)\\ &=\begin{bmatrix}x^2(1-c_\theta)+c_\theta & xy(1-c_\theta)-zs_\theta & xz(1-c_\theta)+ys_\theta\\ xy(1-c_\theta)+zs_\theta & y^2(1-c_\theta)+c_\theta & yz(1-c_\theta)-xs_\theta\\ xz(1-c_\theta)-ys_\theta & yz(1-c_\theta)+xs_\theta & z^2(1-c_\theta)+c_\theta\end{bmatrix}\end{aligned}$$

where s_θ is $\sin\theta$, c_θ is $\cos\theta$. With homogeneous coordinate, $Q_u(\theta)$ would be:

$$Q_u(\theta)=\begin{bmatrix} x^2(1-c_\theta)+c_\theta & xy(1-c_\theta)-zs_\theta & xz(1-c_\theta)+ys_\theta & 0 \\ xy(1-c_\theta)+zs_\theta & y^2(1-c_\theta)+c_\theta & yz(1-c_\theta)-xs_\theta & 0 \\ xz(1-c_\theta)-ys_\theta & yz(1-c_\theta)+xs_\theta & z^2(1-c_\theta)+c_\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

In the following, we introduce quaternion method. A quaternion can specify the orientation of a rigid body. It carries the information of the rigid body rotate about any axis by any angle. We denote a quaternion $\boldsymbol{q}=\{a,\mathbf{b}\}$, where a is scalar part, and $\mathbf{b}$ is vector part. Thus a quaternion can be essentially seen as a 4×1 vector. When we want to represent an orientation state of the rigid body which rotates about the axis $\mathbf{v}$ by θ, the quaternion is :

$$\boldsymbol{q}(\theta,\mathbf{v})=\left\{\cos\frac{\theta}{2},\mathbf{v}\sin\frac{\theta}{2}\right\} \tag{4.6}$$

Actually, quaternion is often preferable to rotation matrices or Euler angles. It avoids the singularity problems of the other methods, and offers some additional benefits such as providing an easy way to correctly interpolate between rotations [58]. Rotation is the quantity of the orientation change. Quaternions can be easily combined by multiplying two different quaternions together, and the representation is defined as follows:

$$\begin{aligned} \boldsymbol{q}_1\boldsymbol{q}_2 &= \{a_1,\mathbf{b}_1\}\{a_2,\mathbf{b}_2\} \\ &= \{a_1a_2-\mathbf{b}_1\cdot\mathbf{b}_2,\ a_1\mathbf{b}_2+a_2\mathbf{b}_1+\mathbf{b}_1\times\mathbf{b}_2\} \end{aligned} \tag{4.7}$$

The inverse of the quaternion is simply the rotation in reverse direction, *i.e.*

$$\boldsymbol{q}^{-1}=\{-a,\mathbf{b}\}=\{a,-\mathbf{b}\} \tag{4.8}$$

If a rotation matrix $\mathbf{R}$ specifies the same orientation as the quaternion $\boldsymbol{q}$ does, then the transformation of a vector $\mathbf{r}$ is:

$$\mathbf{Rr} = \boldsymbol{q}[\mathbf{r}] = unquat\left(\boldsymbol{q}quat(\mathbf{r})\boldsymbol{q}^{-1}\right) \tag{4.9}$$

where $quat(\mathbf{r}) = \{0, \mathbf{r}\}$, and $unquat(\{a, \mathbf{b}\}) = \mathbf{b}$. The conjugate scaling is defined as follow:

$$\begin{aligned} \boldsymbol{q}\mathbf{p}\boldsymbol{q}^{-1} &= \frac{1}{\|\boldsymbol{q}\|^2}\{a, \mathbf{b}\}\{t, \mathbf{u}\}\{a, -\mathbf{b}\} \\ &= \left\{ t, \frac{1}{\|\boldsymbol{q}\|^2}\left(\|\boldsymbol{q}\|^2 \mathbf{I} - 2\left(\mathbf{b}^{\mathrm{T}}\mathbf{b}\mathbf{I} - \mathbf{b}\mathbf{b}^{\mathrm{T}}\right) + 2a\hat{\mathbf{b}}\right)\mathbf{u} \right\} \end{aligned} \tag{4.10}$$

4.1.2 The Velocity of Rigid Body

Velocity of a rigid body is usually characterized by linear velocity $\boldsymbol{v}$ and angular velocity $\boldsymbol{\omega}$. We define a fixed point at our rigid body (at the center of mass). The displacement of the fixed point on the rigid body within a unit time step is the definition of linear velocity, *i.e.*,

$$\mathbf{v} = \frac{d\mathbf{x}}{dt} \tag{4.11}$$

where $\mathbf{x}$ is the position of the rigid body.

The angular velocity $\boldsymbol{\omega}$ is defined as the rate of rotating of the rigid body about a certain axis, namely,

$$\boldsymbol{\omega} = \frac{d\boldsymbol{\theta}}{dt} \tag{4.12}$$

where $\boldsymbol{\theta}$ is the angular position of the rigid body.

In quaternion method, the rate of change of $\boldsymbol{q}$ is given by

$$\dot{\boldsymbol{q}} = \frac{1}{2} quat(\boldsymbol{\omega})\boldsymbol{q} \tag{4.13}$$

Rearranging equation (4-13), we can obtain the angular velocity from $\dot{\boldsymbol{q}}$ as

$$\begin{aligned} 2\dot{\boldsymbol{q}}\boldsymbol{q}^{-1} &= quat(\boldsymbol{\omega}) \\ \boldsymbol{\omega} &= unquat(2\dot{\boldsymbol{q}}\boldsymbol{q}^{-1}) \end{aligned} \tag{4.14}$$

Suppose that there is a point p attached to a rigid body, and $\mathbf{r}$ is the vector pointing from the center of mass to p, then given the velocities of the rigid body $\mathbf{v}_{cm}$, we can compute the velocity at any point p, *i.e.*

$$\mathbf{v}_p = \mathbf{v}_{cm} + \boldsymbol{\omega} \times \mathbf{r} \tag{4.15}$$

4.1.3 Equations of Motion

With the Newton's second law, given external force $\mathbf{F}$ and torque $\boldsymbol{\tau}$ acting on a rigid body, we can determine the acceleration and angular acceleration of the rigid body. Then we can update its velocity and angular velocity.

$$\mathbf{F} = \frac{d}{dt}(m\mathbf{v})$$
$$\boldsymbol{\tau} = \frac{d}{dt}(\mathbf{I}\boldsymbol{\omega}) \tag{4.16}$$

From equation (4.16), the external force is the rate of change of the linear momentum, and the external torque is the rate of change of the angular momentum. The mass and moment of inertia of the rigid body can be obtained by integrating all of the particles of it, *i.e.*

$$m = \sum m_i$$
$$\mathbf{I} = \sum m_i(\mathbf{r}_i^T\mathbf{r}_i\mathbf{I} - \mathbf{r}_i\mathbf{r}_i^T) \tag{4.17}$$

where m_i is the mass of the i^{th} particle, and $\mathbf{r}_i$ is the vector from the center of mass to the i^{th} particle. In fact, the moment of inertia matrix is depends on the shape of the rigid body. Rotating about different axes would cause the different moments of inertia. The moment of inertia tensor is a convenient way to summarize all moments of inertia of an object with one quantity. For an object with N particles, the moment of inertia tensor is given by

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{xz} & I_{zy} & I_{zz} \end{bmatrix}$$
$$I_{ij} \equiv \sum_{k=1}^{N} m_k(r_k{}^2\delta_{ij} - r_{ki}r_{kj}) \tag{4.18}$$

where i, j are equal to 1, 2, or 3 for x, y, and z, respectively, r_k is the distance of mass k from the point to the center of mass, and $\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$ also known as Kronecker delta.

The matrix $\mathbf{I}$ is the angular inertia tensor considered in the world coordinate system. It can also be transferred from $\mathbf{I}_{body}$ as shown below:

$$\mathbf{I} = \mathbf{R}\mathbf{I}_{body}\mathbf{R}^T \tag{4.19}$$

where $\mathbf{I}_{body}$ is the inertial tensor specified in body coordinate system, and $\mathbf{R}$ is the rotation matrix of the rigid body. Finally we can get the changing rate of angular velocity by

$$\begin{aligned}
\boldsymbol{\tau} &= \frac{d}{dt}(\mathbf{I}\boldsymbol{\omega}) \\
&= \mathbf{I}\dot{\boldsymbol{\omega}} + \dot{\mathbf{I}}\boldsymbol{\omega} \\
&= \mathbf{I}\dot{\boldsymbol{\omega}} + \frac{d}{dt}(\mathbf{R}\mathbf{I}_{body}\mathbf{R}^T)\boldsymbol{\omega} \\
&= \mathbf{I}\dot{\boldsymbol{\omega}} + (\dot{\mathbf{R}}\mathbf{I}_{body}\mathbf{R}^T + \mathbf{R}\mathbf{I}_{body}\dot{\mathbf{R}}^T)\boldsymbol{\omega} \\
&= \mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega}\times\mathbf{I}\boldsymbol{\omega} + \mathbf{I}\boldsymbol{\omega}\times\boldsymbol{\omega} \\
&= \mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega}\times\mathbf{I}\boldsymbol{\omega}
\end{aligned} \tag{4.20}$$

Therefore it shows the angular acceleration would be

$$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega}\times\mathbf{I}\boldsymbol{\omega}) \tag{4.21}$$

