Fast Fluid Simulation and its Applications

Xiaoyin Zhou

A thesis submitted

in partial fulfillment of the requirements for

the degree of master of philosophy at

The University of Hong Kong

April 2003

Abstract of thesis entitled

Fast Fluid Simulation and its Applications

submitted by Xiaoyin Zhou

for the degree of Master of Philosophy

at The University of Hong Kong in April 2003

Unlike solid, whose physical properties decide its fixed shape and stable behavior, fluid presents a tantalizing task for computer graphics researchers in that its shape is decided by the surrounding boundary and it is in constant movement. Fluid has this kind of special appearance because its molecular structure is quite different from other substances.

Realistic visualization of fluid in a 3D environment is very useful in many fields, such as computer art, advertising, education and entertainment. Since this issue was brought up, researchers have developed many methodologies to simulate fluid phenomena. There have been 2 major trends of water surface models for simulation: physics-based methods and non-physics-based methods. Physics-based methods concentrate on fluid's physical properties, and render it by solving equations governing the physical behavior of fluid. It is well-known that these methods rely heavily on accurate calculation of those equations, thus they are very complex and time-consuming. On the other hand, non-physics-based methods use existing mathematical model to approximate wave forms, without regards to the real physical property of fluid. To make the water look more real, these methods implement high-end computer graphics techniques to simulate the interactions between water

and air, but in the meantime, the rendering speed is also slow. As a result, they are not suitable for real-time simulation.

The method presented in this thesis aims at the visualization of fluid in engineering projects, in which real-time rendering is of first priority. It concentrates on the visual effect instead of physical properties. We get the inspiration from physics-based methods, in that we simplify the governing equations to a stable simulation model. We develop non-physics-related mathematical models to move the water surface in constant movement. While it needs less and simpler calculation, our approach simulates the fluidity well and fast. Moreover, we can run this real-time simulation on ordinary PC, which makes its applications more useful.

We have tested our model in a joint project with the Department of Civil Engineering, the University of Hong Kong. The result is quite satisfactory. It proves to be efficient in its applications in engineering projects with various fluid simulations, such as waste water, clean water and sludge, etc.

Contents

Chapter 1

Introduction

The development of PC has such a great impact on our lives, that many professional tasks which used to be carried out in the studios can now be performed at homes, with properly equipped software. One such example is that the SGI company, which is famous for its workstations for professional computer graphics, is now producing PC for home users to make their imagination come true on the computer screens. In the meantime, PC's capacities have been greatly upgraded to meet the requirements of modern people's ever increasing demand for a realistic virtual reality. Under this trend, computer graphics researchers have been working relentlessly to implement their innovations and ideas to utilize the hardware available to render almost real computer graphics scenes.

Nowadays, as the computer graphics' development has made it possible to simulate every aspect of our life, how to depict water and its interaction with the environment in a virtual reality environment becomes more and more important. Simulation of water is an intriguing task for computer graphics researchers not only because of water's constantly changing fluidity, which challenges the most ambitious researchers, but also because of water's role as an indispensable part of our life. In fact, in many visualizations projects, water has become the leading role. We cannot help noticing that what new breakthroughs in computer graphics have done to enrich our life with their vivid representation of the real world, which cannot be fulfilled without a realistic

simulation of water. This is very obvious in the entertainment industry. Many movies and games have set their backgrounds related to water and other fluid phenomena; furthermore, in some of them, fluid phenomena even become the leading role, such as the movie "Perfect Storm", in which the crew of a fishing boat has to fight many adverse weather conditions, such as heavy rain, tornado, huge surges, hurricanes, etc. In this film, what impress the spectators most is not the actor stars, but the gigantic surges formed by the hurricane, which are all attributed to the computer graphics artist's work. (Figure 1-1)

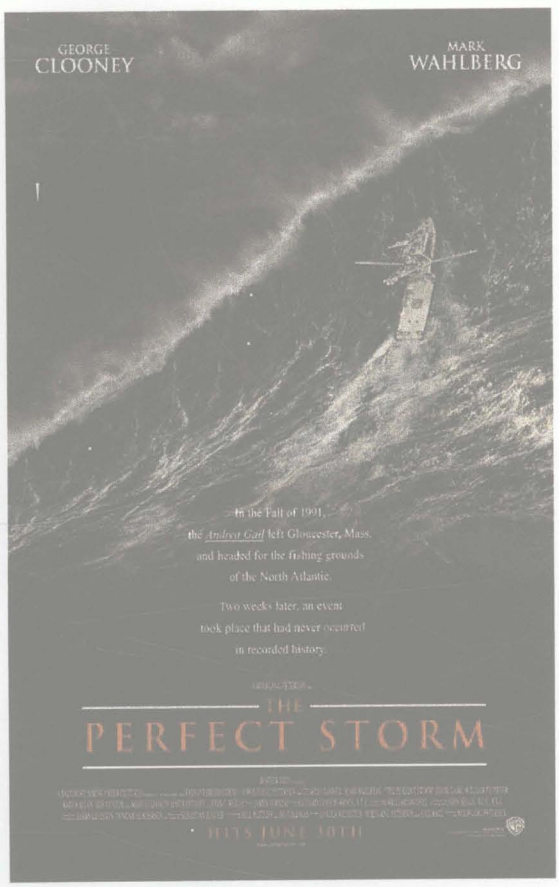


Figure 1-1: A film in which water plays the leading role

Apart from the entertainment industry, simulation of fluidity is also important in the engineering field. This kind of simulation has a major impact on the overall success of engineering projects, because those projects are to solve real life problems, while the use of computer graphics' visualization techniques will help engineers or end users to visualize the effects before the actual outcomes and to get better understanding of the

actual projects. With the help of modern visualization techniques, both hardware and software, engineers now can view their work on the computer even before it is actually built, which provides the end users with an intuitive understanding. As we mentioned above, fluidity has such a close relationship with human, that its interaction with us is also a part of most of the engineering projects.

However, fluid simulation in the engineering field differs from that in the entertainment field in that it calls for rendering speed. It does not require the final product to have 100% similarity to the real thing, but requires that it can reflect the whole process effectively. A single computer generated scene in the movies we see may take days to render, which is inconceivable in engineering projects. Engineers want their ideas to be reflected as quickly as possible, and they want to see the results immediately; the water here is only a supporting role.

Even the simulation of fluid will be somewhat simplified in engineering projects, the process still requires heavy computation duties and optimizations of algorithms. It is well-known that, unlike solid, whose physical properties decide its fixed shape and stable behavior, fluid presents a tantalizing task for computer graphics researchers in that its shape is decided by the surrounding boundary and it is in constant movement. Although researchers have developed many methodologies to animate fluid phenomena, they are mostly concentrating on its physical properties, and rendering water by solving equations governing the physical behavior of fluid. These methods rely heavily on accurate calculation of those equations, thus these methods are very complex and time-consuming. As a result, they are not suitable for real-time simulation. Our method will concentrate on the visual effect instead of physical properties. While it needs less and simpler calculation, our approach simulates the fluidity well and fast. Moreover, we can run this real-time simulation on an ordinary PC.

We have tested our methods in a Virtual Sewage Treatment System. project with the Civil Engineering Department. The results will be discussed in details in Chapter 4.

Chapter 2

Background

To visualize an object, we must know what it is made of, and what characteristics we want to simulate, what results we want to achieve, finally what methods shall be taken. In the following sections, we begin by a brief introduction of the physical properties of water. Based on this introduction, we will use virtual reality techniques to simulate the visual effect of water.

2.1 Physical properties of Water

Physical property is a characteristic of a substance that can be determined through one of the five senses without changing the basic nature of the substance. Water's physical properties are determined by its molecular structure [27]. One water molecule is composed of two different elements, oxygen and hydrogen. Oxygen, an atom with a negative electrical charge, is attracted to two hydrogen atoms that have positive electrical charges.(see **Figure 2-1**). The hydrogen atoms are "attached" to one side of the oxygen atom, resulting in a water molecule having a positive charge on the side where the hydrogen atoms are and a negative charge on the other side, where the oxygen atom is. Since opposite electrical charges attract each other, the side of a water

molecule with the hydrogen atoms (positive charge) attracts the oxygen side (negative charge) of a different water molecule, which makes water kind of "sticky." This information is important in the discussion of water's physical properties.

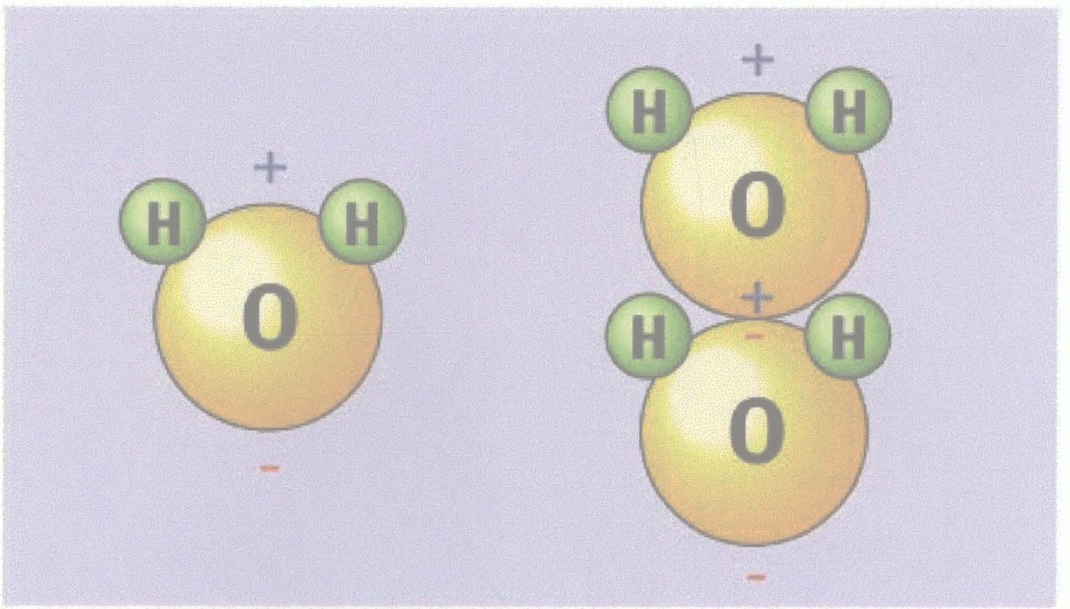


Figure 2-1: The molecular structure of water.

Other important and interesting water physical properties include:

- Water freezes at 32° Fahrenheit (F) and boils at 212° F (at sea level, but 186.4° at 14,000 feet). In fact, water's freezing and boiling points are the baseline with which temperature is measured: 0° on the Celsius scale is water's freezing point, and 100° is water's boiling point. Water is unusual in that its solid form, ice, is less dense than the liquid form, which explains why ice floats.

- Water is unique in that it is the only natural substance that is found in all three states -- liquid, solid (ice), and gas (steam) -- at the temperatures normally found on Earth. Earth's water is constantly interacting, changing, and in movement. Water's molecules arrange themselves in distinctly different patterns (**Figure 2-2**) in different states. The pattern taken by water when it is frozen causes its volume to expand and its density to decrease. Expansion of water in solid state allows ice to float on top of liquid water.