4.1.4 Forward Euler Integration

In the physical simulation field, there are lots of ordinary differential equation (ODE) solvers which have been discussed [59]. We take a simple forward Euler integration to replace time integration. After a period of time step Δt, the linear and

angular velocity can be updated with equation (4.16) and (4.21). Therefore, we can obtain the new linear and angular velocity of the next time step as

$$\begin{aligned} \mathbf{v}^{n+1} &= \mathbf{v}^n + \frac{\mathbf{F}}{m}\Delta t \\ \boldsymbol{\omega}^{n+1} &= \boldsymbol{\omega}^n + \mathbf{I}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega}\times\mathbf{I}\boldsymbol{\omega})\Delta t \end{aligned} \tag{4.22}$$

After updating velocities, the position and orientation can be updated by the same approach with the forward Euler integration. Hence, equation (4.11) and (4.12) are modified as:

$$\begin{aligned} \mathbf{x}^{n+1} &= \mathbf{x}^n + \mathbf{v}\Delta t \\ \boldsymbol{\theta}^{n+1} &= \boldsymbol{\theta}^n + \boldsymbol{\omega}\Delta t \end{aligned} \tag{4.23}$$

Similarly, with quaternion method, equation (4-13) is modified as:

$$\boldsymbol{q}^{n+1} = \boldsymbol{q}^n + \frac{1}{2}\Delta t\ quat\left(\boldsymbol{\omega}^n\right)\boldsymbol{q}^n \tag{4.24}$$

However, we can adopt another concept here. The angular velocity $\boldsymbol{\omega}$ means that the rigid body is currently, rotating at a rate $|\boldsymbol{\omega}|$ about axis $\boldsymbol{\omega}/|\boldsymbol{\omega}|$. Thus, the rigid body will rotate by angle $\Delta t|\boldsymbol{\omega}|$ about $\boldsymbol{\omega}/|\boldsymbol{\omega}|$. Then, we can update the orientation $\boldsymbol{q}$ by

$$\boldsymbol{q}^{n+1} = \left(\cos\left(|\boldsymbol{\omega}^n|\Delta t/2\right), \sin\left(|\boldsymbol{\omega}^n|\Delta t/2\right)\boldsymbol{\omega}^n/|\boldsymbol{\omega}^n|\right)\boldsymbol{q}^n \tag{4.25}$$

4.2 Solid-Fluid Interaction

The two-way coupling of solid-fluid interaction with particle systems is easier to solve than grid-based methods. Several works regard rigid bodies as composition of particles, and use SPH interpolation technique to update the densities and forces of

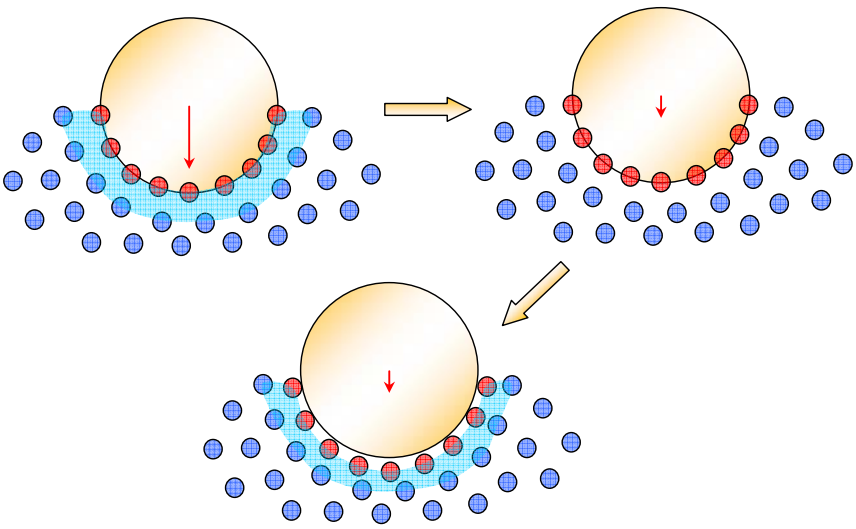


Figure 4-2: A sphere-shaped rigid body floating on fluid. We can see the distances between colliding particles and their neighboring particles are closer (the shaded region is thinner). Thus, at the next time step, the pressure forces will tend to push them apart.

these particles. The resultant forces and torques are then applied to rigid bodies, and then the rigid particles are enforced to maintain their rigidities. Another approach is that the fluid particles are regarded as rigid spheres, and the solid-fluid interaction is accomplished by processing the collisions between rigid bodies and fluid particles [24]. The former method may trap some fluid particles in the rigid body, since the rigid body is made of particles. The latter prevents the fluid particles from penetrating into rigid bodies, since the collisions between them are resolved. It is intuitive to process collisions between particles and rigid bodies to achieve the two-way coupling of solid-fluid interaction.

The collision impulse between the particle and the rigid body can be obtained by solving the collision equation. Thus, we accumulate all the collision impulses from

colliding particles, and modify the velocity of the rigid body. Then, the velocity of the rigid body is changed, and the colliding particles are also updated, as illustrated in Figure 4-2.

4.3 The Ship Modeling

In order to simulate the ship's sailing motion on the ocean, it requires establishing the ship model. We consider the ship as a rigid body, and apply equations in last section. We use ship dynamic equation of the marine control system[30] :

$$\mathbf{M}\dot{\mathbf{v}} + \mathbf{C}(\mathbf{v})\mathbf{v} + \mathbf{D}(\mathbf{v})\mathbf{v} + \mathbf{g} = \mathbf{u} + \mathbf{g}_0 + \mathbf{w} \tag{4.26}$$

where $\mathbf{v}$ is the ship's velocity. $\mathbf{M}$ is the system inertia matrix, $\mathbf{C}(\mathbf{v})$ is the Coriolis-centripetal matrix, $\mathbf{D}(\mathbf{v})$ is the damping matrix, $\mathbf{g}$ is the vector of the gravitational / buoyancy forces and moments, $\mathbf{u}$ is the vector of control inputs, $\mathbf{g}_0$ is the vector used for pre-trimming, and $\mathbf{w}$ is the vector of environment interference. Here we neglect $\mathbf{C}(\mathbf{v})$ and $\mathbf{g}_0$ terms, because the velocity is too low to consider $\mathbf{C}(\mathbf{v})$ and there are no pre-trimming term in yacht. The damping matrix is given by hydrodynamics.

The coordinate systems we use include two sets of Cartesian coordinate system, which are discussed in 4-1-1. The world coordinate system is denoted as O-XYZ, and

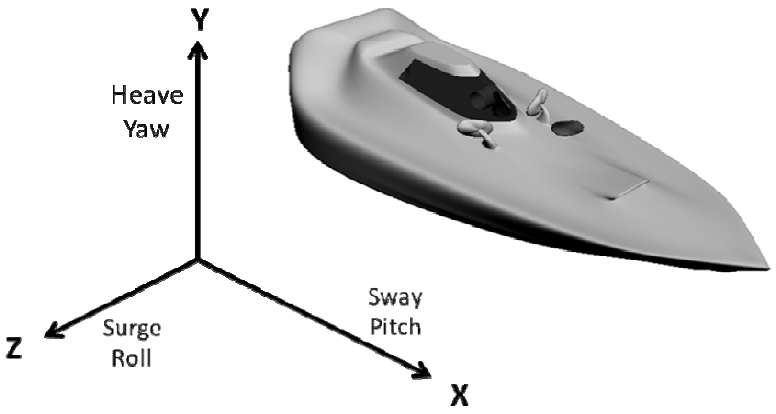


Figure 4-3: The 6 DOFs of a ship.

the body coordinate is denoted as B-XYZ. Where O is the origin of the world coordinate system, and B is the center of mass of the ship. We define the XZ plane as the horizontal plane, and Y axis directs upward. After setting of the coordinate, we can define the six degrees of freedom motion of the ship model. There are three kinds of linear motions of the ship, which are sway, heave, and surge along the x-axis, y-axis and z-axis, respectively, in the world coordinate system. Angular motions also include three kinds of motions. They are pitch, roll, and yaw about the x-axis, y-axis and z-axis, respectively (Figure 4-3).

The total force and torque is:

$$\begin{aligned} \mathbf{F}_{total} &= \mathbf{F}_{buoyancy} + \mathbf{F}_{gravity} + \mathbf{F}_{d} + \mathbf{F}_{l} + \mathbf{F}_{w} + \mathbf{F}_{ship} \\ \boldsymbol{\tau}_{total} &= \boldsymbol{\tau}_{buoyancy} + \boldsymbol{\tau}_{gravity} + \boldsymbol{\tau}_{d} + \boldsymbol{\tau}_{l} + \boldsymbol{\tau}_{w} + \boldsymbol{\tau}_{ship} \end{aligned} . \tag{4.27}$$

$\mathbf{F}_{buoyancy}$ is buoyancy, which is calculated with the height information of ocean surface. Since we know $\mathbf{F}_{buoyancy} = -\rho \cdot V$, where ρ is the density of the fluid, and V is the

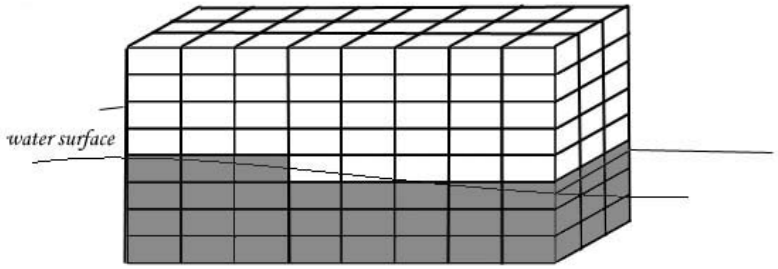


Figure 4-4: The segments of the ship. The cubbies below the ocean surface (gray part).

submerged volume of the ship. We can get V by partitioning the ship to many small cubes, and calculate each cube's vertical position based on the orientation information of the ship. We give a buoyancy force and torque if the cube is below the ocean surface as in Figure 4-4. $\mathbf{F}_{gravity}$ is gravity. $\mathbf{F}_d$ and $\mathbf{F}_L$ are drag force and lift force described in equation (2.21) and (2.22). $\mathbf{F}_w$ is the force of the wave (particle fluid), and $\mathbf{F}_{ship}$ is the force of the ship motor and propeller. The subscription's meaning is the same for $\boldsymbol{\tau}$.

We use this equation to our ship model by summing up the buoyancy, drag, lift, waves, gravitation, propeller forces and moments. These forces and moments are taken to update the position and orientation of the ship in forward Euler integration. It is the physical simulation loop.

Chapter 5
Implementation

In this chapter, we present the algorithm used in detail. The implementation is divided into several parts. The first is waves modeling, which describes how we use particle-based fluid to simulate ocean waves, with the influence of wind force. And then is ocean surface, which is formed by marching cubes method or our cloth-fitting method, and some techniques are used when rendering. The third is ship modeling and dynamics, and we also introduce the interaction between ocean and ship. We summarize the procedure and simulation loop finally.

5.1 Ocean Waves Modeling

5.1.1 Particle-based Fluid

Our ocean waves are the animation of particle-based fluid. Since we let wind force acting on fluid, and then modeling the motion of waves. The preliminary thing we have to consider is using SPH method to simulate particle-based fluid. Some field quantity of fluid are considered and calculated, as mentioned in equation (2.1). We present these properties of fluid with our codes:

```
NxFluid* CreateEmFluid()

{
   // Create fluid from emitter
   NxFluidDesc fluidDesc;
   fluidDesc.kernelRadiusMultiplier           = KERNEL_RADIUS_MULTIPLIER;//2.0
   fluidDesc.restParticlesPerMeter            = REST_PARTICLES_PER_METER;//0.8
   fluidDesc.motionLimitMultiplier            = MOTION_LIMIT_MULTIPLIER;//2
   fluidDesc.packetSizeMultiplier             = PACKET_SIZE_MULTIPLIER;//16
   fluidDesc.collisionDistanceMultiplier      = 0.2;
   fluidDesc.stiffness                        = 20.0f;
   fluidDesc.viscosity                        = 40.0f;
   fluidDesc.restDensity                      = 1000.0f;

   fluidDesc.restitutionForStaticShapes = 0.162f;
   fluidDesc.dynamicFrictionForStaticShapes = 0.146f;
   fluidDesc.restitutionForDynamicShapes = 0.5f;
   fluidDesc.dynamicFrictionForDynamicShapes = 0.5f;

   fluidDesc.simulationMethod = NX_F_SPH;
   //fluidDesc.simulationMethod = NX_F_NO_PARTICLE_INTERACTION;
   fluidDesc.maxParticles = gParticleBufferCap;

   fluidDesc.collisionResponseCoefficient = 0.5;
   fluidDesc.flags |= NxFluidFlag::NX_FF_COLLISION_TWOWAY;
   ......
   if(!bHardwareFluid)
        fluidDesc.flags &= ~NX_FF_HARDWARE;

   NxFluid* mFluid = gScene->createFluid(fluidDesc);
   return mFluid;
}
```

We use a class named NxFluidDesc to describe properties of fluid, and we can set parameters which influence these properties. For example, the smoothing radius of kernel *h* in equation (2.2) is related to fluidDesc.kernelRadiusMultiplier. Notice that as range extending, the FPS (frame per second) will decrease. The stiffness and viscosity parameters related to the influence of pressure and viscous force respectively in equation (2.11). The simulationMethod member of NxFluidDesc controls how the fluid simulated, either with particles interaction or as independent particles. We can choose NX_F_SPH or NX_F_NO_PARTICLE_INTERACTION to determine which method. On the other hand, the two-way interaction between fluid and rigid bodies/soft bodies is also a property which has to be considered. To enable the interaction, both the corresponding shape flag and the fluid flag must be set:

```
fluidDesc.flags  |=  NxFluidFlag::NX_FF_COLLISION_TWOWAY;
shapeDesc.flags  |=  NxShapeFlag::NX_SF_FLUID_TWOWAY;
```

The interaction is not necessarily symmetrical. We can set the collision response coefficient to influence fraction of the collision impulse, which is applied to the colliding rigid body. Finally we adopt hardware acceleration and use emitters to create fluid particles.

5.1.2 Wind Force Modeling

Once we have our particle-based fluid, we attempt to add wind force on fluid to simulate ocean waves. Since we want the forces can be applied to specified objects and vary at every time step, we use force fields to model wind force mentioned in section 2.4. The following function is the implementation of our wind force.

```
NxForceField* createEllipAForceField(const NxVec3& pos)

{
  // Create a Forcefield kernel
  NxForceFieldLinearKernelDesc linearKernelDesc;
 //  Wind direction
  linearKernelDesc.constant = NxVec3(-12.125f, 20.0f, 7.0f);
  linearKernelDesc.noise = NxVec3(2, 2, 2);

  NxForceFieldLinearKernel* pLinearKernel;
  pLinearKernel = gScene->createForceFieldLinearKernel(linearKernelDesc);

  // The Forcefield descriptor
  NxForceFieldDesc fieldDesc;
  fieldDesc.kernel = pLinearKernel;

  fieldDesc.rigidBodyType = NX_FF_TYPE_GRAVITATIONAL;
  //fieldDesc.rigidBodyType = NX_FF_TYPE_NO_INTERACTION;
  fieldDesc.softBodyType = NX_FF_TYPE_NO_INTERACTION;
  fieldDesc.fluidType = NX_FF_TYPE_GRAVITATIONAL;

  // A box forcefield shape descriptor
  NxBoxForceFieldShapeDesc box;
  box.dimensions.set(0.5, 15, 18);
  box.pose.t.set(pos.x, pos.y, pos.z);

  float ran = NxMath::rand(20, 40);
  float angle_box = NxMath::degToRad(ran);
```

```
  box.pose.M(0,0) = cos(angle_box);     box.pose.M(0,1) = 0;
  box.pose.M(0,2) = sin(angle_box);     box.pose.M(1,0) = 0;
  box.pose.M(1,1) = 1;                  box.pose.M(1,2) = 0;
  box.pose.M(2,0) = -sin(angle_box);    box.pose.M(2,1) =0;
  box.pose.M(2,2) = cos(angle_box);
  fieldDesc.includeGroupShapes.push_back(&box);

  NxForceField *pForceField;
  pForceField = gScene->createForceField(fieldDesc);

  return pForceField;
}
```

```
NxForceField* createSimForceField2(const NxVec3& pos)

{
  // Create a Forcefield kernel
  NxForceFieldLinearKernelDesc linearKernelDesc;
  //linearKernelDesc.constant = NxVec3(-2.0f,0.0f,-2.0f);
  linearKernelDesc.noise = NxVec3(30,30,30); //adds a random noise on the forces

  NxForceFieldLinearKernel* pLinearKernel;
  pLinearKernel = gScene->createForceFieldLinearKernel(linearKernelDesc);

  // The Forcefield descriptor
  NxForceFieldDesc fieldDesc;
  fieldDesc.kernel = pLinearKernel;

  fieldDesc.rigidBodyType = NX_FF_TYPE_GRAVITATIONAL;
  fieldDesc.softBodyType = NX_FF_TYPE_NO_INTERACTION;
  fieldDesc.fluidType = NX_FF_TYPE_GRAVITATIONAL;

  // A box forcefield shape descriptor
  NxBoxForceFieldShapeDesc box;
  box.dimensions.set(20, 3, 20);
  box.pose.t.set(pos.x, pos.y, pos.z);
```

```
    fieldDesc.includeGroupShapes.push_back(&box);
    NxForceField *pForceField;
    pForceField = gScene->createForceField(fieldDesc);

    return pForceField;
}
```

Force fields have two important geometric properties: force field kernel and the volume of activity. The linear kernels implement force functions described in equation (2.21) and (2.22). We create several force fields with different kernels and superpose their influence on fluid as mentioned in equation (2.18). We can also create our own custom kernel in the simulation, but it is not necessary to do that. For each kind of object in the field, such as fluid, rigid body, soft body, etc., has its own scaling properties. We can set scaling of each type with NX_FF_TYPE_GRAVITATIONAL and NX_FF_TYPE_NO_INTERACTION. The volume of activity is determined by the shapes belonging to force field groups. We can use different shapes, for example, sphere, capsule, box and convex mesh. Notice that the actual volume used is the union of all included shapes' volumes, minus the union of all excluded shapes' volumes. We can push these two kinds shape into includeGroupShapes vector and excludeGroupShapes vector respectively. We add random term to the direction and the magnitude of force field to simulate the turbulence or noise of wind.

After we construct our wind force field, we have to update it at every time step. Then we can model wind with the following algorithm.

Algorithm 1: UpdateWindForce (timestep)

```
if wind force field created
  for every force field
      set velocity of each force field
      update position of each force field with velocity * timestep
      if the position out of border
          reset the position of the force field
          change the direction and magnitude randomly
      end if
  end for
end if
```

5.2 Ocean Surface Construction and Rendering

5.2.1 Marching Cubes

In the simulation, we have position data of each fluid particle. But since we prefer to see continuous ocean surface instead of a set of particles, we attempt to use marching cubes algorithm to generate triangle meshes, which is our ocean surface. We present several algorithms as below:

Algorithm 2: InitMarchingCubes ( )

```
//Set the configuration such as resolution for Marching Cubes
isosurface_radius ← radius
marchingcubes_stride ← stride
```

```
marchingcubes_range ← range
allocate memory to volume of sph_render*
set width, height, depth of volume
```

As we initialize our particle-based fluid, we also set the configuration for marching cubes method, which influence the performance of surface.