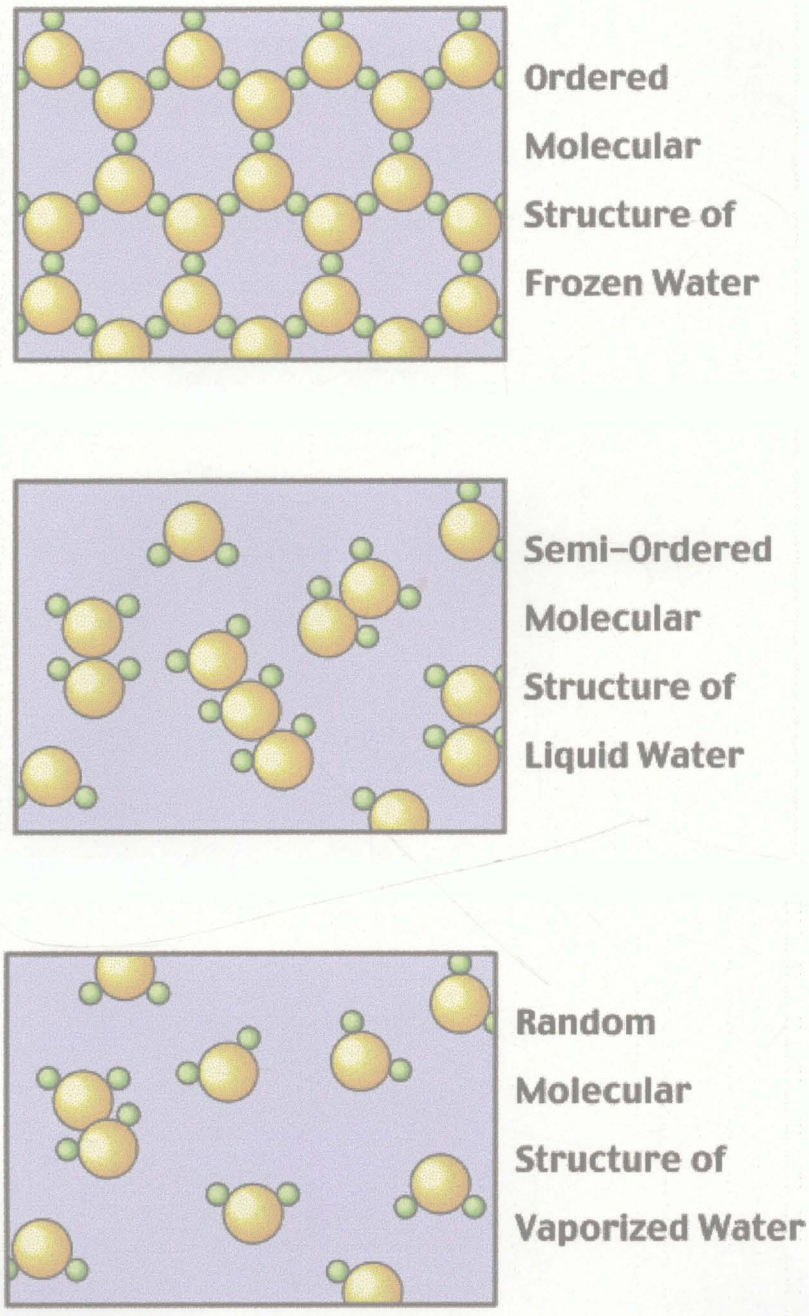


Figure 2-2: The three diagrams above illustrate the distinct patterns of molecular arrangement in water when it changes its physical state from ice to water to gas.

Figure 2.2 shows a slice through a mass of ice that is one molecule wide. In the liquid phase, water molecules arrange themselves into small groups of joined particles. The fact that these arrangements are small allows liquid water to move and flow. Water in the form of a gas is highly charged with energy. This high energy state causes the molecules to be always moving and thus reducing the likelihood of bonds between individual molecules from forming.

- Water has a high specific heat index. This means that water can absorb a lot of heat before it begins to get hot. This is why water is valuable to industries. The high specific heat index of water also helps regulate the rate at which air changes temperature, which is why the temperature change between seasons is gradual rather than sudden, especially near the oceans.

- Water has a very high surface tension. In other words, water is sticky and elastic, and tends to clump together in drops rather than spread out in a thin film. Surface tension is responsible for capillary action which allows water (and its dissolved substances) to move through the roots of plants and through the tiny blood vessels in our bodies. Figure 2-3 shows how the water molecules are attracted to each other to create high surface tension. This property can cause water to exist as an extensive thin film over solid surfaces. In the figure, the film is two layers of molecules thick.

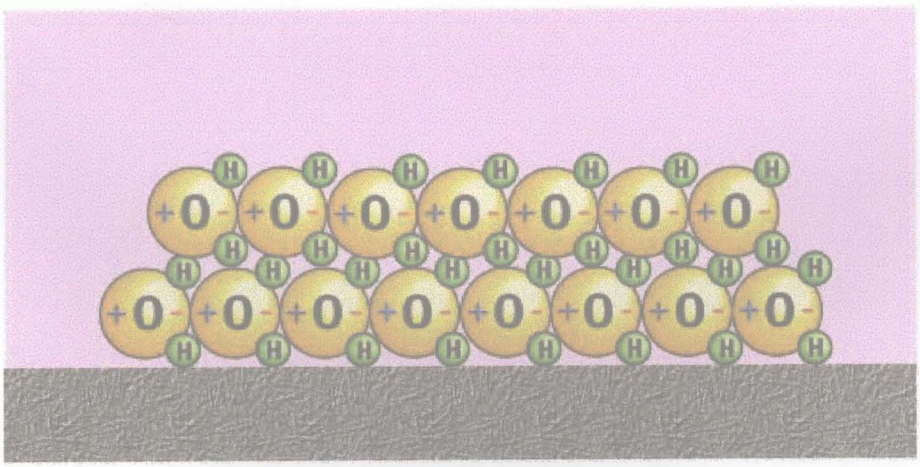


Figure 2-3: The film is two layers of molecules thick.

Among them, the high surface tension causes water to stick to the sides of vertical structures despite gravity's downward pull. Water's high surface tension allows for the formation of water droplets and waves, which gives it a glittering surface and constantly changing shape.

The following discussions are: "Virtual Reality", in which we will briefly introduce the idea of Virtual Reality – a useful method to visualize the world; "Different approaches", in which various approaches to visualize water's fluidity will be introduced for creating the virtual reality environment of water; "Our approaches", in which we will briefly outline the methods we adapted from previous works and introduced to produce high performance water simulation.

2.2 Virtual Reality

Virtual Reality systems create a cyberspace where it is possible to interact with anything and anyone on a virtual level. In these bizarre worlds, conventional laws of space and time need not hold – anything can be simulated, so long as it can be programmed.

Like most technologies, Virtual Reality did not suddenly appear. It emerged in the public domain after a considerable period of research and development in industrial, military and academic laboratories. Before it was introduced to the public, VR was best known for its implementation in the flight simulation. One such system is a training system where pilots can acquire flying skills without involving a real plane nor airports. In this system, a computer is used to create a perspective view of a 3D virtual world, and the view of this world is determined by the orientation of the aircraft. Later on, VR

is brought to the public by its implementations in various fields, but in these fields, the view of the world has been modified to be determined by the orientation of the user.

VR is based on the technologies of computer graphics, especially Real-time computer graphics. The domain of computer graphics is now well established and developed to such a sophistication, that the majority of text and imagery we come across in books, newspaper, television and films has been processed by computers to some degree. But what Real-time computer graphics differs from other computer graphics applications is that it requires the images be rendered by computer in several milliseconds. In other words, no matter how good the quality of the images, speed is of the essence.

Since the fluid model we are developing is to be implemented in VR environment, the rendering speed is of first priority over other factors. Some useful simplifications were taken to make the model work faster than most of the existing models. Its easy implementations have made the model suitable for different applications.

2.3 Different Approaches

As we mentioned earlier, how to bring out the ever-changing characteristic of water has been a tantalizing task for both computer graphics researchers and artists alike. Ever since this problem was put forward, there have been 2 major trends of developing suitable water surface models for simulation: physics-based methods and non-physics-based methods.

2.3.1 Physics-based methods

Over the years, there has been a consensus among researchers that the *Navier-Stokes* equations [1, 2, 3], the governing equation in fluid dynamics, are very good models for various computations of fluid flow, because the motion of a fluid at any point within a

flow is completely described by this set of nonlinear equations. Thus, the simulation of fluid should be based on the solution of these equations. They have the following basic forms of the following:

$$\rho\frac{Du}{Dt}=\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)+\rho g_x$$
$$\rho\frac{Dv}{Dt}=\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right)+\rho g_y$$
$$\rho\frac{Dw}{Dt}=\frac{\partial p}{\partial z}+\mu\left(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right)+\rho g_z$$

where

$$\frac{D}{Dt}=u\frac{\partial}{\partial x}+v\frac{\partial}{\partial y}+w\frac{\partial}{\partial z}+\frac{\partial}{\partial t},$$
$$u=\frac{dx}{dt},v=\frac{dy}{dt},w=\frac{dz}{dt}$$

ρ is the density, p is the pressure, μ is the viscosity, and g_x, g_y, g_z are the gravity vectors.

Although they may seem rather daunting at first glance, these equations have very humble origins. They are derived from Newton's Second Law which states that momentum is always conserved. The *Navier-Stokes* equations simply account for all momentum exchange possibilities within a fluid. Specifically, the terms on the left-hand side of the equations account for changes in velocity due to local fluid acceleration and convection. The right-hand terms take into account of acceleration due to the force of gravity (or any body force g), acceleration due to the local pressure gradient, ∇p, and drag due to the kinematical viscosity, μ, or thickness of the fluid. The *Navier-Stokes* equations cover all the necessary parameters to describe water, so computer graphics researchers have focused on how to solve the equations effectively, so that water's shape can be computed at any given time.

Thousands of books and articles have been published in various areas on how to solve these equations numerically, which have been the basics of Computational Fluid Dynamics (CFD) [1, 2, 3]. Which solver to use in practice depends largely on the computing power available; these methods involve large amount of calculation, which leads its limited use in real-time simulation. There is no easy way to solve *Navier-Stokes* equations. Until recently, there is a major breakthrough in this field: Chen et.al used a CFD model developed by them in [4] to solve the 2D *Navier-Stokes* equations via CFD method and map the resulting 2D surfaces into 3D and achieves realistic real-time fluid surface behaviors [5]. This method avoids calculation in 3D, thus greatly reduces the workload. By preserving some liquid's physical properties, it achieved a realistic simulation in a dynamic virtual environment. However, it is not stable when running for a long period of time, in that the mapping from 2D to 3D will accumulate the errors and magnify them as time lapses.

2.3.2 Non-physics-based methods

Back in the 80's, when the computational capacities of the computers were comparatively low, researchers had begun to work on the non-physics-based methods. As interaction between water and the obstacles is not the major concern of the project, the accurate simulation of fluid behavior does not seem to be important, in other words, even though the physics-based functions, such as the *Navier-Stokes* equations, will yield accurate results, the computation efforts on the solutions of these functions are not worthwhile. Researchers want something "like" water, not something "exactly the same as" water. Under the light of this, they have developed various mathematical models. Among them, suggested methods have included parametric functions [9] and sinusoidal phase functions [10, 11]. These methods are capable of simulating the appearance and behavior of ocean waves. Though they adopt different approaches to achieve remarkable results, one thing is in common: they do not expect any "physical" answer from their models, that is, the models which might be useless to an oceanographer

might turn out to be great for their work. Furthermore, the coefficients used to achieve the needed effects might have no actual physical meaning, or might be really bad for the actual physical model.

However, though the wave model can be set up and quickly computed, there is still some distance from the real thing. One notable problem is the reflection on the water surface. Without the interaction between light and water, the water surface will look like a fluctuating surface. As we mentioned before, it is water's unique surface tension that gives it the glittering outlook. Since the shape can be rendered quickly, researchers now lay their emphasis on the realistic effects. Effects such as caustic shading, reflection, refraction, and internal scattering have been addressed in detail to produce breathtaking scenarios [6, 7, 8].

One with some knowledge of computer graphics can easily recognize the obvious drawback of this method – the rendering speed. As it is known, these kinds of calculation are based mainly on one expensive rendering method – ray tracing. Although the effect is so realistic that it is widely used in entertainment industry, it always takes days to render one sequence of water flow, not to mention real-time simulation. Besides, once the parameters are set, one will wait for a long time to see the final outcome and make amendments. Hence, it still has no practical use in the engineering field.

2.4 Our approach

All the approaches mentioned above have rendered good effects; however, they seem to be rather unfit for engineering projects, which stress the real-time simulations. In physics-based methods, solving the *Navier-Stokes* equations does render accurate description of the water behavior, including speed, pressure, density, and so on. The data computed from those models are perfect for the study of hydraulics, in that the various parameters can be reckoned for any time given. However, the computation of

the solution is too complicated for real-time. In non-physics-based methods, even though one can develop the mathematical models which seem to be unrelated to water but turn out to be like water, the process of adding realistic light and water effect to the surface is rather expensive, in that it implements ray-tracing in order to render the glittering water surface. Neither of them can be real-time.

However, we cannot overlook the fact that in all the projects related to the methods described above, the major concern is water itself. For examples, most researchers use the physics-based methods to study the water properties; and mostly artists implement non-physics-based methods to render sequence of still pictures for entertainment business. But there are some engineering projects which emphasize the engineering details rather than the water inside it. In other words, researchers are more concerned with the container of the water than the water inside it. For example, the Sewage Treatment Simulation project that we are going to discuss in later chapters. Users want to address in details the technical specifications of the tanks of various treatment of sewage, including the movements of the different parts inside the tanks which are the essentials of the projects. In such occasions, researchers only want the water to come into interaction with those moving parts of the tanks, while water's own properties will not be their major concerns. Besides, in order to observe the tanks thoroughly and quickly, the overall rendering must be fast enough to catch up with the users' viewpoints, which is the ultimate goal of real-time rendering.

Under the hood of these requirements, we have developed our models. We sacrifice physical accuracy for visual appearance. This method can enable users to achieve fluid-like behavior and fine-tune the simulation in real-time. This method is cost-effective and stable in that it uses fixed texture to render animations; its mathematical model is simple; and it uses environmental mapping to simulate reflection of the water surface. Since it involves no solution of complex equations, no advanced mathematical models and no ray tracing, this approach can fulfill the needs for real-time animation.

We get the inspiration from physics-based methods, in that we adapt the CFD to set up the scene. We develop non-physics-related mathematical models to move the water surface in constant movement. Though the parameters seem to be physics-related, they do not carry any actually physical meanings. In fact, changes of such parameters may bring undesirable effects.

Chapter 3

Fluid Visualization Model

Since our goal is to develop a simple and less complex fluid model, most of the technical details related to real water are supposed to be simplified. Kass and Miller's simulation model [12] provides us with a modest example to start with. Although this model is very simple, it has preserved many useful properties of fluid, such as the propagation of waves. Moreover, it is very stable and cost-effective. We further improved this model and combined it with an innovative texture mapping technique to construct a fluid simulation model suitable for various applications.

3.1 Simplified shallow water waves model

Kass and Miller used a simplified set of equations to simulate ocean waves [12]. Those equations are a simulating set of the full *Navier-Stokes* equations, but in a rather simplified way. The simplification arises from three approximations. The first approximation is that the water surface is a height field. The second assumption is that the vertical component of the velocity of the water particles can be ignored. The third assumption is that the horizontal component of the velocity of the water in a vertical column is approximately constant [15,16,17]. Though their limitations are quite

obvious, they are very useful simplifications.

For simplicity, we begin with a height-field curve in two dimensions. Later, the same techniques will be extended to a height-field surface in three dimensions. Let *h(x)* be the height of the water surface and let *b(x) be* the height of the ground, If *d(x)=h(x)-b(x)* is the water depth and *u(x)* is the horizontal velocity of a vertical column of water, the shallow water equations that follow from the assumptions [16; 17]can be written as follows:

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+G\frac{\partial h}{\partial x}=0 \qquad \text{Equation 3-1}$$

$$\frac{\partial d}{\partial t}+\frac{\partial}{\partial x}(ud)=0 \qquad \text{Equation 3-2}$$

where G is the gravitational acceleration. Equation 3-1 expresses Newton's law *F=ma* while Equation 3-2 expresses the constraint of volume conservation. Note that the resulting differential equations are non-linear. A further simplification which is often used is to ignore the second term in Equation 3-1 and linearize around a constant value of h. This will be reasonable if the fluid velocity is small and the depth is slowly varying. The resulting equations are then:

$$\frac{\partial u}{\partial t}+G\frac{\partial h}{\partial x}=0 \qquad \text{Equation 3-3}$$

$$\frac{\partial h}{\partial t}+d\frac{\partial u}{\partial x}=0 \qquad \text{Equation 3-4}$$

where *G* is gravity (and other global forces), *h* is the height of the water surface, *d* is the depth, and *u* is the horizontal velocity of a vertical column of water. We can also combine these two equations. Start by differentiate Equation 3-3 with respect to x and Equation 3-4 with respect to *t:*

$$\frac{\partial^2 u}{\partial t \partial x} + G\frac{\partial^2 h}{\partial x^2} = 0$$ Equation 3-5

$$\frac{\partial^2 h}{\partial^2 t} + d\frac{\partial^2 u}{\partial x \partial t} = 0$$ Equation 3-6

Now substituting the partial cross-derivative of Equation 3-4 into Equation 3-3 we end up with:

$$\frac{\partial^2 h}{\partial^2 t} = Gd\frac{\partial^2 h}{\partial x^2}$$ Equation 3-7

Using finite-differences we can discretize this as:

$$\frac{\partial^2 h}{\partial^2 t} = -G(\frac{d_{i-1} + d_i}{2(\Delta x)^2})(h_i - h_{i-1}) + G(\frac{d_i + d_{i+1}}{2(\Delta x)^2})(h_{i+1} - h_i)$$

Equation 3-8

Now that we have turned the partial-differential equation into a second order ordinary differential equation and we will solve it using a first-order implicit method. We will use finite-differences to discretize the first- and second order time-derivatives of h, let

h_i denote h at the ith iteration and let dots denote differentiation with time, then the first-order implicit equations can be written as:

$$\frac{h_i - h_{i-1}}{\Delta t} = \dot{h}_i$$ Equation 3-9

$$\frac{\dot{h}_i - \dot{h}_{i-1}}{\Delta t} = \ddot{h}_i$$ Equation 3-10

Since we are solving for h_i, we will rearrange Equation 3-9 and substitute it into Equation 3-10:

$$h_i = h_{i-1} + \Delta t\, \dot{h}_i + (\Delta t)^2\, \ddot{h}_i \quad \text{Equation 3-11}$$

$$\Updownarrow$$

$$h_i = 2h_{i-1} - h_{i-2} + (\Delta t)^2\, \ddot{h}_i \quad \text{Equation 3-12}$$

And substituting Equation 3-6 into this we get:

$$\begin{aligned} h_i &= 2h_{i-1} - h_{i-2} \\ &\quad - G(\Delta t)^2 \left(\frac{d_{i-1} + d_i}{2(\Delta x)^2}\right)(h_i - h_{i-1}) \\ &\quad + G(\Delta t)^2 \left(\frac{d_i + d_{i+1}}{2(\Delta x)^2}\right)(h_{i+1} - h_i) \end{aligned} \quad \text{Equation 3-13}$$

To this step, we are still left with non-linear equations because d depends on h. The last discretisation done is to treat the depth d as constant during iteration, so one ends up with the following linear system:

$$\mathrm{A}h_i = 2h_{i-1} - h_{i-2} \quad \text{Equation 3-14}$$

Where A is given by:

$$\mathrm{A} = \begin{pmatrix} e_0 & f_0 & & & & & \\ f_0 & e_1 & f_1 & & & & \\ & f_1 & e_2 & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & e_{i-3} & f_{i-3} & \\ & & & & f_{i-3} & e_{i-2} & f_{i-2} \\ & & & & & f_{i-2} & e_{i-1} \end{pmatrix} \quad \text{Equation 3-15}$$

and the elements of A are as follows:

$$e_0 = 1 + G(\Delta t)^2 (\frac{d_0 + d_1}{2(\Delta x)^2})$$

$$e_i = 1 + G(\Delta t)^2 (\frac{d_{n-1} + 2d_n + d_{n+1}}{2(\Delta x)^2}), n \in (1, i-2)$$

$$e_{i-1} = 1 + G(\Delta t)^2 (\frac{d_{i-2} + d_{i-1}}{2(\Delta x)^2})$$

$$f_i = -G(\Delta t)^2 (\frac{d_i + d_{i+1}}{2(\Delta x)^2})$$

Now this matrix gives a symmetric tridiagonal linear system, which can be solved relatively fast, see [13] for more info. Expanding Equation 3-7 to 3D is done by substituting the partial derivative of h with respect to x with the *Laplacian*:

$$\frac{\partial^2 h}{\partial^2 t} = Gd\nabla^2 h$$

$$\Updownarrow$$

$$\frac{\partial^2 h}{\partial^2 t} = Gd(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2})$$

Equation 3-16

And it's solved exactly as the 2D case simply by splitting it up into two systems - one dependent on x and one on y.

3.2 Surface Waves

Now we are about to come to the basic equations for our simplified water model. If we constrain the water to a fixed depth, Equation 3-16 reduces to:

$$\frac{\partial^2 h}{\partial^2 t} = |V|^2 (\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2})$$ Equation 3-17

where |V| is the velocity of the wave (across the surface). Let $h_{x,y}^t$ be the height of the grid at position x and y at time t, then Equation 3-17 can be discretisised using central differences as [14]:

$$\frac{h_{x,y}^{t+1} - 2h_{x,y}^t + h_{x,y}^{t-1}}{(\Delta t)^2} = |V|^2 (\frac{h_{x+1,y}^t + h_{x-1,y}^t + h_{x,y+1}^t + h_{x,y-1}^t - 4h_{x,y}^t}{\Delta h^2})$$

Equation 3-18

Rearranging the above for t+1:

$$h_{x,y}^{t+1} = \frac{|V|^2 (\Delta t)^2}{\Delta h^2} (h_{x,y}^t + h_{x-1,y}^t + h_{x,y+1}^t + h_{x,y-1}^t) + (2 - \frac{4|V|^2 (\Delta t)^2}{\Delta h^2}) h_{x,y}^t - h_{x,y}^{t-1}$$

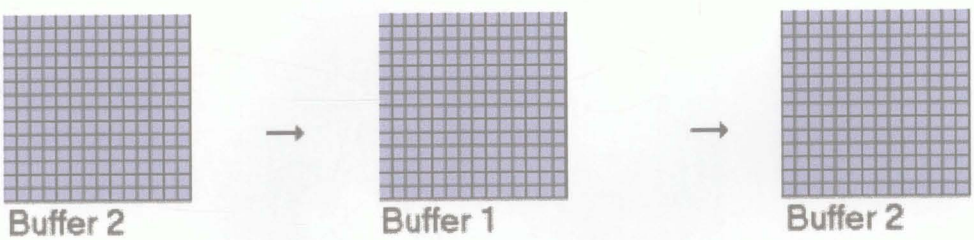
Equation 3-19

The equation above has been widely used as a filter in games for generating water ripples. As shown in great detail in [14] this can be animated with just a few arithmetic operations per grid-element. To explain how this algorithm achieve such an effect, let us take a look at the following sections which will illustrate how the filter, with the help of 2 arrays, causes the ripples to propagate outwards from any disturbances made in the arrays.

To achieve a satisfying effect, we only need 2 arrays. These arrays will hold the state of the water. One holds the current state, the other holds the state from the previous frame. It is important to have 2 arrays, since we need to know how the water has changed since the last frame, and the frame before that, as described in Equation 3-19.

Data from the previous frame (Buffer1) and the frame before that (Buffer2) is used together, and the result is written into Buffer2.

Buffer1 contains the current state of the water.



The pseudo-code is as follows:

damping = some non-integer between 0 and 1, a factor which is to cause the ripple to diminish. The best value is between 0.9 – 1.0, otherwise the ripple will damp very quickly.

```
begin loop
        for every non-edge element:
        loop
            Buffer2(x, y) = (Buffer1(x-1,y)+
                             Buffer1(x+1,y)+
                             Buffer1(x,y+1)+
                             Buffer1(x,y-1)) / 2 - Buffer2(x,y)
            Buffer2(x,y) = Buffer2(x,y) * damping
        end loop
        Display Buffer2
        Swap the buffers
  end loop
```

3.3 Some improvements

One big defect with this fast algorithm is that the ripples are not quite circular. It is because of the fact that the original smooth factor is only calculated in a 4-direction base. On a machine fast enough, a smooth factor of 8 directions will perform much better.

Smoothed(x,y) = (Buffer1(x-1, y-1)+ Buffer1(x-1, y+1)+ Buffer1(x+1, y+1)+
Buffer1(x+1, y+1)+ Buffer1(x-1, y) + Buffer1(x+1, y) +
Buffer1(x, y-1) + Buffer1(x, y+1)) / 8

Secondly, since the climaxes of the waves are always caused by the disturbances, the climaxes will appear rather outstanding among the waves, in that they will form acmes among the waves.

Inspired by the Gaussian function which has been widely used in the Image Processing, we adapt this filter in our method to smooth the effect of the disturbances.

Gaussian function [19] has the form of

$$g(x, y) = ce^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

where c is the scale factor, σ is the spread parameter. The function decreases smoothly to zero with increasing distance from the origin, meaning that image values nearer the central location are more important than values that are more remote.

Similar to the Gaussian filter in Image Processing [18], in our approach, for each of the disturbance produced, we apply the Gaussian Functions as follows:

```
sigma =  Scale Filter;
for(i=-2;i<=2;i++){
```

```
        for(j=-2;j<=2;j++)
        {
            dist2 = double (i*i+j*j);
            Buffer[x+i][y+j] = disturbance*(float)exp(-dist2/(sigma));
        }
}
```

3.4 Control of the model

As shown in the above sections, the water model is comparatively easy to control, since it only has several essential parameters which decide the shape, speed and appearance of the water. There are no fixed parameters to use in the model, as long as one user finds the water suitable for his purpose. So basically the model is able to be fine-tuned to fit every user's needs by adjusting the following parameters below:

1. Grids

The grid resolution directly influences the overall performance of the water model, because the resolution decides the computation workload, which, in turn, will influence the rendering speed. However, it is well-known that the higher resolution of the grids, the more detailed simulation will be. Hence the resolution of the grid is to be chosen according to the actual needs.

The recommended grid resolution is 200*200 in a square pool. Exceeding this will result in obvious delay in the rendering, according to our test report.