Algorithm 3: MarchingCubesCallBack ( )

```
for every fluid particles
   if render_particle_num >= N_PARTICLES
        return
   end if
   if point inside bounding box
        sph_render data ← particle's position
   end if
end for
//Construct surface by Marching Cubes algorithm
sph_render_create_implicit( sph_render*, isovalue, pos, num, bounding box)
```

During each simulation loop, we construct scalar fields according to the distance from the center of each particle; namely, assign a value to vertices of each cube. Every cube require an index, which is related to the state of vertices, to look up table. Then, we can locate the surface corresponding to our specified value, as one of 15 patterns in Figure (3-2).

Algorithm 4: MarchingCubesRendering( )

```
for every cubes
   calculate the position of interaction linearly interpolated
   calculate the normal of interaction linearly interpolated
   draw triangles
end for
```

Once we calculate the position and normal, we can render triangles to form surface.

5.2.2 Fitting with Cloth

With marching cubes algorithm, since it requires forming triangle meshes in every simulation loop, and the cost of calculation is expensive. This is drawback to our real-time applications. We present a new method to construct ocean surface, which uses cloth to fit particle-based fluid. The following code presents properties of cloth model.

```
void CreateCloth(NxReal width, NxReal length, NxReal d)

{
  // Create Cloth
  NxClothDesc clothDesc;
  clothDesc.globalPose.t = NxVec3(0, height, 0); //height=6.0
  clothDesc.thickness = 1.5;
  clothDesc.bendingStiffness = 0.01;
  clothDesc.stretchingStiffness = 0.01;
  clothDesc.dampingCoefficient = 0.001;
  clothDesc.density = 50;
  clothDesc.fromFluidResponseCoefficient = 1.0;
  clothDesc.toFluidResponseCoefficient = 0.0;
  clothDesc.flags |= NX_CLF_BENDING;
  clothDesc.flags |= NX_CLF_VISUALIZATION;
  clothDesc.flags |= NX_CLF_FLUID_COLLISION;
  clothDesc.flags |= NX_CLF_DISABLE_COLLISION;
  clothDesc.flags &= ~NX_CLF_GRAVITY;

  if (gHardwareCloth)
       clothDesc.flags |= NX_CLF_HARDWARE;

  MyCloth *regularCloth = new MyCloth(gScene, clothDesc, width, length, d,
                                      "nvidia.bmp");

  ......
```

```
    int numX = (int)(width / d) + 1;
    int numY = (int)(length / d) + 1;
    int i,j;

    i = 0;
    for (j = 0; j <= numX; j++)
          regularCloth->getNxCloth()->attachVertexToGlobalPosition(i*(numX+1)+j,
          NxVec3(OCEAN_WIDTH/2.0f-j*d, height-0.5, OCEAN_LENGTH/2.0f));
    i = numY;
    for (j = 0; j <= numX; j++)
          regularCloth->getNxCloth()->attachVertexToGlobalPosition(i*(numX+1)+j,
          NxVec3(OCEAN_WIDTH/2.0f-j*d, height-0.5, -OCEAN_LENGTH/2.0f));
    j = 0;
    for (i = 0; i <= numY; i++)
          regularCloth->getNxCloth()->attachVertexToGlobalPosition(i*(numX+1)+j,
          NxVec3(OCEAN_WIDTH/2.0f, height-0.5, OCEAN_LENGTH/2.0f-i*d));
    j = numX;
    for (i = 0; i <= numY; i++)
          regularCloth->getNxCloth()->attachVertexToGlobalPosition(i*(numX+1)+j,
          NxVec3(-OCEAN_WIDTH/2.0f, height-0.5, OCEAN_LENGTH/2.0f-i*d));
}
```

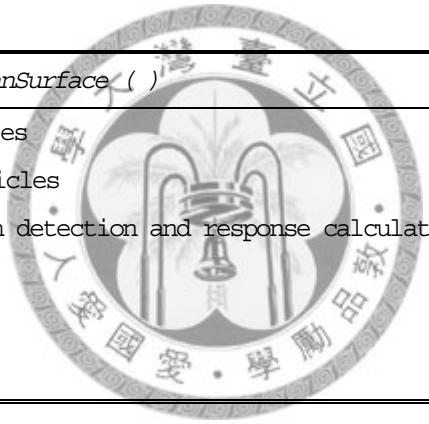
The cloth physical model provides a mesh, which defines a set of particles connected with springs. Here we consider two main types of cloth constraints, which are stretching and bending, as mentioned in Figure 3-3. Since we use cloth to be our ocean surface, in this situation, we want to decrease constraints between particles, so they can move more unrestrained. Thus, setting the bendingStiffness and stretchingStiffness to very small values. The other thing has to be noticed is that we attempt to let fluid particles influence cloth and construct ocean surface, and we do not want the cloth influence fluid particles. In other words, it is an one-way

interaction which does not change direction or velocity of waves. We set the descriptive parameter of cloth, fromFluidResponseCoefficient to 1.0, and set toFluidResponseCoefficient to 0.0. We also add NX_CLF_DISABLE_COLLISION flag to avoid colliding with other rigid body, and do not enable self collision. Finally, fixing the boundary of cloth to stationary positions, this increases the stability of surface and prevents it moving away. In addition, cloth particles can also be pinned to shapes and global positions.

In the simulation loop, collision detection is performed with cloth particles.

Algorithm 5: UpdateOceanSurface ( )

```
for every cloth particles
  for every fluid particles
      do the collision detection and response calculation
  end for
end for
```



5.2.3 GPU Shading

In order to perform more realistic and faster rendering, we adopt Cg language to calculate color on GPU instead of CPU. There are two main parts of pipeline in shader of Cg, the first part is called vertex program, and the other is called fragment program. The former mainly does mathematical transformations such as modeling,

viewing and projection transformation, and the vectors mentioned in Figure (3-4) is processed and calculated as below:

```
void v_basicLight(    float4 position     : POSITION,
                      float4 color        : COLOR0,
                      float3 normal       : NORMAL,

                      out float4 oPosition    : POSITION,
                      out float4 oColor       : COLOR0,
                      out float3 oEyeVec      : TEXCOORD0,
                      out float4 oNormal      : TEXCOORD1,
                      uniform float3 eyePosition)
{
  oPosition = mul(glstate.matrix.mvp, position);
  oColor = color;
  oEyeVec = position.xyz - eyePosition;
  oNormal.xyz = normal;
}
```

Then, the vertex program sends the vertex information to the fragment program, and the fragment color is processed and calculated.

```
Float4 f_basicLight( in float4 Col0      : COLOR0,
                     in float3 EyeVec    : TEXCOORD0,
                     in float4 INP       : TEXCOORD1,

                     uniform samplerCUBE EnvCMap,
                     uniform sampler2D  FoamMap ) : COLOR
{
  float3 N = INP.xyz;
  float3 Eye = normalize(EyeVec.xyz);
  float3 Normal = normalize(N);

  //// Calc reflection vector
```

```
    float  dotEye_N = dot(Eye, Normal);
    //float3 reflectVec = 2*Normal*dotEye_N - Eye;  /**dot(N,N)*/
    float3 reflectVec = reflect(Eye, Normal);
    // Add sun road splashes
    float3 reflectColor = texCUBE(EnvCMap, reflectVec.xzy*float3(1,-1,1)).xyz;
    //HDR
    reflectColor*=1+saturate(reflectColor.x - 0.9)*100.0h;
}
```

The reflection is determined by the environment, which is our sky cube. The texCUBE is the function to get the color. Besides, doing HDR can show the radiance of sunshine on the ocean surface better. We implement the equation (3.16) in last line.

```
//Fresnel

    float nair=1;                    //refractive index of air
    float nwater=1.333;              //refractive index of water
    float ci= dotEye_N/(length(Eye)*length(Normal));
    float angi = acos(ci);
    float angt = asin(nair*sin(angi)/nwater);
    float sum = angi+angt;
    float mins = angi-angt;
    half Freshel = max(0, min(1 , 0.02 + 0.5*(pow(sin(mins)/sin(sum), 2)
                          + pow(tan(mins)/tan(sum), 2))));
```

The Fresnel term determines the ratio of the reflection and refraction. we calculate $\cos\theta_i$ in equation (3.9), and get the reflection coefficient in equation (3.8).

```
    //Foam

    INP.w = (Wave.y/0.8)
    half Foam = saturate(INP.w);
    Foam=Foam*(tex2D(FoamMap, INP.xz).w*1.5h);
```

The foam effect is also considered on the ocean surface, and the transparency is assigned in vertex program. It is also showed by texture mapping.

Algorithm 6: CalaulateFinalColor ( )

ambient $\leftarrow K_a \cdot$ deepColor
emissive $\leftarrow K_e \cdot$ (1-Fresnel) $\cdot$ waterColor
diffuse $\leftarrow K_d \cdot$ Fresnel $\cdot$ reflecColor //including HDR
specular $\leftarrow$ (max(Normal $\cdot$ (Light-Eye), 0)$^{\text{shininess}}$
totalColor $\leftarrow$ emissive+ambient+diffuse+specular
//Foam
totalColor $\leftarrow$ lerp(totalColor, coefficientFoam, Foam)

At the end of fragment program, we calculate color as mentioned in equation (3.11). All color terms are considered and the final color is rendered on the screen.