Furthermore, the shape of the grid also decides the shape of the waves. If the grids are just squares, the ripples will appear circular. Similarly, if the grids are adjusted to rectangles, the ripples will then propagate like ellipses.

2. *Height of the wave*

Actually the height of the waves is decided by the disturbances added. As we discussed before, the basic model of the water has the ability to propagate any disturbance outwards, thus forms the ripples. When ripples are combined, the water model will have a wavy surface. If the disturbances are too violent at a time, it will take longer time for them to diminish, which will be adverse for other ripples' propagations. This will result in the waves all being squeezed to some intense concentrations of the disturbances.

So the height of the wave will have a dominant effect on the waves. The recommended value is 0.0—1.0, which is enough to create the small ripples formed by a breeze.

3. *Frequency*

The frequency here does not mean the frequency we encounter in the waves theories. It is the frequency of the disturbances. The water surface will be crowded with disturbances if the frequencies are high. When this happens, as one could imagine, the ripples do not have enough time to propagate the disturbance before another disturbance hits.

If the frequency is set too high, it will cause some similar adverse results as the case with high disturbance. So, basically, the frequency and the disturbance must be taken into account together.

4. *Damping*

The wave is traveling outwards while losing the energy. If the energy remains the same, the wave will continue traveling and bouncing back and forth in the pool.

How to choose a suitable damping value depends on the physical properties of the fluid which the user wants to simulate.

For pure water, the damping value is around 0.95 to 0.98, because it does not contain much foreign substance other than water. Similarly, the more foreign substance the fluid has, the smaller value the damping value is. Users can choose the value best representing the physical properties of the fluid.

5. *Smoothing scale factor*

The sigma value is introduced as the smooth scale factor, which we have gone through in the last section. It has the direct relation with the smoothing filter – Gaussian filter, which is implemented to smooth the effect of disturbance. And by the use of the filter, the wave surface will be greatly improved. The scale factor is the decisive value in the smoothing filter, in which it decides the influenced scale, and how much each of the influenced cell's weight in the whole filter.

The suggested value under a grid of 200*200, wave height 0.5, disturbance frequency 50 HZ and damping value 0.98 is 1.0, which is enough to create a placid water surface with occasional ripples.

Through the proper combination of the 5 values mentioned above, users will be able to come up with a quite satisfying model for their purposes. Now the complex water simulation has been turned into a balanced control of those 5 factors. Please note that though this method is by no means a highly accurate simulation of rippling water (though it is not too far off), its robust controllability is greatly over others, in that one can control the waves generation and propagation just by adjusting those 5 values in a small range.

3.5 Rendering the Water

Though we have the wavy surface, it will just look like a fluctuation of polygons without rendering. We will render refraction and reflection of water. With the help of texture mapping, the effect will be greatly enhanced.

3.5.1 Refraction

As [14] has suggested, refraction is easy to achieve using the height difference between waves. With a texture to go behind the water, the refraction can be done by this.

```
for every pixel (x,y) in the buffer

   Xoffset = buffer(x-1, y) - buffer(x+1, y)

   Yoffset = buffer(x, y-1) - buffer(x, y+1)

   Shading = Xoffset

   t = texture(x+Xoffset, y+Yoffset)

   p = t + Shading

   plot pixel at (x,y) with colour p

end loop
```

But the refraction effect has not much actual use because if the refraction can be seen clearly, the water is supposed to be very clear, which is not common in the reality.

3.5.2 Reflection

As most computer artists' favorite rendering technique, ray tracing has been the most common tool for realistic scenes. It is the same case in rendering of the water. Without ray-tracing, no matter how brilliant the texture is, or how accurate the wave form is, the effect will be greatly compromised. However, since we are dealing with real-time rendering, this technique surely does not apply to our approach. That is why we chose Environment Mapping as an alternative to ray tracing.

3.5.2.1 What is Environment Mapping

First brought up by Blinn and Newell in 1976 [20], environment mapping was introduced to the public as reflection mapping. In 1984, Miller and Huffman [21] suggested that environment mapping can also create diffuse as well as specular objects. Since then, environment mapping has developed into a useful tool to create stunning effects. Among various effects it has created, the most popular one is the Advanced Robot type T-1000, in the film *Terminator 2- Judgment Day*. The stunning effect impressed everybody in 1991.

Ray tracing can capture reflective environment but is too slow, while environment mapping, which captures reflections from every direction with the help of Texture Mapping, can render similar effects. The major difference is that in ray tracing, the reflective surface is computed by intersection of various rays, but in environment mapping, the surface is solely decided by the texture used.

The basic idea of environment mapping is illustrated in the Figure 3-3, considering the light from all directions reflected from the chrome sphere as seen by an infinite viewer:

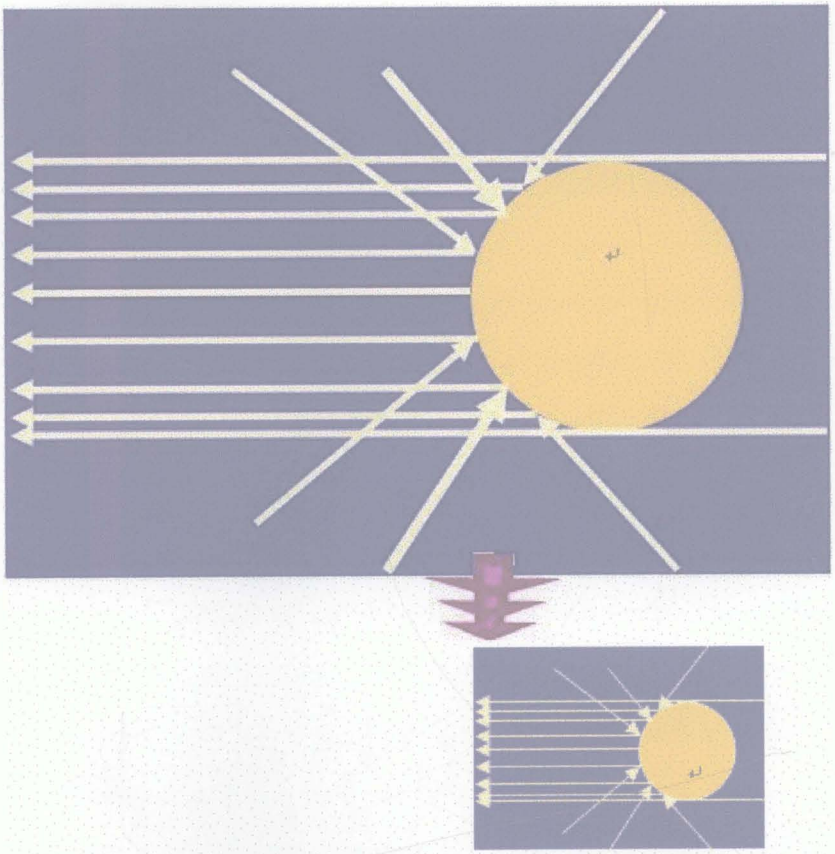


Figure 3-3: Illustration of the basic idea of environmental mapping.

Imagine if we shrink the sphere to an infinitely small point, the environment will be captured in all the directions. If applied on the curve surface, the effect will be quite stunning.

Environment mapping is an efficient technique to compute reflections in computer-synthesized environments without tracing a myriad of secondary rays. By using pre-defined images cast as a geometric map shape that surrounds an object, reflections may be defined for a single point in space. Environment mapping samples a texture image from the environment map based on the reflected site vector for each point on the object. By projecting the 3D environment onto a 2D environment map that

surrounds an object, reflections can be created with some degree of accuracy. Environment mapping allows the number of computations needed to simulate reflections to be greatly reduced while producing results that maintain a high degree of image quality [22].

If we look closely at the wave form, we would notice that the surface is made up of various curve surfaces. Besides, from the viewpoint of physics, if taken out separately, water drops will form a sphere (if without gravity) because of surface tensions. Since those water drops are the interior causes of the glittering outlook of water surface, we link the sphere mapping with water surface rendering.

3.5.2.2 Implementation of environment mapping – sphere-mapping

The most widely and accepted way of environment mapping is the sphere-mapping, which provides an interesting way of mapping a single-image texture onto the surface of an object that allows for view-dependent effects such as reflections, and even outlining and other non-photorealistic effects. More specifically, sphere-mapping maps all "reflected" vector directions into a single image (actually into the largest circle that will fit in the texture image). The problem then becomes determining the relationship between the texture coordinates (s,t), the texture-map, the reflection vector, the eye, and a particular vertex for which we want to find the texture coordinates.

In order to achieve the sphere-mapping, we need:

- Normalized viewing direction u from the eye to the vertex in eye-space. The eye is at the origin in eye-space, so u is simply the vector to the vertex in eye-space

- Vertex normal n in eye-space. In OpenGL, the user specifies the object-space normal with glNormal3f, so the eye-space normal is the object-space normal transformed into eye-space with the MODELVIEW matrix. This step is done

internally in OpenGL. The user only need to provide the proper object-space vertex normals.

- A sphere-map texture that is a rectangular texture, but only the centered largest circular region is used

We wish to find the following:

- reflected direction **r**
- final (s,t) texture coordinates

In order to understand how we get s and t from the reflection vector, we must first understand how we get from the s and t values on the sphere-map to a normal, and then from a normal to a reflection vector. We can then invert this operation to derive the texture coordinates.

Imagine that the sphere-mapping is an orthographic projection of a unit sphere that is centered at the origin. The (s,t) values within the circular projection of the sphere are within [0,1], so for given (s,t) values we can obtain the (x,y) sphere coordinates in [-1,1] as follows through a simple scaling and translation:

$x = 2*s - 1$

$y = 2*t - 1$

Since we know we are looking at the orthographic (or parallel) projection of the unit sphere and we have the (x,y) coordinates of a point on the sphere, we can compute the z coordinate from the implicit sphere equation:

$x^2 + y^2 + z^2 = 1$ // implicit equation for a unit-sphere at the origin

$z = \text{sqrt}\,(1-x^2-y^2)$

Now we have a mapping from the texture coordinates (s,t) to the points on the sphere in the sphere-map. This now directly relates the (s,t) values to the normals on the

sphere by noticing that the normal of a sphere at a point p on the surface is simply p minus the sphere's origin. We are dealing with a unit sphere centered at the origin, so the normal is simply the (x,y,z) point we computed:

Normal $n = (x,y,z) = (2s-1, 2t-1, sqrt(1-x^2-y^2))$

Now since the sphere-map is assumed to be an orthographic projection of a sphere, the viewing direction u is simply down the negative z-axis. So we can easily compute the reflection vector in terms of the normal and then in terms of s and t (see Figure 3-4):

$u = (0,0,-1)$

$\mathbf{n} = (2s-1, 2t-1, sqrt(1-x^2-y^2))$

$\mathbf{r} = \mathbf{u} - 2(\mathbf{u}.\mathbf{n})\mathbf{n}$

$= (0,0,-1) - 2*-sqrt(1-x^2-y^2)*(2s-1,2t-1,sqrt(1-x^2-y^2))$

$= (0,0,-1) - 2*-nz*n$

Now we have the reflection vector r in terms of the (s,t) tex-coords. Now if we work back towards a formulation of s and t in terms of the reflection vector, thus deriving the original sphere-mapping equations.

First we will invert the last equation and obtain n in terms of r:

$n = (rx,ry,rz+1) / (2*nz)$

Hence, we see that the normal is proportional to (rx,ry,rz+1) (meaning that it is the same direction, but not the same length) scaled by the value 1/(2*nz). We can ignore this scalar factor and simply normalize (rx,ry,rz+1) by its magnitude:

$n = (rx,ry,rz+1) / sqrt(rx^2 + ry^2 + (rz+1)^2)$

Now we obtain a definition of n in terms of the reflection vector components. We can continue to work backwards through our previous derivation to obtain s and t:

$nx = rx / sqrt(rx^2 + ry^2 + (rz+1)^2)$

$ny = ry / sqrt(rx^2 + ry^2 + (rz+1)^2)$

Using definition of nx and ny from before, and then solving for s and t:

$nx = 2*s - 1$

$ny = 2*t - 1$

$2*s - 1 = rx / sqrt(rx^2 + ry^2 + (rz+1)^2)$

$2*t - 1 = ry / sqrt(rx^2 + ry^2 + (rz+1)^2)$

$s = rx/(2*sqrt(rx^2 + ry^2 + (rz+1)^2)) + 1/2$

$t = ry/(2*sqrt(rx^2 + ry^2 + (rz+1)^2)) + 1/2$

$m = (2*sqrt(rx^2 + ry^2 + (rz+1)^2))$

$s = rx/m + 1/2$

$t = ry/m + 1/2$

Now we need to find out the reflection vector r:

$r = u - 2(u.n)n$

That is the reflection vector equals the viewing direction u minus 2 times the normal n scaled by the length of the projection of u onto n (or the cosine of the angle between u and n). This simply means that r is the reflection of u about the plane through the vertex with normal n (the standard reflection vector derived from the rule that the angle of incidence equals the angle of reflection, but it is hitting a plane that goes through the vertex and is perpendicular to the normal n).

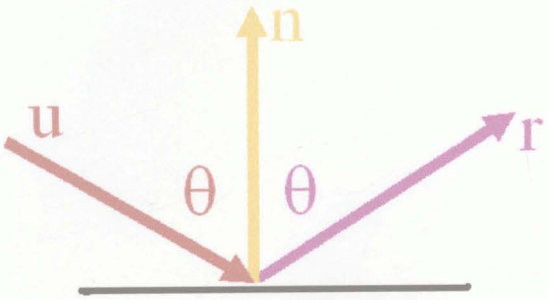


Figure 3-4: Computing the (s,t) tex-coords based on the reflection vector r:

The goal is to map an arbitrary 3D direction r into a 2D circle. The (s,t) coordinates access the 2D bounding rectangle of the circle; the lower-left corner is (0,0), the top-right is (1,1), and the center of the circle is at (0.5,0.5) with a radius of 0.5. As a result, the sphere-map should actually be a square to avoid any unnecessary waste (except for the corners).

$$m = 2 * \text{sqrt}(rx^2 + ry^2 + (rz+1)^2)$$

$$s = rx/m + 1/2$$
$$t = ry/m + 1/2$$

Figure 3-5 and Figure 3-6 give a brief comparison between those 2 types of texture mappings on the water rendering.

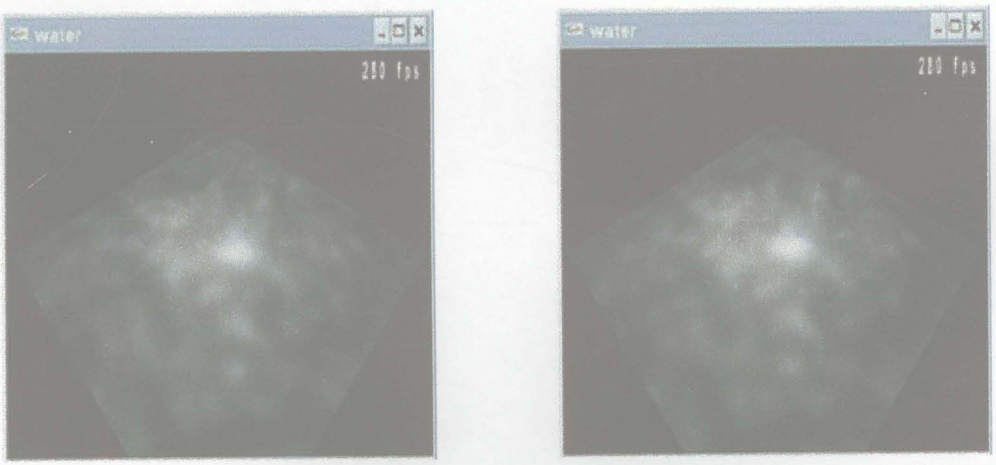


Figure 3-5: Water surface with ordinary texture mapping

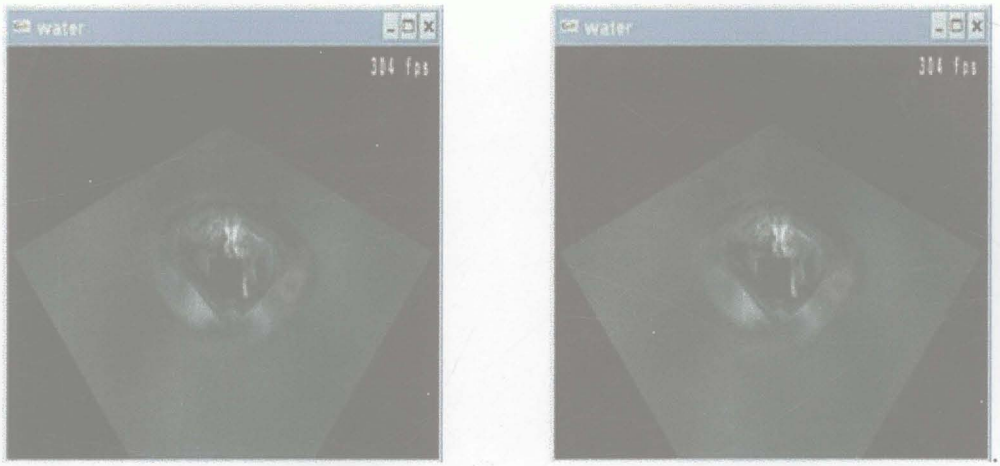


Figure 3-6: Water surface with Environmental Mapping

3.6 Fluid flows

The flow of the water has not been mentioned in the above sections. In fact, the water in discussion cannot look like flowing without the method we introduce next.

Unlike most other methods, our method will use some features of the texture mapping to render the flow. Texture mapping is very useful in computer graphics applications, because of its simple implementations. Its appearance greatly relieved computer graphics programmer' workload on modeling 3D object' details geometrically. Through the proper use of texture, many realistic effects can be done with great ease, such as the environmental mapping we mentioned in last section.

Texture mapping is so popular not only because of its capability of giving convincing visual details to a surface without actually changing the surface details, but also because of its easy manipulation. Just by specifying the texture coordinates for all the vertices of a polygonal mesh, the texture is ready. Besides, texture mapping can be applied via simple transformation matrixces, and thus, some simple animations, such

as translation, rotation, scaling, can be easily achieved

Translating the texture in the scenes may give human perception system the wrong impression that the object is actually moving. Thus, we can implement the translation of texture on the fluid surface to make it feel like flowing. This method can save a lot of troubles of moving the objects, such as calculating the horizontal velocity of the waves.

The control of the flow is comparatively isolated from the other controls of the wave generations or the rendering. There are 2 controls on the surface flow, the horizontal and vertical speeds, which are actually the translational speeds of the texture. These speeds do not have to be in accordance with each other, In other words, they can be adjusted separately, as long as the user feels comfortable about it.

Some will ask water's horizontal speed is not uni-directional, i.e. it cannot flow to left or right alone, and instead, it will flow in both directions. Here, we will implement multiple textures on the same surface. By adjusting the alpha value of the textures, the textures cheat the human perception again – they will look like one texture. Then the textures' speed will be adjusted separately to let the fluid flow in one vertical direction while rushing to both horizontal directions.

3.7 Conclusion

The fluid simulation engine we developed has implemented the following techniques to make it suitable for real-time water simulation.

1. Wave generation

On the base of the shallow wave model developed by Kass and Miller [14], we further improve the wave forms by smooth factor and propagation equation. 5

parameters have been introduced to control the water waves. Though these 5 parameters do not have actual physical meanings as their name imply, they are to be adjusted in accordance with each other to give users satisfactory results.

2. Refraction

We observe that under most situations, refraction is not obvious. Thus, we do not take refractioninto account in our model. The user's feedbacks also support this view.

3. Reflection

This is the major rendering technique for the water surface. With the help of environmental mapping, we are able to simulate the feature of water surface – reflective, ever changing. The glittering outlook of the surface gives the users persuasive impressions [22].

4. Flow

The water flow is achieved by controlling the translational speeds of the texture mapping. Since texture mapping can be applied via transformation matrices to achieve simple translations, they can be harnessed to render animations on the object surface. Its control is separate from the rest of the controls of fluid model.

3.8 An extension of the fluid simulation model -- Sludge Simulation

As mentioned above, our fluid simulation method can be adjusted to fulfill different purposes. The sludge model developed for the Virtual Sewage Treatment System is a good example.

Sludge is a rather new topic for most of the computer graphics artists, probably because of its repelling appearance and the associated stink. Yet it is quite inevitable in some engineering projects such as the sewage system, in which the sludge is the sediment of the waste. Its quality reflects the efficiency of the system.

There are not many papers concerning sludge simulation, not to mention the existing simulation models. However, we could not overlook the fact that sludge is also a fluid phenomenon, which means that our fluid simulation method could be applied to it, if the parameters are set correctly.

Motivated by this, we started to establish a similarity link between the sludge and the water model. Since our model can actually be implemented to simulate almost every kind of fluid, provided that the parameters are set correctly, once the link is established, we can turn the existing fluid model into a sludge model. In the following sections, we will discuss how to achieve a satisfactory outlook of sludge by adjusting the 5 parameters in accordance with the physical properties of sludge and illustrate how the fluid model can fit in various fluid phenomena.

Grids of the whole scene are not only considered the essential factors of the overall performance, but also the factors of the shape of the waves generated. Even though the waves are not obvious in the sludge, they are still the causes of the fluctuating surface. However, in this case, since the waves should appear rather smooth, no much detail is needed. Hence, it is not necessary to create a fluctuating surface made up of too many small grids. The tests show that for general purpose, a surface made up of 50*50 grids is quite enough.

Sludge is placid, with occasional low disturbances. Thus the heights of the waves of the sludge model are set to a comparatively low value, so is the frequency of the disturbances due to the same reason.

Sludge is a sticky and thick fluid with comparatively higher density, which means the damping value will be set to smaller value, so that the waves generated by the disturbance will not travel very far and instead, they tend to faint away very quickly.

The scale factor will mainly affect the wave forms by adjusting the disturbance's influence range. This range should be large enough to influence the waves as many as possible.

The rendering of the sludge model is simpler than the water model discussed. It does not have refraction or much reflection. In this case, the sludge model does not need the technique we used on the water model. A simple lighting effect decided by the normal vector of the grids is sufficient to render the visual effect of sludge. The result will be shown in next chapter.

So far we have finished the brief description of how the sludge model can be derived from the fluid model we developed, just by adjusting the 5 parameters. Similarly, the fluid model can be expanded to accommodate various purposes.

In the next chapter, we will give a detailed description on the implementation of the fluid model – Virtual Sewage Treatment System.

Chapter 4

Virtual Sewage Treatment System

The fluid simulation engine is implemented in a joint project described in this chapter. According to various feedbacks from the users, the engine has achieved its goal – a fast and realistic simulation which can be run on an ordinary PC.

4.1 Introduction

As one of the dominant design and visualization software for engineering purpose, AutoCAD has gained its wide popularity through its user-friendly interface and powerful drawing system, which permit user to construct or edit a drawing on a graphics display screen. Although it provides some basic 3D features (e.g. lighting effect rendering, transformation of the 2D drawing into 3D models), and viewing tools (e.g. rotation, translation) for viewing the model from different angles, those features are too limited for 3D visualization. Nowadays, engineers are not merely satisfied with viewing the model; they would like to manipulate the model developed, or to actually feel the model, which is called Virtual Reality.

Under this trend, we have recently made some interesting attempts. We implemented some advanced computer graphics techniques on the 3D model developed in AutoCAD by the Department of Civil Engineering of the University of Hong Kong. After adding our Virtual Reality elements, the model is turned into a comprehensive demonstration system which provides teachers with a powerful teaching aid in demonstrating the structure, functions and features of the model. While reflecting the original model with 100% loyalty, the system has some extra features to capture every user's attention right at the spot.

It all starts with the model, a sewage treatment facility, in AutoCAD file format. This system will convert the AutoCAD model developed by the Department of Civil Engineering into a Virtual Reality environment in computer graphics. Furthermore, with a user-friendly walk through mode provided, the user can easily travel through the scene and get to the destination position; by the simulation of the water flow, the user can identify various stages of wastewater treatment.

We hope this system will present an intriguing 3D environment with detailed, real demonstration of the removal of the waste or pollutants from industry effluent and give users, mostly students, perceptual knowledge of the water treatment facilities.

4.2 Sewage Treatment Facility

Wastewater is the after-use effluent of the water supply of a community after it has been fouled by a variety of uses. It contains a variety of substances such as organic matter, toxic materials, pathogenic microorganisms and nutrients, which would cause serious pollution in natural receiving waters if discharged without treatment. Therefore, proper treatment and disposal of wastewater are essential in an industrialized society.

A typical Sewage Treatment method includes the following,

1. Physical unit operation
   -- the removal of pollutants by physical forces.
2. Chemical unit process
   -- the removal of pollutants by addition of chemicals or chemical reactions.
3. Biological unit process
   -- the removal of pollutants by biological activities

These processes will be handled by 3 different tanks. They are Primary Sediment Tank, Aeration Tank, and Secondary Clarifier. The process in each tank is very complex, which involves physical, chemical and biological reactions. Due to the complexity of the facilities, students often have a hard time to envision the detailed reactions. Even some would have visited the real facilities, rarely could they have the chances to get to know what is going on under water, neither will they have a clear impression of how the system work.