5.3 The Ship Modeling and Dynamics

After constructing the ocean surface, we present our ship modeling and implementation of rigid body dynamics. We create a rigid body according to the shape of the loading model, and let the rigid body interact with fluid particles and ocean surface.

```
void GenerateShip(float scale, const NxVec3& pos)

{
  // Create body
  NxBodyDesc BodyDesc;
  BodyDesc.angularDamping = 0.4;
  BodyDesc.linearDamping = 0.2;
```

```
    //Create ship according to the shape
    int nbInsideCirclePts = inside_vertices;
    int nbOutsideCirclePts = outide_vertices;
    int nbVerts = nbInsideCirclePts + nbOutsideCirclePts;

    NxVec3* verts = new NxVec3[nbVerts];
    GenerateShipSidePts(verts, scale, 0.0f);
    GenerateShipSidePts(verts+nbInsideCirclePts, scale, height); // height=5.0f

    // Create descriptor for convex mesh
    NxConvexMeshDesc convexDesc;
    convexDesc.numVertices            = nbVerts;
    convexDesc.pointStrideBytes       = sizeof(NxVec3);
    convexDesc.points                 = verts;
    convexDesc.flags                  = NX_CF_COMPUTE_CONVEX;
    convexDesc.flags |= NxShapeFlag::NX_SF_FLUID_TWOWAY;
    .......
    convexShapeDesc.shapeFlags|= NxShapeFlag::NX_SF_FLUID_TWOWAY;
    convexShapeDesc.shapeFlags|= NxShapeFlag::NX_SF_CLOTH_DISABLE_COLLISION;
    if(convexShapeDesc.meshData)
    {
         NxActorDesc ActorDesc;
         ActorDesc.shapes.pushBack(&convexShapeDesc);
         ActorDesc.body          = &BodyDesc;
         ActorDesc.density       = ship_density;
         ActorDesc.globalPose.t = pos;
         ActorDesc.name = "Ship";
         convexActor = gScene->createActor(ActorDesc);
    }
    delete[] verts;
}
```

First we set the angularDamping and linearDamping parameters of the rigid body, which influence the change of angular velocity and velocity. Since we create our ship according to the shape of the loading model, it requires assigning each vertex of the

ship, then using function GenerateShipSidePts to create our ship. The most important properties of the ship is it can interact with fluid particles, but cannot influence the cloth (ocean surface), directly. Setting NxShapeFlag::NX_SF_FLUID_TWOWAY and NxShapeFlag::NX_SF_CLOTH_DISABLE_COLLISION flags can achieve these requirements. Finally we set other properties such as density, position, name, etc.

Algorithm 7: UpdateShipPose ( )

```
//Accumulate all the forces and torques
F_total ← ∑Force
τ_total ← ∑Torque          //equation (4.27)
//Update position
a ← F_total / m
v ← a · timestep
x ← v · timestep
//Update orientation
α ← I^-1(τ_total - ω×Iω)
ω ← α · timestep
θ ← ω · timestep
```

In the simulation loop, it requires accumulating all the forces and torques, as mentioned in equation (4.27). The wave force is added with the collision between the ship (rigid body) and fluid particles, and this is a two-way interaction in nature, so it changes not only ship's pose but also waves' shape. Since we set rigidBodyType in the force field to be NX_FF_TYPE_GRAVITATIONAL, the wind force, namely, the drag force and lift force is also added. The gravity is added automatically and the ship

force is under our control. So the buoyancy is one thing that requires more effort, and we have to calculate the volume under the ocean surface. The following is our codes:

```
NxReal HoskissNxShip::getWaveHeight(MySurface *my_surface, NxReal x, NxReal z)



{
  NxVec3 surface_pos;
  float index_i;
  float index_j;
  float nop = my_surface ->getNxCloth()->getNumberOfParticles();
  float sqrt_nop = sqrt(nop);

  index_i = sqrt_nop*(_ocean_width/2.0f-x)/_ocean_width;
  index_j = sqrt_nop*(_ocean_length/2.0f-z)/_ocean_length;
  surface_pos = my_surface->getNxCloth()
              ->getPosition((int)index_j*sqrt_nop+(int)index_i-1);
  return surface_pos.y;
}
```

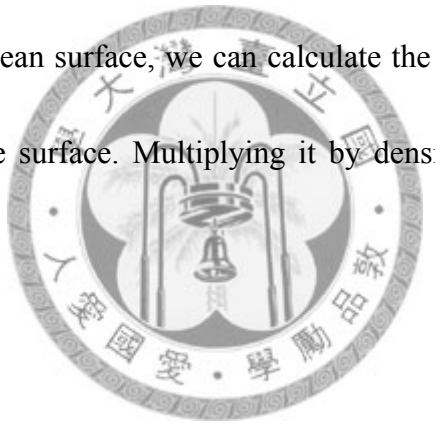
```
void HoskissNxShip::add Buoyancy (NxReal gDeltaTime)

{
  for(unsigned int j = 0; j < _totalBottNum; j++)
  {
      NxVec3 nxGraGloPos(_segBottGloPos[j].x, _segBottGloPos[j].y +
                         _shipSize.y/2.0f, _segBottGloPos[j].z);
      _shipActor->addForceAtPos(NxVec3(0,
                  -_density*_segVolume*GRAVITY*(float)gDeltaTime, 0),
                  nxGraGloPos, NX_IMPULSE, false);

      if(_waveHeightDiff[j]<0)//under the ocean surface
      {
         NxVec3 nxBuoGloPos(_segBottGloPos[j].x,
                            _segBottGloPos[j].y-_waveHeightDiff[j]/2.0f,
```

```
                                _segBottGloPos[j].z);
          _shipActor->addForceAtPos(NxVec3(0,
                b_coeff*(_segVolume*-_waveHeightDiff[j]/(float)_shipSize.y)
                *GRAVITY*(float)gDeltaTime, 0), nxBuoGloPos, NX_IMPULSE, false);
        }//end if
   }//end for
}
```

Given the planar position of the ocean surface, we create function getWaveHeight to get surface height at this specified position. Now that we know where the ship is, we can get each surface height at the area where the ship occupied. If the ship is under the ocean surface, we can calculate the difference between them, and get volume under the surface. Multiplying it by density, that is our method to calculate buoyancy.



5.4 Integrated Simulation Loop

In the simulation loop, the pipeline of our graphical and physical system is:

Algorithm 8: IntegratedSimulationLoop ( )

```
For each frame()
   Update Wind Force()
   Interaction between Wind and Fluid()
   Interaction between Fluid and Surface()
   Interaction between Wind, Fluid, Surface and Ship()
   Accumulate forces and torques()
   Update Position and Orientation()
   Render()
```

```
Frame end()
```

Update Wind Force() updates the direction, velocity and range of wind, and it influence particle-based fluid. The ocean surface and waves is then updated with fluid. As we know the height of ocean surface, we can calculate buoyancy and accumulate all other forces from wind, waves, control forces and gravity. The position and orientation of ship are updated, and we render all visible objects on the screen finally.

Chapter 6
Experimental Results

In this chapter, we demonstrate our experiments on simulating ocean and ship, and the simulation can be applied to the 6-DOF motion platform. Our CPU is Intel Core i7 940 2.93GHz, Graphic card is Nvidia 9800GTX 512MB, Memory is 3GB, and particles number is 6000. All the system is implemented with C++, PhysX SDK, OpenGL, Cg and DirectInput.

6.1 Ocean

We construct our ocean surface scene with particle-based fluid, wind model and ocean surface. The size of ocean surface (cloth) is relative to the cost of computation and FPS, and we select an appropriate one to fit fluid particles. We show the difference between rendering and not rendering ocean surface in Figure 6-1. It seems the surface can fit fluid particles adequately.

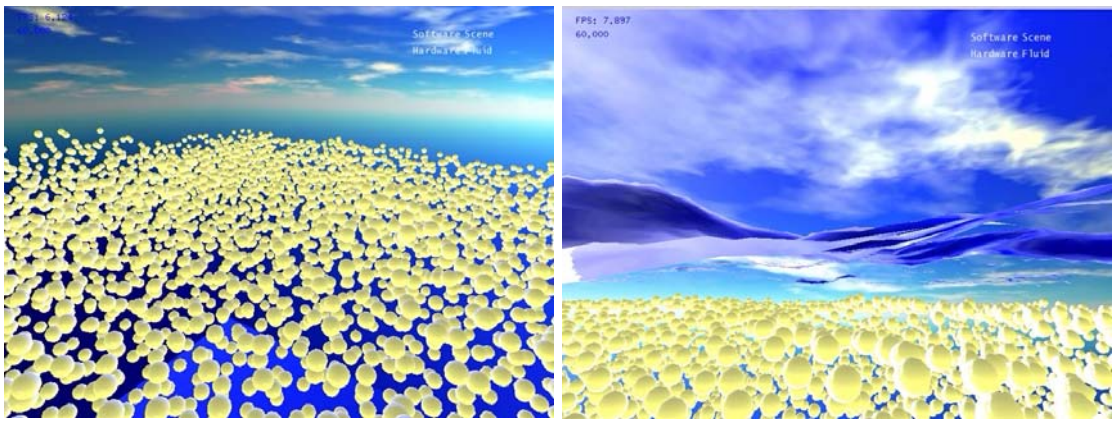
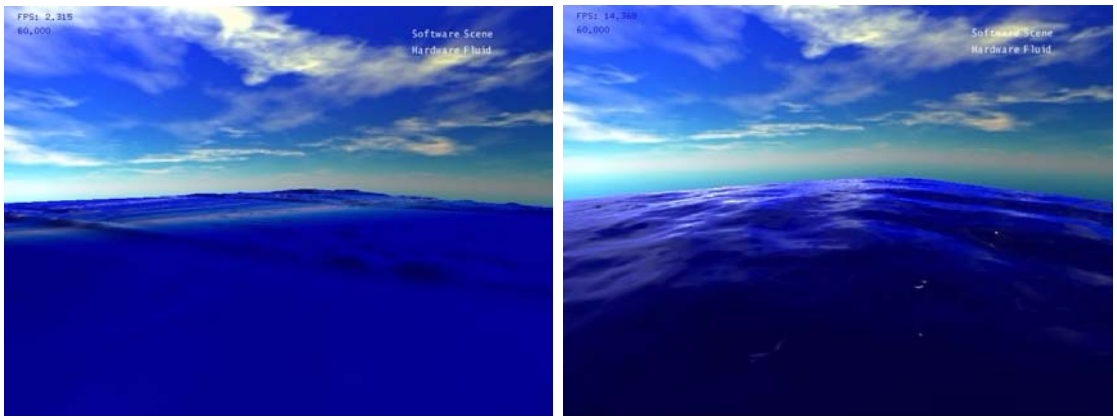


Figure 6-1 Particle-based fluid and fitting with surface.