Now, with the aid of computer graphics, the whole process will be simulated in a VR environment, in which users have a good chance to get a close look of each part of the facilities, and further to observe the process through animation. Thus, this system will provide an all-new, exciting experience even to those researchers who are familiar with facilities.

4.3 System description

4.3.1. AutoCAD file conversion

The original wastewater treatment facilities models are developed in AutoCAD, a

software commonly used in engineering. Though this model can provide engineers with accurate measurements and dimensions, it only provides a poor representation of the real thing in 3D in the same time. Moreover, no animation can be done in this model, not to mention the interactions between users and the machine.

In order to render the model in a VR environment, we should be able to transform the model in AutoCAD files into data which we can manipulate. AutoCAD has included an interface, known as DXF file, to output the data as points with 3D coordinates.

The DXF file format was originally created by AutoDesk to represent 3D models and scenes built with AutoCAD. Though it is made up of ASCII code only, which makes the file format more like a txt file, the DXF file format is much more complicated than we originally thought. Its typical overall structure is as follows [24]:

1. HEADER section - General information about the drawing. Each parameter has a variable name and an associated value.

2. TABLES section - This section contains definitions of named items.
   - Linetype table (LTYPE)
   - Layer table (LAYER)
   - Text Style table (STYLE)
   - View table (VIEW)
   - User Coordinate System table (UCS)
   - Viewport configuration table (VPORT)
   - Dimension Style table (DIMSTYLE)
   - Application Identification table (APPID)

3. BLOCKS section - This section contains Block Definition entities describing the entities that make up each Block in the drawing.

4. ENTITIES section - This section contains the drawing entities, including any Block References. Various 3D entities can be drawn in AutoCAD R12, including Line, Point, 3DFace, 3D Polyline, 3D Vertex, 3D Mesh, 3D Mesh Vertex.

5. END OF FILE

From the descriptions above, we can see that the data included in the DXF file is so comprehensive that we need a protocol to extract and transform the useful data into the model in the graphics pipeline. Furthermore, since a DXF file is a complete representation of the drawing database, and as AutoCAD is further enhanced, new groups of entities will be added to accommodate additional features. In order to increase system efficiency in rendering, the varieties of entities which can be rendered in the Virtual Reality environment should be restricted.

After clarifying both parties' needs and abilities, then, we worked out the most efficient DXF file suitable for rendering. The file will consist of 3D Face and 3D Polyline entities, which are ready to be rendered in OpenGL, in AutoCAD R12 dxf file format, which is an early DXF version with enough vital data for OpenGL rendering purpose, such as viewpoint information and layer definition.

With all the dimension data available, it didn't take much time till the DXF file complied with our requirements came out. As expected, the 3D Face and 3D Polyline entities are enough to produce an accurate 3D model of the wastewater treatment facilities.

4.3.2. Virtual Reality Environment

To give the users a realistic feeling when browsing the model, multiple Virtual Reality components have been introduced. We adopted the most widely used and supported

2D and 3D graphics Application Programming Interface (API) -- OpenGL to achieve them.

4.3.2.1 A convenient walkthrough mode

This is the basic requirement for Virtual Reality. A convenient walkthrough is the key to success for the whole system, because it decides how the users view the model. A good walkthrough should be designed specifically for different purposes.

In our system, users need to go to every position of the model and observe it in every direction, including the bottom design. Considering most users' browsing habits, the walkthrough mode enables the users to walk through the whole environment with the mouse.

Mouse functions include:

Left-Button: Users can go through the scene by pressing this button, just like actually walking.

Left-Button + Shift: Pan move.

Left-Button + Ctrl: Rotate move around the current viewpoint.

Right-Button: Roll move around the current viewpoint.

Right-Button + Shift: Pitch move around the current viewpoint.

4.3.2.2 Map

Considering the size of the scene, a map indicating the current position of the user's position is very important; just like in the real environment, one cannot go through the whole facilities without a map.

A map is generated by the system every time a new model is introduced to ensure the accuracy of the dimensions. A user's position is indicated by a flickering red dot on

the map. The map is also a portal to get the user from the current position to any position in the model. In case a user is lost in the scene, with just a mouse click on the map, the user will be transferred to the actual position reflected on the map.

4.3.2.3 Environment

We created the terrain and moving blue sky to give the user a comfortable environment to go around. It truly enhanced the user's feeling Virtual Reality.

We implemented the ROAM algorithm to generate the terrain [25]. Though the algorithm is capable of maintaining dynamic, view-dependent triangle meshes and texture maps that produce good images at the required frame rate, it is only implemented once when the model is constructed to create a scene big enough, thus to save the resource for other complex real-time calculation and animations.

Similar to the terrain generation, the sky's generation implements the Fractal theory to render a height field [26]. Instead of being represented as the heights in 3D drawing, the height field is represented with different color value to render a realistic sky with clouds texture. After mapping the sky texture onto the 3D skybox, the sky is made moved by OpenGL's built-in texture manipulating functions [23].

4.3.2.4 Animation of the model

As an integrated part of Virtual Reality, a good animation is pivotal to represent the reality in the VR environment, because the reality is in constant movements.

There are 2 kinds of animation in our system. One is the animation of the moving parts of the model, which are meant to have motions in carrying out their work. While the original AutoCAD file can only provide the information about their position in

their still state, the system will render the animation based on this data and feedbacks from the designers of the model, such as the sludge collector which moves back and forth to push the sediments into collectors in the bottom of the Primary Clarifier.

The other kind of animation is important for the illustration of the whole process of wastewater treatment, because it is the simulation of the water being processed. Users will have a better understanding of how the facilities work after the observation of the water in different stages. This topic will be discussed in details below.

4.4 Water simulation

It is important to know how actual treatment is being processed in each stage, so as to choose the best solution for that part of water simulation.

4.4.1 Primary Sedimentation Tanks

After the coarse solids are removed, the sewage enters primary sedimentation tanks for primary treatment. Most of the suspended matter is settled out and removed as primary sludge for further treatment.

The primary sedimentation tanks accept flow from the central distribution channels. A zone of tranquil flow allows the floc to settle and the scum to float on the surface. The floc settles as sludge and is removed using chain and flight sludge collectors to sludge hoppers. The scum is skimmed off by the same equipment. Both the sludge and scum are then pumped to a separate facility for dewatering.

Here basically the water in this tank is quite placid, with no visible ripples. Under this situation, the water model can be rendered by very small occasional disturbances (see Figure 4-1).

Figure 4-1: Virtual Primary Sedimentation Tank and the real one.

4.4.2 Aeration Tank

In secondary treatment, biodegradable organics and nutrients are removed from the sewage by biological treatment - activated sludge process.

Sewage is passed to the aeration tanks for biological treatment. Air is supplied continuously into the aeration tanks for the growth and reproduction of micro-organisms, which assimilate and decompose the organic matter in the sewage.

Inside the aeration tank, a fine bubble diffused air system is installed to maintain set levels of dissolved oxygen to sustain the growth of the micro-organisms required for biological treatment (see Figure 4-2).

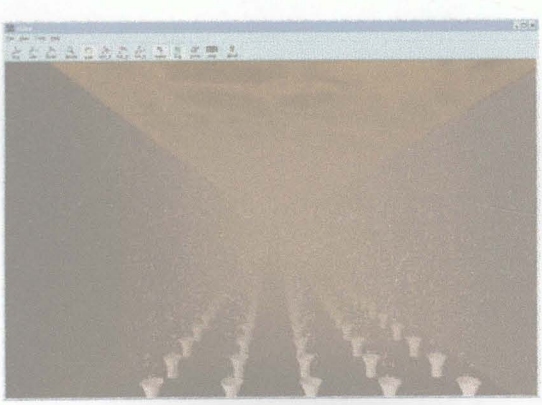


Figure 4-2: Bubble system

The preliminary treated sewage enters into the aeration tank without primary sedimentation process. It first flows through the air contact zone for complete mixing.

After that, it reaches the anoxic denitrification zone. Finally, the sewage flows to the aerobic nitrification zone for nitrogen removal.

As described above, the water is violent while pushed by the air bubbles released by bubble air system installed on the bottom, but quite placid in the noxic denitrification zone. There is very obvious distinction between their surfaces (see Figure 4-3).

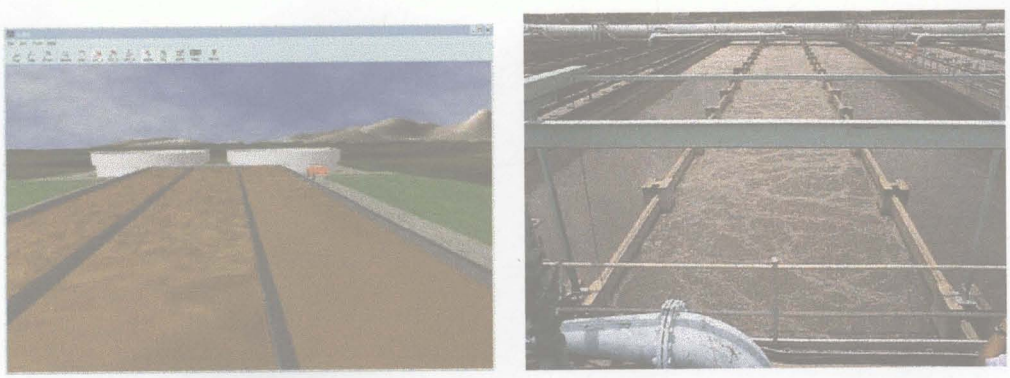


Figure 4-3: Virtual aeration tank and the real one

Since the disturbances are the causes of the ripples on the surface, the locations of the disturbances decide where the waves will occur. Here by confining the disturbances' influential range and adjusting the height and frequency of the disturbances, the fluid model has done a good job in this part. The degree of violence of the water can be controlled by the heights and the frequencies of the disturbances.

The texture underlying the water surface is responsible for the creation of the foaming caused by the turbulence of fluids. The user can thus define how much foam is in the water by controlling the texture's distance factor. The bigger the distance factor, the more textures will be integrated, giving the effect of more foam floating on the water surface.

4.4.3 Secondary Clarifier

The effluent from the aeration tanks is then directed to the final sedimentation tanks. The "activated-sludge" formed in the aeration tanks is settled out. Part of the settled

activated sludge is returned to the aeration tank to maintain the micro-organism population there, while the surplus bio-sludge is collected and delivered to the sludge treatment facilities.

One noticeable feature of water in this tank is that the container has a circular shape and the water will spill out in the radial directions. Besides, the water here is so clean that one could see the sludge down on the bottom. Those features are shown in accordance with the real thing (see Figure 4-4).

Figure 4-4: Virtual aeration tank with water so clean that one could see the sludge on the bottom

4.5 Sludge Simulation

Sludge is formed by the sediments of the wastewater after different stages of treatment. Its animation is also an important part of the waste water treatment simulation. We used the sludge model discussed in the last chapter.

Even though we have developed suitable model for sludge simulation, to fit the sludge into the scene, many modifications must be done. Furthermore, according to the user's demands, since the sludge will accumulate as the sludge collector sweeping on the bottom, the accumulation process must be realized.

We set out from the fluid model's basic properties – the 5 parameters, to achieve the accumulation simulation step by step. Understanding that the major factor of the waves is the disturbance, instead of creating disturbance everywhere in the surface, which is the routine in water simulation, we let the disturbance occur only in certain regions of the surface, i.e. the region close to the sludge collector. Further, the disturbance will be increasing instead of remaining the same.

By scaling the grids into rectangles instead of squares, the sludge is formed to have noticeable slopes in the direction of sludge collectors' movement. By decreasing the grids' number, only limited slopes are formed before they turn plain, which will give the perception of the sludge' accumulation.

The last step is to tune the grid maps' moving speed in accordance with the sludge collectors' movements. Now the sludge is ready to accumulate as the collectors push them(see Figure 4-5).

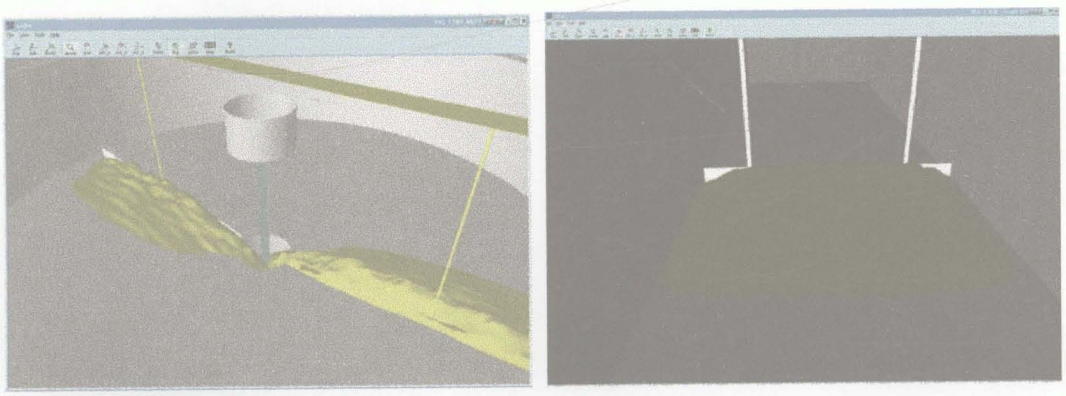


Figure 4-5: Sludge model.

4.6 Air Bubbles Simulation

In the Aeration Tanks, what actually carrying out the job of cleaning the water are the air bubbles emitted from the air nozzles mounted on the bottom of the tank. Their

functions have been briefly introduced in the last section.

There are many ways to simulate air bubbles. We use particle system to render the bubbles.

4.6.1 Particle Systems

Particle system is one successful way of simulating irregular object's movements. In the particle systems, an object is defined by hundreds, thousands or even millions of particles, which individually have their own life period and properties. The particle system is not a simple static system. The combination and the movement of the system are constantly changing. As time elapses, the existing particles in the system will develop and transform. New born particles will be added to the system while old particles which reach their end of the life will be eliminated. Meanwhile, to give the particle system more irregularities, random elements may be introduced to the parameters of each particle, such as speed and acceleration.

The particle system is now widely used in simulating dynamic nature scenarios, such as flowing water and clouds, because of its properties mentioned above. Compared to the traditional modeling methods which are based on the facet elements, the particle systems have the following advantages:

1. Particles are much more simple elements than facets, such as polygons. In this way it is more cost-effective.
2. The definition of the model is defined by the process dynamically. Hence when defining very complex models, it won't take as much as the tradition way which defines the elements in the very beginning. Furthermore, the particle system can be adaptively adjusted according to the variables in the system.
3. The particle system is a dynamic system inherently. Hence it is easier to render

animations.

To construct and render a particle system is not difficult. Here is what happens at a certain moment of particle system:

- New particles are born and added to the system.
- Apply certain attributes to these new particles according to the basic rules of the particle system.
- Delete those old particles whose lives have exceeded the life cycle of the particles.
- Apply necessary motions to each particle in the system according to their attributes.
- Render those particles.

The particle system was first introduced by Reeves[29], who then implemented it in the movie *Star Trek* to simulate the fire walls of a Cosmo explosion. To stress the randomness of the particle system, Reeves made use of some very simplified random processes to control the particle's shape, characteristics and movements. He set the ranges of each particle's parameters, then chose a random value within these ranges. This process can be expressed as:

Parameter = MeanParameter + Rand()*VarParameter

Where Parameter is a certain parameter needed to be decided randomly, Rand() is a random function within the range of [-1.1], MeanParameter is the average value of the Parameter and VarParameter is the variance.

Since the enormous number of particles involved in one particle system, Reeves [30] proposed a render algorithm specifically for particle systems to increase the efficiency of the system. The algorithm is based on the following assumptions:

1. The objects which are represented by the particle systems do not intercept with the geometry models in the scenes, thus the rendering of the particle

system is separate from that of geometry models. In consideration that there may inevitably be some sheltering of the objects, the scene can be divided into sub-scenes, which can be then applied rendering separately. Those sub-scenes can be integrated into one scene using the image processing technologies.

2. Every particle can be approximated as a point light source, which contributes to the lighting of the pixels it covers. The contribution is decided by the particle's properties, i.e. color, transparency.

Based on these assumptions, the workload of the culling of the hidden faces of the particle system will be greatly reduced, because it won't involve any sorting or re-arranging the faces and the efficiency will be enhanced.

4.6.2 Air bubbles animation

The number of the particles is not only related to the rendering effect, but also with rendering speed. A typical particle system often consists of several thousand particles, some of them over five thousand. Even with the above assumptions, the particle system's speed will be greatly reduced if there are too many particles.

Taking into account the various factors concerning the air bubbles' movements in the system, a particle can be constructed as follows:

```
typedef struct
{
    float x,y,z;        //Current position;
    float vx,vy,vz;     //Speed in three directions;
    float dvy;          //Vertical acceleration;
    float dvx;          //Horizontal acceleration;
}PARTICLE;
```

Let us take a look back at the model. If each nozzle will emit 5 particles at a time, and 5 particles will be eliminated when reaching the water surface, then the nozzle will maintain 20 particles at a time. Multiplying the number of particles by the number of nozzles, which are 1500 in our system, the result is 20*1500 = 30,000. This particle system will be quite a challenging task. If we were to treat each particle in this system with same weight, the system would not run smoothly, even though the result must be quite impressive. Note that all elements at each particle must be computed at each moment and the calculation work load will be very heavy.

To decrease the workload of the system, so that the overall rendering can be made in real time, we shrink the whole particle system consisted of 30000 particles, which is used to depict all the nozzles, to a small particle system consisted of 20 particles, which is used to depict one nozzle. The rest of the nozzles will just be a copy of this particle system.

However, taking the above mentioned methods will bring in new problems into the system, in that it will destroy the randomness of the system, because every nozzle's particles will act in the same way as the template. In order to reflect the overall randomness of the air bubbles, we have to find new ways to introduce new randomness.

The new randomness are the drawing positions. The codes below carry out this:

```
for (int i=0;i< Number_of_Nozzles; i++){
                glPushMatrix();
                 glTranslatef(Position_of_Each_Nozzle);
                glRotatef(rand()%360,0.0,0.0,1.0);
                Draw_Particles();
                glPopMatrix();
```

From those codes, we can see that each time before drawing the particles, there is a translation and a rotation. The translation is to move the particles' drawing position to each nozzle's position, while the rotation will alter the drawing angle randomly by randomly deciding the rotation angle. In this way, the animations of the overall air bubbles can be done in real-time.

4.7 Speed-up Techniques

So far the system is complete, but it is still far away from practical use. It is important to tune-up the system's performance to allow it to run smoothly.

Here are some measures taken:

1. *Dynamically change the sizes of the grids*

   The water simulation consumes most of the calculation capacities, because at every moment, every grid's height is to be determined. Much optimization is done to increase the performance of the water simulation while decreasing the resources wasted on it.

   As we mentioned in the earlier chapters, the sizes and the number of the grids are the key of the overall performance of the water simulation. The more the grids, the more detailed the simulation is able to render. However, the system does not have to provide the same details for a viewpoint at a far distance as a near one.

   In light of this, we will dynamically change the grids as needed. Several thresholds are used to determine the number of the grids. For example, if the distance between the viewpoint and the water surface is more than 5000 units, the

grids will be 50*50, which are enough for a viewer from that distance to recognize the water surface. Similarly, in 3000 units, grids will be 100*100.

However, there will still be rendering delay if we don't take other further measures. For example, in a short distance, say 50 units, the grids will be 250*250, which are almost the maximum resolutions. Furthermore, in such a short distance, the rendering delay is very obvious. The solution discussed in the next section will be quite useful.

2. Dynamically change the size of the viewing frustum

Frustum is an importance concept in computer graphics. Its mathematics meaning is: The part of a solid, such as a cone or pyramid, between two parallel planes cutting the solid, especially the section between the base and a plane parallel to the base. In computer graphics, the viewing frustum means the scope between two parallel planes in which the rendering tool will render. In other words, a viewer can only see the objects in the frustum.

In OpenGL, the frustum is defined by:

glPerspective(float fFovy, float fRatio, float nearPlane, float farPlane);

where the nearPlane and farPlane decide the position of the two parallel planes, as their names imply. In short, the OpenGL will only draw objects within these two planes. The fewer the objects are in the range, the less workload. Hence, if we can control the frustum according to the actual needs, the system can be balanced to be utilized to its utmost capacities.

In the system, when the viewpoint is in a distance from the water surface, though the water surface does not require much detail, the objects surrounding the water, such as the tanks, terrains, are to be rendered. The frustum is about 5000 units in height. However, when the viewpoint is in the proximities of the water surface, it

will take a large portion of the calculation power to render the water, while the surrounding objects are to be overlooked or even omitted. In this way, the frustum can be cut to 500—1000 units in height, so the objects outside the frustum will not be drawn and the machine can concentrate its computation power on the water's rendering.

3. Timer management

Timer is introduced to the system for the purpose of animation of various objects, such as water, sludge collector, even the walkthrough and viewpoint change.

There are 5 timers in the system:

Timer 1: Handling the viewpoint change when the mouse left button is pushed. It will calculate the difference between the mouse click and the original mouse position, then move the viewpoint according to the difference.

Timer 2: Similar to Timer 1's function, it will handle the viewpoint change according to the mouse events. But it will respond to the mouse right button's event.

Timer 3: Handling the water's animation. Because there are 3 water animations in the scene, which share the same timer, their motions' period are the same. Since the workload of this animation is the heaviest, to give the priority to other animations, this timer is set to have longest period.

Timer 4: Handling the sludge and sludge collector's motion. Because the sludge has the same moving speed as the collector's movement, it is quite convenient to give it the same speed, position and acceleration.

Timer 5: Handing particle system's animation. As we mentioned just now, the particle system involved is quite a simple one, which consists of 20 particles, it occupies the shortest time to compute.

The management of the timers is to balance their weights in the system, so their

work would not interfere with each other, which will cause some unnecessary delays in computation.

The other aspect of timer management is to control their life cycles, so that they could work effectively without overlap with other timers. For example, when the user is moving in the scene, it won't be necessary for other animations such as water animation, sludge animation to take effect. In this case, it will be advisable to turn off these animation's timers; on the other hand, when the user stopped in one spot, the timer controlling the viewpoint change should be disabled while all the other animations' timer become activated.

4. Dynamically change the particles' properties

In the last section we have mentioned the particle system's optimization, in which we duplicate one nozzle's particle's system and implement it on all the nozzles. Although this method has greatly enhanced the performance, the machine still needs to render 30,000 particles at any moment, which is still very expensive to render because the machine has to handle each particle's color and transparency. There is still much to be done to give a more efficient animation.

Similar to the idea of changing the sizes of the grids and the height of the frustum, the particle system can also be optimized by dynamically changing the quantity of the particles in the system. For example, when viewpoint is in the distance, so not much detail is required, we limit the particle's quantity in the system to only 10, so the overall particles' amount is 15,000. This already saves a lot of computation and rendering, though the view would not be much different from the originals. When at a short distance, since the frustum is truncated, there will leave more computational power to render the particles at full throttle.

4. *Display list*

Although we have enhanced the rendering speed a lot with the above mentioned

speed-up techniques, the display list will boost the performance of the system to the greatest extend.

If an object will not change its shape, color or other properties over time, such as background (i.e. still models), it is not necessary to re-calculate its state every time the scene is refreshed. A display list is a list of OpenGL commands cached in the RAM. It is called to render the object which does not change over time, thus saving the time of re-calculating the object.

The display list can store commands between a glBegin()/glEnd() pair. Thus if one is going to execute repeatedly the same sequence of OpenGL commands, one can create and store a display list. This cached sequence of calls can then be repeated with minimal overhead, since all of the vertices, lighting calculations, textures, and matrix operations stored in the list are calculated only when the list is created, but not when it's replayed. Only the results of calculations are stored in display lists, and thus any complicated calculations can usually benefit by being placed in a display list and having them replayed at a different time.

In the system, the sky, terrain and the facilities model are all calculated and stored into a display list when the system starts. In the rest of the time, the computational power is all focused on the calculation of the animations such as water, air bubbles, sludge.

Through the proper use of display list, complex animations can be decomposed to a sequence of display lists because the animations are made up of still images. This feature enables us to speed up the air bubbles animation by 200%.

When the particle system is initialized, we calculate 20 frames of it, and store into 20 display lists. Later on, the animations of air bubbles are just the replay of these 20 display lists repeatedly. Since the particles' movements are random inherently,

the combination of the display lists is a good description of those movements.

5. *Dynamic Visibility Computation*

This technique was first brought up by Bin Chan [30]. It is composed of a series of computation to find out the visibility of objects and upon that, a culling action is taken to get rid of the invisible objects. It is extremely useful in the rendering of a complex scene. When user's walkthrough only covers limited objects in the viewing frustum; objects out of the frustum should be eliminated before rendering.

For each tank in the scene, one bounding box is created. The box is used later to test the intersection with the viewing frustum. If the box lies in the frustum, system will render that object.

The algorithm is as follows:

When rendering a 3D scene, the stored objects' models may be very large and complex. However the visible objects on the computer screen are possibly only small part of them. If we can judge which objects are visible on the screen or not and only render those visible objects, many drawing time can be saved. The speed up method is called object's bounding box clipping in computer graphics.