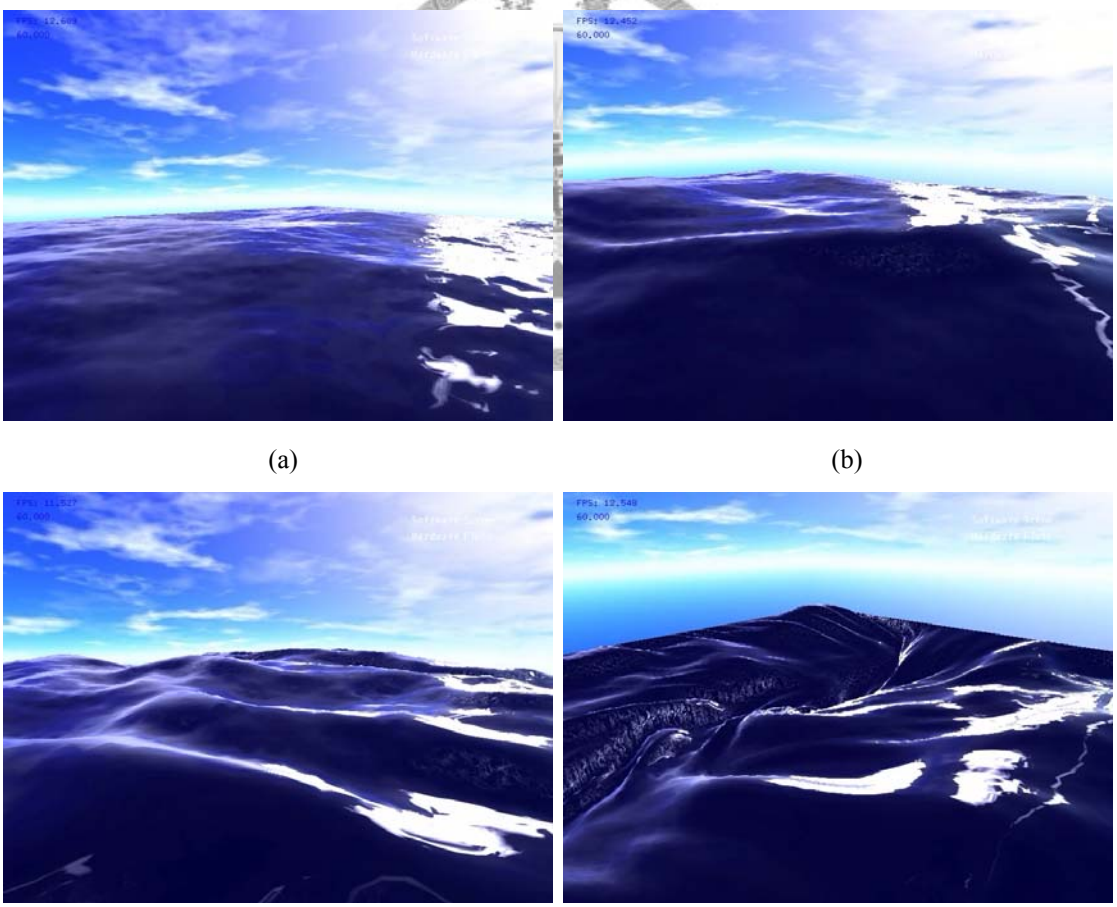
We also compare marching cubes algorithm and our ocean surface in Figure 6-2.



(a) (b)

Figure 6-2: (a) Marching cubes method. (b) Our ocean surface

We control the range, speed, direction, amplitude of wind forces as below:



(a) (b)

(c) (d)

Figure 6-3: Ocean Waves influenced with different wind range, speed, direction, amplitude.

The Figure 6-3 shows different wind range, speed, direction and amplitude cause different amplitude of waves. Figure 6-3(a) shows that we only add noise wind force term on fluid. Figure 6-3(b) shows smaller range and speed of wind force. Figure 6-3(c) is larger range and speed. We even try to create vortex in Figure 6-3(d).

White-capping is the phenomenon occurs as the surface moves too steep, and it is an usual scene on the ocean. We also consider this phenomenon in our simulation. (Figure 6-4)

(a) (b)

Figure 6-4: (a) Real ocean white-capping. (b) Our ocean white-capping

The shader is implemented in the scene. We determine the ration of reflection and refraction according to Fresnel effect. This can let the ocean scene seem more realistic, and it is hard to see the motion of waves without the effect. Finally, with HDR, the color information of the environment can be calculated and mapped on the ocean surface, and we can compare the differences in Figure 6-5.

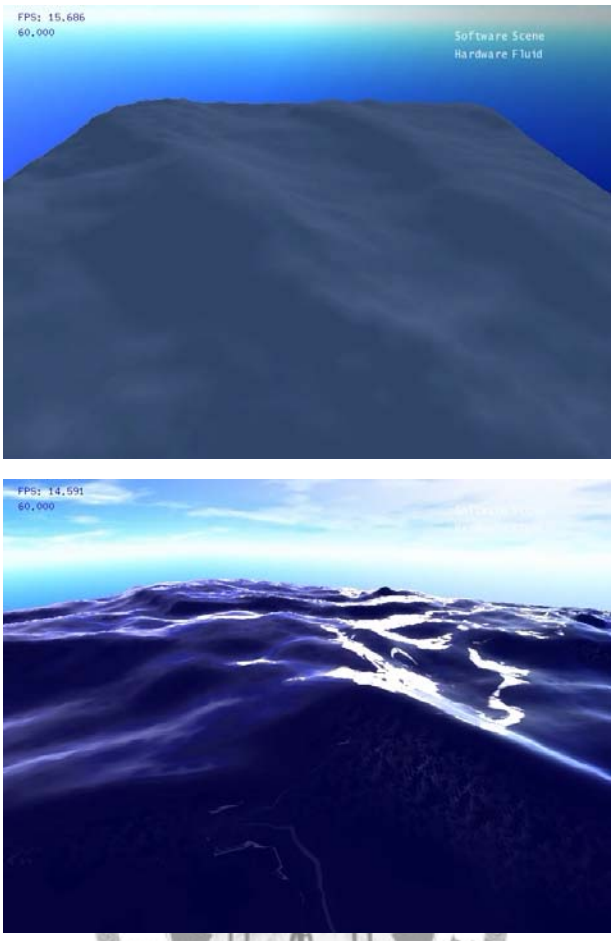


Figure 6-5: Image with and without Fresnel effect, white-capping and HDR

6.2 Ship

To do the two-way coupling simulation between solid and fluid, is the main reason we use particle-based fluid. In this thesis, our ship can interact with wind, waves and ocean surface, and we can get more physical and realistic motion of ship. For the convenience of sailing the ship, we change the camera mode to first-person view to navigate the ship, or third-person view to observe the interaction between ship and ocean. We can control the ship by the propellers at the back of ship. These results are shown as below:

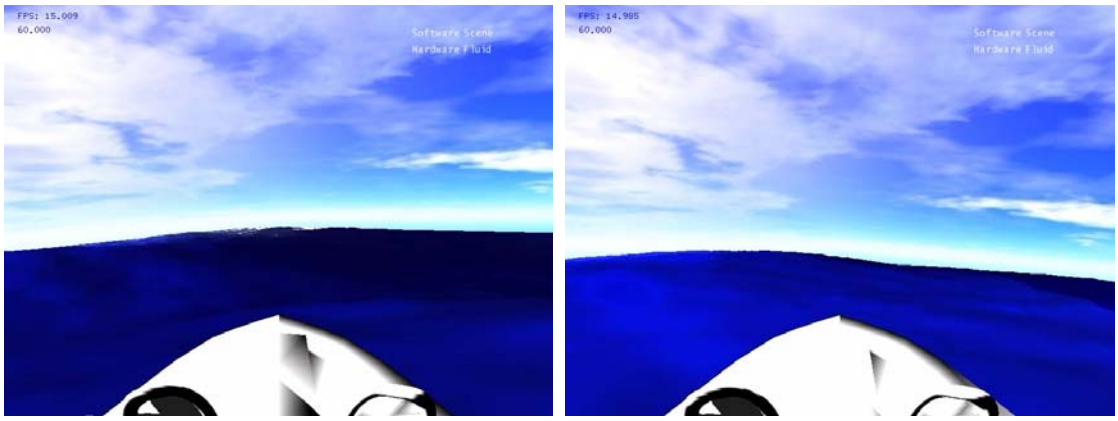
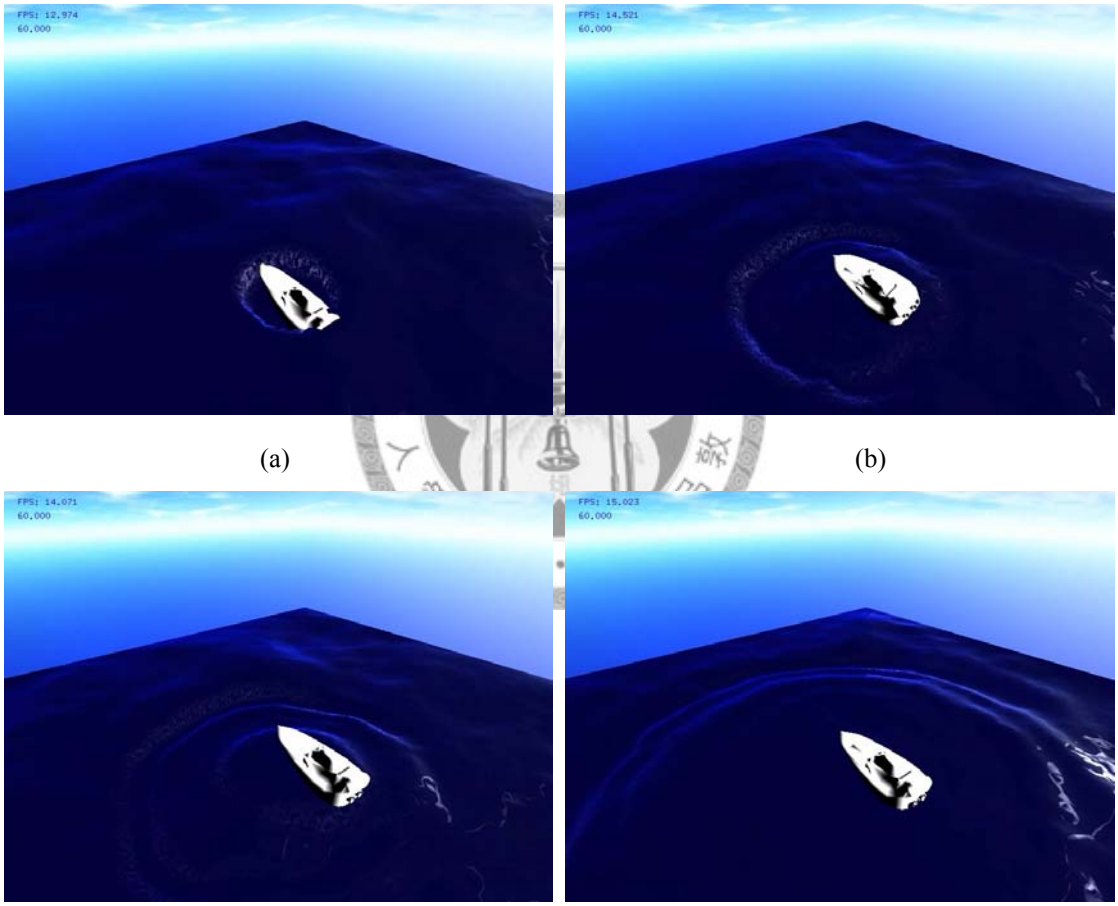