Assuming a plane constructed by clockwise points $P_1P_2P_3P_4$, the plane equation is:

$$(\vec{P}-\vec{P_1})\cdot\vec{N}=0, \text{ with } \vec{N}=\frac{(\vec{P_2}-\vec{P_1})\times(\vec{P_3}-\vec{P_1})}{\|\vec{P_2}-\vec{P_1}\|\cdot\|\vec{P_3}-\vec{P_1}\|}$$

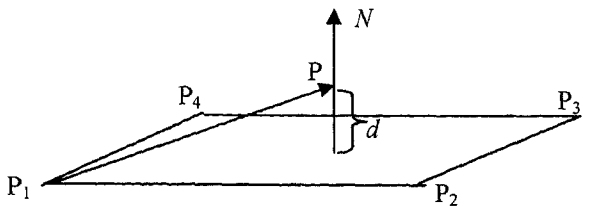


Figure 4-6: the distance d between point P and the plane.

The distance d between any point P and the plane is: $d = (\vec{P} - \vec{P_1}) \cdot \vec{N}$ (see Figure 4-6)

If $d<0$, the point P is under of the plane, otherwise the point is over the plane.

Define an object's bounding box as $[x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}]$; any points of the object is inside the bounding box.

Define Plane 1,2,3,4,5,6: ABFE, BCGF, CDHG, DAEH, ADCB, EFGH;

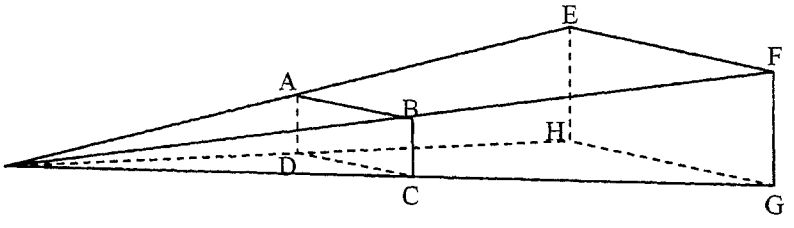


Figure 4-7: Perspective frustum

If the six points of an object's bounding box are all over the six planes, namely $d_i>0$ for any i =1,2,3,4,5,6, the object is outside the perspective frustum.

If an object's bounding box is fully outside the frustum, don't display the object because it is not visible. Otherwise draw the object by OpenGL functions.

By the object's bounding box clipping method, much time is saved for rendering because only small possible visible objects are drawn in the scene.

4.8 Further work

Since our initial motive was to develop a 3D model, the system is far from completion at this stage. There are still many improvements needed.

- The VR environment in our system is only designed specifically for the wastewater treatment facilities, it is not a general one for most of the other engineering purpose. In our system, except the terrain and skybox, other VR components will be useless if implemented in projects, or example, the water model. Actually, if we were to develop a comprehensive model for other visualization projects, various VR components can be pre-made into the system, and the users can choose according to their needs.

- The interaction with the users is not enough. So far the system can only carry out limited interactions, such as allowing the users to go around the environment and letting them adjust the wastewater properties. We get some feedbacks that, since wastewater treatment facilities can have multiple forms of layout or components, it would be more useful if the users can design their layouts and components with the existing model.

- So far the system can only read limited components of the AutoCAD DXF files, which limits its applications. Even though the users can convert their models under AutoCAD to the format readable by our system, it will be more convenient if our system is ready for different DXF files. As mentioned earlier, the DXF file format is designed to carry the new forms and new entities of later version of AutoCAD, our system must be updated, and expanded to meet this requirement.

4.9 Conclusion

The system has basically fulfilled the need for an interactive, interesting and impressive education software, and provides a tentative prototype to give the traditional engineering education a new look.

The fluid simulation engine has fulfilled the need for a real time simulation of fluid phenomenon. In this system, the waste water and sludge are simulated in real time. Users can perform the interactions with the machine while these animations are running.

Although the system only has limited implementations, it provides a good example of how the advanced visualization techniques can be combined with the engineering software, to turn the tedious engineering dimensions into a colorful 3D virtual environment, which, in turn, can be implemented as a new way of education for serious topics.

Chapter 5

Conclusions

In this thesis we have introduced a new model of fluid simulation, which is suitable for real-time engineering visualization projects, due to its easy-to-use controllability, fast rendering speed, stable and convenient implementations.

This simulation model is based on a simplified set of the governing equations in fluid dynamics -- *Navier Stokes* equations, which are able to accurately describe fluidity. Though *Navier Stokes* equations are very powerful, their complex computations limit their real-time implementations. The simplified sets, on the other hand, have been deduced under some reasonable assumptions, thus eliminate the non-linear factors in *Navier Stokes* equations. Hence they are very efficient in calculating the waves.

This simplified model provides us a good model to start with. Based on this model, there are many fast and efficient computational methods. By the use of 2 arrays, one containing the water's state in the last frame can be combined with the other one containing the water's state in the frame before the last to calculate the state of the water in the current frame, through simple calculations. The result is that any disturbance in the grids can be transferred like ripples in fluid to other grids. The process is fast, stable and cost-effective.

However, the model still has some defects, such as the square waves' forms and the acute climaxes in the fluid surface. We further take measures to optimize the model. We use a filter which will extend the disturbance to 8 directions surrounding the disturbance, in order to form a more round ripple form. Furthermore, motivated by the smoothing filtering in Image Processing, we implement Gaussian filter to alleviate the climax formed by the disturbance. With these filters, the wave forms are optimized significantly.

The rendering of the fluid is achieved by the implementation of environment mapping, a texture mapping technique which is capable of creating the illusion of reflections using simple texture mappings, without actually calculating the rays' intersections in the scenes. With the help of environment mapping, the simulation takes on a glittering outlook, which is comparable to the real water.

Texture mapping not only plays an important role of rendering the water in the simulation, it is also responsible for the flow of the simulation. Due to its properties in computer graphics applications, it can be easily implemented to create simple animations such as translation, rotation. In this way, the water is brought to life by moving the texture underlying the water surface.

There are 6 governing parameters in this model:

1. Grids

Setting up the grids will define the detail level of the fluid. The more the grids, the more detailed the simulation will be, so will the computation workload.

The shape of the grid also decides the shape of the waves. If the grid is just a square, the ripples will appear circular; similarly, if the grid is adjusted to a rectangle, the ripples will then propagate like an ellipse.

2. *Height of the wave*

The heights of the waves are decided by the disturbances added to the model. The higher the disturbance, the more violent the fluid will be. This value must be accordance with the frequency of the disturbance.

3. *Frequency*

The frequency here does not mean the frequency we encounter in the waves theories. It is the frequency of the disturbances. If the frequency is set too high, it will cause some similar adverse results as the disturbance will do. Hence, the frequency and the disturbance must be taken into account together.

4. *Damping*

The wave is traveling outwards while losing the energy. If the energy remains the same, the wave will continue traveling and bouncing back and forth in the pool. How to choose a suitable damping value depends on the physical properties of the fluid which the user wants to simulate.

5. *Smoothing scale factor*

The smoothing scale factor has the direct relation with the smoothing filer – Gaussian filter, which is implemented to smooth the effect of disturbance. The scale factor is the decisive value in the smoothing filter, in which it decides the influenced scale, and how much each of the influenced cell's weight in the whole filter.

6. *The flow speed*

This factor is related to the texture mapping. As we mentioned just now, the flow speed is actually the moving speed of the texture used to render the surface.

Though these parameters do not have actual physical meanings, through the controls over the governing parameters of the simulation model, users can make use of them to achieve different fluidity effects, such as the sludge model which has been implemented on the a visualization project – Virtual Sewage Treatment System.

Virtual Sewage Treatment System is a joint project by the Department of Computer Science and the Department of Civil Engineering, designed to simulate the process of sewage treatment. Various fluid phenomena can be seen in this Virtual Sewage Treatment System, including still water, violent turbulence and sticky sludge. The result proves that by adjusting those parameters, the fluid model we develop can simulate different fluidity well. Furthermore, the system can achieve real-time rendering by taking relatively low system resource. It is a fast simulation model which can be adapted to various engineering projects which do not require accurate descriptions of the fluids. Equipped with this fluid model, Virtual Sewage Treatment System has provided a strong visual aid for both students and researchers alike.

[1]R.Peyret and T.D.Taylor, *Computational Methods for Fluid Flow*, Springer-Verlag, New York, 1985.

[2] W.R.Fox and A.T.McDonal, *Introduction to Fluid Mechanics*, 4th edition, John Wiley and Sons, New York, 1992.

[3] M.B. Abbott, *Computational Fluid Dynamics: An Introduction for Engineers*. Wiley, New York, 1989.

[4] J.X. Chen and N.V.Lobo, "Toward Interactive-rate Simulation of Fluids with Moving Obstacles Using Navier-Strokes Equations", *Graphical Models and Image Processing*, Vol 57, No. 2, Mar. 1995, pp. 107-116.

[5] J. X. Chen, el. "Real-Time Fluid Simulation in a Dynamic Virtual Environment",

[6] T. Nishita and E.Nakamae, "Method of displaying optical effects within water: Using the accumulation buffer", *Proceedings of SIGGRAPH'94, July 1994*, pp. 24-29.

[7] P.T's and B.Barsky, "Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping", *ACM Trans. Graphics*, 6(3), 1987, pp. 191-214.

[8] M.Watt, "Light-water interaction using backward beam tracing", *Proceedings of SIGGRAPH'90, Computer Graphics*, 24, 1990, pp.377-385.

[9] A. Fournier and W.Reeves, A simple model of ocean waves, Proceedings of SIGGRAPH'86, *Comput. Graphics*, 20(3), 1986, 75 – 84.

[10] N. Max, Vectorized procedural models for natural terrain: Waves and islands in the sunset, Proceedings of SIGGRAPH'81, *Comput. Graphics*, 15(3), 1981, 317 –324.

[11]D. Peachy, Modeling waves and surf, Proceedings of SIGGRAPH'86, *Comput. Graphics*, 20(3), 1986, 65 – 74.

[12] M. Kass and G. Miller, Rapid, stable fluid dynamics for computer graphics, Proceedings of SIGGRAPH'90, *Comput. Graphics*, 24(3), 1990, 49 – 57.

[13] Teukolsky, Vetterling, Flannery, *Numerical Recipes in C, The Art of Scientific Computing, Second edition*, Cambridge University Press.

[14] Miguel Gomez. "Interactive Simulation of Water Surfaces". *Game Programming*

Gems, Charles River Media, 2000.

[151 Le Mehaute, B., *An Introduction to Hydrodynamics and Water Waves,* Springer-Verlag, New York (1976).

[16] Crapper, G., *Introduction to Water Waves,* John Wiley & Sons, New York (1984).

[17] Stoker, J., *Water Waves,* Interscience, New York (1957).

[18] Linda Shapiro and George Stockman ,"Filtering and Enhancing Images", *Computer Vision,* Prentice Hall, 2001

[19] Eric Weissten, "Gaussian Function", *The CRC Concise Encyclopedia of Mathematics,* Book News, Inc.

[20] Blinn, J. F. and Newell, M. E. Texture and reflection in computer generated images. *Communications of the ACM* Vol. 19, No. 10 (October 1976), 542-547

[21]Gene S. Miller and C. Robert Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. *Course Notes for Advanced Computer Graphics Animation,* SIGGRAPH 84

[22] Paul Haeberli and Mark Segal. Texture Mapping as a Fundamental Drawing Primitive. *Fourth Eurographics Workshop on Rendering.* June 1993, pp. 259-266

[23] Mason Woo, el. *OpenGL Programming Guide*, 2nd edition, Addison-Wesley, New York, 1997.

[24] AutoDesk, *Drawing Interchange and File Formats Release 12*, 1992 Autodesk, Inc.

[25] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent implification for polygonal models. In *Proc. Visualization '96*, pages 327–334. IEEE Comput. Soc. Press, 1996.

[26] Fournier, A., Fussell, D., Carpenter, L., Computer rendering of stochastic models, *Communications of the ACM,* June 1982

[27] Andrew S. Glassner, *Graphics Gems*, Academic Press, 1990

[28] Hart, John C., George K. Francis, and Louis H. Kauffman. Visualizing quaternion rotation, *ACM Transactions on Graphics*, 13 (3): 256-276

[29] Reeves W. T. "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects".*Computer Graphics*, Vol. 17, No. 3, pp. 359-376, 1983.

[30] Reeves W. T. "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems". *Computer Graphics*, Vol. 19, No. 3, pp. 313-322, 1985.

[31] Bin Chan, "A Virtual Walkthrough System For Complex Indoor Environments", *M. Phil thesis, Department of Computer Science, The University of Hong Kong*, 1998.