Figure 6-6: First-person view of ship



(a) (b)

(c) (d)

Figure 6-7: Two-way coupling simulation

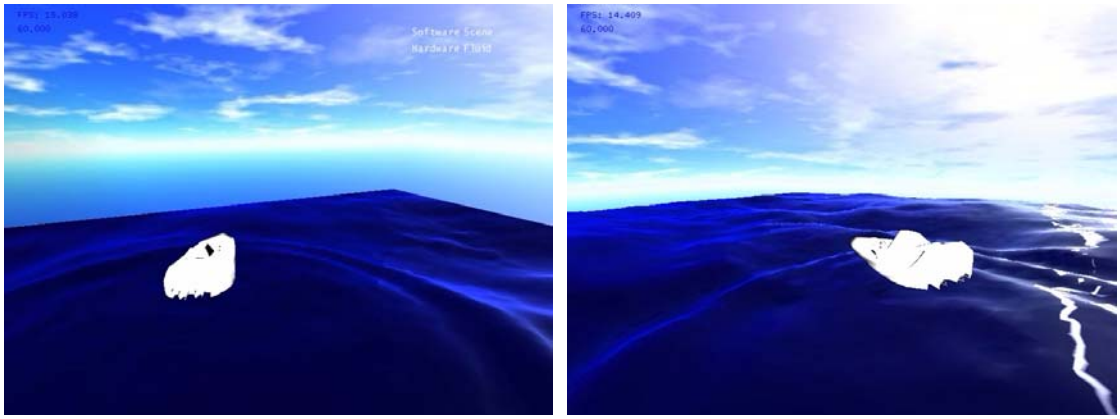


Figure 6-8: The ship sailing on the ocean

6.3 System Integration with 6-DOF Motion Platform

In Virtual-Reality field, driving simulator is one kind of advanced equipment which creates realistic driving sensations in a laboratory environment. We developed the driving simulator system, which is divided into three subsystems. As shown in Figure 6-9, it includes vehicle dynamics system, washout filter system, and motion platform system respectively. Our 6-dof motion platform is built up with six hydraulic actuators and a cabin, and there are also a projector, a joystick and a seat mounted in the cabin. Joystick includes steering wheel, throttle pedal and a braking pedal in front of the chair.

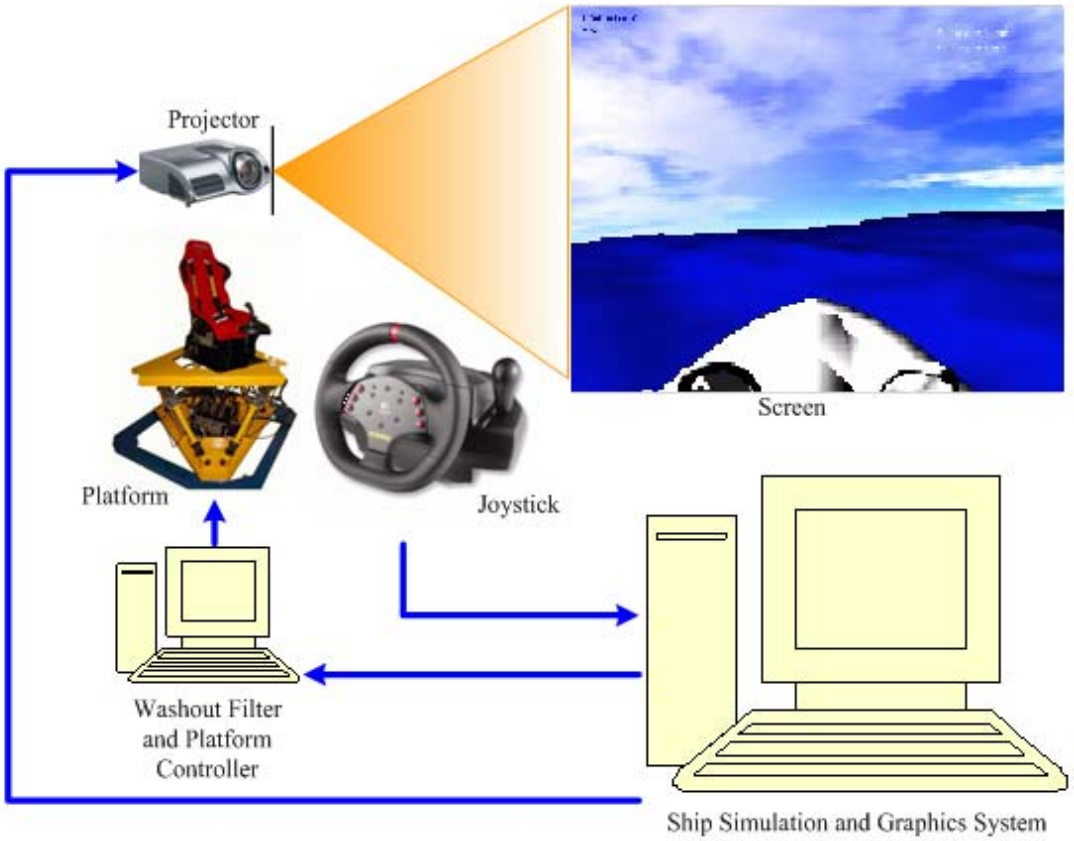


Figure 6-9: The system architecture of our VR-based motion simulator.

In the experiment, our ship simulation system receives the command inputs from the joystick, and performs graphical and physical results in the simulation. The image is displayed on the screen through the projector. The physical data such acceleration and angular velocity are sent to the washout filter to control the platform motion. The driver can see the displayed scene on the screen inside the cabin.

Our ship simulation is based on the driving simulation system. The various motion of ship can be demonstrated on the platform reliably, and high fidelity and smooth motion can be achieved, as shown in Figure 6-10. Once we feel the motion of

6-DOF platform and see the realistic scene on the screen, we can immerse ourselves into the virtual reality of sailing a ship.

Pitch

Roll

Yaw

Figure 6-10: The 6-dof motion of our platform.

Chapter 7
Conclusions

In this thesis, we present some new methods to achieve our goal of simulating ocean waves. Particle-based fluid is appropriate for interaction with rigid bodies in nature, but it is seldom used to construct large scenery such as ocean. We attempt to create various sizes of fluid particles to create our ocean. From the hint of nature, we model wind forces acting on these fluid particles to create ocean waves. Unlike general approaches such as Gerstner model, FFT or Eulerian method, it is a kind of new methods to generate ocean waves.

One of drawbacks of particle-based fluid is hard to draw the surface though we know their positions. Using physically simulated cloth to fit the surface is also a new attempt to construct ocean surface. Once we set appropriate parameters and properties of cloth, we can get pleasant results and satisfying our requirement. Considering Fresnel effect and HDR can obtain more realistic rendering in visualization.

Our ship can interact with wind, waves and surface in the physics-based system, since collision can be detected and processed. Two-way coupling is achieved in nature, and we also calculate buoyancy with the height information of surface. The total

forces and torques acting on the ship update ship's pose, and we set these data to washout filter to control 6-DOF motion platform. With our simulator, we construct a physical and graphical system of sailing a ship. This system of virtual-reality can be widely used in other applications, such as training simulator, entertaining equipment, or even in riding film theater.

There are still some interesting challenges left. In the future, combining our ocean surface with unbounded ocean surface (projected grid concept) is one goal to achieve, and then we can sail our ship freely. Applying level-of-detail method is another way to increase the efficiency of simulation. There are some algorithm presented to make translucent objects seem more realistic, such as "Rendering translucent materials using photon diffusion" or "An Empirical BSSRDF Model" These papers are written by Henrik Wann Jensen in University of California, San Diego. On the other hand, how to add ripples or splashes with our surface is also a task to fulfill, this makes more physical and realistic scenery on the ocean. Finally, we sincerely hope this thesis is helpful to the people who are interested in the physical simulation and computer graphics field.

Reference

[1] A. Fournier and W. T. Reeves, "A simple model of ocean waves," in *SIGGRAPH* '86(1986), 75–84

[2] J. C. Gonzato and B. L. Sa¨ec, "On modeling and rendering ocean scenes," *J. of Visualization and Computer Animation* 11(1) (2000) 27–37

[3] G. A. Mastin, P. A. Watterberg, and J. F. Mareda, "Fourier synthesis of ocean scenes," *IEEE Computer Graphics and Applications* 7, 3 (Mar.), 1987 16–23.

[4] W. J. Pierson and L. Moskowitz, "A proposed spectral form for fully developed wind seas based on the similarity theory of s.a. kilaigorodskii," *Journal of Geophysical Research*, pages 5181–5190, 1964.

[5] J. Tessendorf, "Simulating ocean water," In *Siggraph Course Notes*, Addison-Wesley. 1999.

[6] S. Thon, J. M. DISCHLER, and D. Ghazanfarpour, "Ocean waves synthesis using a spectrum-based turbulence function," In *Computer Graphics International Proceeding*, 2000.

[7] D. Hinsinger, F. Neyret, and M.P. Cani, "Interactive animation of ocean waves," In: *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation.* (2002) 161–166

[8] K. Perlin, "An image synthesizer," In *Computer Graphics (SIGGRAPH '85 Proceedings)*, B. A. Barsky, Ed., vol. 19(3), 287–296. 1985.

[9] D. F. Young, B. R. Munson, and T. H. Okiishi, "Brief Introduction to Fluid Mechanics."

[10] N. Foster, and D. Metaxas, "Realistic animation of liquids," *Graph. Mod. and Im. Proc.* 58, 5, 471–483. 1996.

[11] N. Foster, and D. Metaxas, "Controlling fluid animation," 178–188. 1997.

[12] J. Stam, "Stable fluids," In *SIGGRAPH* '99, 121–128. 1999.

[13] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating water and smoke with an octree data structure," In *SIGGRAPH*'04, 457–462. 2004.

[14] G. Irving, E. Guendelman, F. Losasso, and R. Fedkiw, "Efficient simulation of large bodies of water by coupling two and three dimensional techniques," In *ACM Trans. Graph.* 25, 3, 805–811. 2006.

[15] B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O'brien, "Fluid animation with dynamic meshes," In *SIGGRAPH* '06.

[16] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," *The Astronomical Journal*, vol. 82, pp. 1013-1024, 1977.

[17] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics-Theory and application to non-spherical stars," *Royal Astronomical Society, Monthly Notices*, vol. 181, pp. 375-389, 1977.

[18] G. Miller and A. Pearce, "Globular dynamics: A connected particle system for animating viscous fluids," *Computers and Graphics*, vol. 13, no. 3, pp. 305-309, 1989.

[19] M. Desbrun and M. P. Cani. "Smoothed particles: A new paradigm for animating highly deformable bodies," In *Computer Animation and Simulation* '96 (*Proceedings of EG Workshop on Animation and Simulation*), pages 61–76, Springer-Verlag, Aug 1996.

[20] D. Tonnesen. "Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation," PhD thesis, University of Toronto, November 1998.

[21] M. Muller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the ACM SIGGRAPH /Eurographics symposium on Computer animation.* San Diego, California: Eurographics Association, 2003.

[22] W. E. Lorensen, and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 3, pp. 163-169, July 1987.

[23] S. Premože, T. Tasdizen, J. Bigler, A. Lefohn, and R. T. Whitaker, "Particle-based simulation of fluids," *Computer Graphics Forum*, vol. 22, pp. 401-411, 2003.

[24] S. Clavet, P. Beaudoin, and P. Poulin, "Particle-based viscoelastic fluid simulation," In *SCA* 2005, 219–228.

[25] P. Kipfer, and R. Westermann, "Realistic and interactive simulation of rivers,"

In *Proceedings Graphics Interface* 2006, 41–48.

[26] K. S. M. Davidson and S. L.I., Turning and Course Keeping Qualities of Ships, SNAME Transaction 1946.

[27] M. A. Abkowitz, Lectures on Ship Hydrodynamics-Steering and Manoeuvrability, Report Hy-5, Hydro-and Aerodynamic Laboratory, Lyngby, Denmark, 1964.

[28] M. Hirano, Calculation Method of Ship Maneuvering Motion at Initial Design Phase, J. SOC. NAVAL ARCHIT. JAPAN, vol. 147, pp. 144-153, 1980.

[29] S. Inoue, M. Hirano, K. Kijima, and J. Takashina, A Practical Calculation Method of Ship Maneuvering Motion, International Shipbuilding Progress, vol. 28, pp. 207-222, 1981.

[30] T. I.Fossen, Marine Control System: Marine Cybernetics, 2002.

[31] A. W. Browning, A mathematical model to simulate small boat behaviour, SIMULATION, vol. 56, p. 329, 1991.

[32] Robert T. Hudspeth, Waves and Wave Forces on Coastal and Ocean Structures: World Scientific, 2006.

[33] Blair Kinsman, Wind Waves: Their Generation and Propagation on the Ocean Surface: Prentice-Hall, 1965.

[34] M. Pharr, GPU Gems 2: Programming Techniques, Tips and Tricks for Real-Time Graphics: Addison Wesley, 2005.

[35] R. Fernando and M. J. Kilgard, The Cg Tutorial: Addison Wesley, 2005.

[36] S.-C. Wang, "System Design of VR-based Motion Simulatior for Wheeled Vehicle on Three Dimension Terrain Application." vol. Master Taipei: National Taiwan University, 2004.

[37] C.-D. Lee, "Impulse-Based Dynamic Simulation of Articulated Rigid Bodies with Aerodynamics." vol. Master Taipei: National Taiwan University, 2006.

[38] C.-T. Chou, "VR-based Motion Simulator for Ships on Real-time Rendered Dynamic Ocean." vol. Master Taipei: National Taiwan University, 2007.

[39] H.-C. Chen, "Real-time Two-way Coupling of Ship Simulator with VR Application." vol. Master Taipei: National Taiwan University, 2008.

[40] D. J. Price, "Magnetic Field in Astrophysics," PhD Thesis, Institute of

Astronomy, University of Cambridge, Oct. 2004.

[41] J. J. Monaghan, "Smoothed Particle Hydrodynamics," *Reports on Progress in Physics*, vol. 68, pp. 1703-1759, 2005.

[42] J. J. Monaghan, "Smoothed Particle Hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, pp. 543-574, 1992.

[43] M. Desbrun and M.-P. Gascuel, "Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies," in *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*, Aug 1996, pp. 61-76.

[44] E. J. Tarbuck and F. K. Lutgens, "Earth Science," Prentice Hall, 2002

[45] D. Stora, P. O. Agliati, M. P. Cani, F. Neyret, and J. D. Gascuel, "Animating lava flows." in *Graphics Interface*, 203–210. 1999.

[46] I. D. Rosenberg and K. Birdwell, "Real-time particle isosurface extraction," In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, *ACM*, New York, NY, USA, 35–43. 2008.

[47] B. W. Williams, "Fluid Surface Reconstruction from Particles," Master's thesis, The University Of British Columbia. 2008.

[48] R. Malladi and J. A. Sethian, "Level set methods for curvature flow, image enhancement, and shape recovery in medical images," In *Proc. of Conf. on Visualization and Mathematics*, Springer-Verlag, 329–345. 1995.

[49] M. Muller, S. Schirm and S. Duthaler, "Screen space meshes," In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, Eurographics Association*, Aire-la-Ville, Switzerland, 9–15. 2007.

[50] D. Terzopoulos, J. C. Platt, and A. H. Barr. "Elastically deformable models," *Computer Graphics (Proc. SIGGRAPH)*, 21:205–214, 1987.

[51] D. Terzopoulos and K. Fleischer. "Deformable models," Visual Computer, 4:306–331, 1988.

[52] D. Terzopoulos and K. Fleischer. "Modeling inelastic deformation: Viscoelasticity, plasticity, fracture," In *Computer Graphics(Proc. SIGGRAPH)*, volume 22, pages 269–278. *ACM*, August 1988.

[53] A. Luciani, S. Jimenez, J. L. Florens, C. Cadoz, and O. Raoult. "Computational physics: a modeler simulator for animated physical objects," In *Eurographics'91*, Vienna, Austria, September 1991.

[54] G. Miller. "The motion dynamics of snakes and worms," *Computer Graphics*, 22(4):169–177, *Proceedings of SIGGRAPH' 88* (Atlanta, Georgia). August 1988.

[55] J. E. Chadwick, D. R. Haumann, and R. E. Parent. "Layered construction for deformable animated characters," *Computer Graphics*, 23(3):243–252, July 1989.

[56] D. Terzopoulos, J. Platt, and K. Fleisher. "Heating and melting deformable models (from goop to glop)," In *Graphics Interface'89*, pages 219–226, London, Ontario, June 1989.

[57] T. K. James, "The rendering equation," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*: *ACM*, 1986.

[58] M. B. Cline, "Rigid Body Simulation with Contact and Constraints," The University of British Columbia, 2002.

[59] S. R. Buss, "Accurate and efficient simulation of rigid-body rotations," *Journal of Computational Physics,* vol. 164, pp. 377-406, 2000.