

Water Simulating in Computer Graphics

Kai Li
Liming Wu

Abstract:

Fluid simulating is one of the most difficult problems in computer graphics. On the other hand, water appears in our life very frequently. This thesis focuses on water simulating. We have two main methods to do this in the thesis: the first is wave based water simulating; Sine wave summing based and Fast Fourier Transform based methods are all belong to this part. The other one is physics based water simulating. We make it based on Navier-Stokes Equation and it is the most realistic animation of water. It can deal with the boundary and spray which other method cannot express. Then we put our emphasis on implement by the physics method using Navier-Stokes Equation.

Keywords:

Water simulating; Sine wave summing; Fast Fourier Transform (FFT); Navier-Stokes Equation (N-S equation); MAC Grid.

Acknowledgements

We have to first thank our supervisor Gösta Sundberg who inspired our interests in Computer Graphics and leaded us to do the reaches in water simulating. His valuable suggestions helped us a lot in our researches. We should appreciate Martin Blomberg whose lectures are very useful and gave us some help on Open GL. On the other hand, we should also express our gratitude to our parents and friends here. They gave us a great deal of spiritual support in the process of the thesis writing.

Thank all people that we mentioned above, they are one of the keys that we successfully finished this thesis.

Växjö, Sweden, Jun 2007

Index

1. Introduction:	1
1.1 background	8
1.2 Purpose and research questions	8
1.2.1 Purpose	8
1.2.2 Research questions	9
1.3 Difficulties	9
1.4 Actuality in water simulating	9
1.5 Emphases	10
1.6 Technical approach	10
1.6.1 Video Cards	10
1.7 Outline of the thesis	12
2 Wave based water simulating	13
2.1 Introduction	13
2.2 Sine wave summing	13
2.2.1 Sine wave	13
2.2.2 Water wave modeling	14
2.2.3 Bump mapping	15
2.2.4 Rays for water	17
2.3 Fast Fourier transform in simulating water	19
2.3.1 Introduction of Fourier transform and Fast Fourier transform	20
2.3.2 Gerstner-wave	21
2.3.3 FFT method which based on Gerstner wave	21
3 Physics based water simulating	26
3.1 Introduction	26
3.2 Previous Work	27
3.3 Basic Mathematics and Physics	28
3.3.1 Mathematical Background	28
3.3.2 Physics Background	32
3.4 Navier-Stokes Equation	35
3.4.1 N-S Equation introduction	35
3.4.3 N-S Equation Expression Symbols	36
3.4.4 Get Momentum Equation from Newton's Second Law	38
3.4.5 Solve Equation Gradually	40
3.5 Water Simulating Strategy	46
3.5.1 MAC Grid	46
3.5.2 Particle-in-Cell Methods	49

3.5.3 Maker particles and Boundaries condition.....	50
3.6 Implementation	52
3.6.1 Platform.....	52
3.6.2 Opengl Structure	53
3.6.3 System structure	54
3.6.4 Data Structure	56
3.6.4 Data Structure	57
3.6.6 Particles	61
4 Conclusion	63
5 Reference.....	64

List of figures

Figure 1. 1 Titanic and simulated ocean water.....	8
Figure 1. 2 it shows different methods nowadays in this field.....	10
Figure 1. 3 this sketch map shows how vertex shader and pixel shader works in GPU.....	11
Figure 1. 4 Image left shows Vertex shader and Pixel Shader is not full used in different situations. Image right shows how unified shader works.	12
Figure 2. 1 bump mapping	15
Figure 2. 2 start image, deeper and higher lines for bump used on the picture, and picture after bump map.....	16
Figure 2. 3 detail about bump mapping.....	16
Figure 2. 4 Polygon with vectors.	17
Figure 2. 5 bump map, and the polygon.....	17
Figure 2. 6 an example of Cube Map.....	18
Figure 2. 7 an example of gloss map.....	19
Figure 2. 8 the final water	19
Figure 2. 9 sine wave and Gerstner wave.....	20
Figure 2. 10 graph of Gerstner wave with different amplitudes.....	21
Figure 2. 11 region with length L_s and waves with different wavelengths	22
Figure 2. 12 two-dimensional region	23
Figure 3. 1 Height field (from SIGGRAPH 2006 Reel Time Fluids in Games).....	27
Figure 3. 2 Pipe	28
Figure 3. 3 left one: MAC Grid in 2D.....	31
Figure 3. 4 Vector Calculus Operators.....	32
Figure 3. 5 Water Properties	33
Figure 3. 6 Coulomb's experiment equipment and the attenuation result.....	34
Figure 3. 7 molecule cohesion.....	34
Figure 3. 8 Comparison Lagrange's way and Euler's way.....	35
Figure 3. 9 Navier-Stokes application (From google pictures)	36
Figure 3. 10 Incompressible equation(divergence free) from: Mark Harris/ GPGPU tutorial Siggraph 04	37
Figure 3. 11 Stam[16] "Stable Fluid " chapter: Method of Solution.....	43
Figure 3. 12 Computing Fluid Advection[23] The implicit advection step traces backward through the velocity field to determine how qualities are carried forward.....	44
Figure 3. 13 bilinear interpolation [24]	44
Figure 3. 14 Linear Solvers From Jos Stam Stable Fluids ppt SIG 2004.....	45
Figure 3. 15 2D MAC Grid From	48
Figure 3. 16 Particle-in-Cell Method (on the right)	49
Figure 3. 17 How define the state of each grid, (a) show the surface grids; (b) grow empty grids;	

(c) growing full grids. (from [24]).....	51
Figure 3. 18 Boundary Conditions on MAC Grid (from [23]).....	52
Figure 3. 19 Velocity field.....	61
Figure 3. 20 water particles.....	62

Glossary:

N-S: Navier-Stokes

MAC Grid: Marker-and-Cell Grid

FFT: Fast Fourier Transform

1. Introduction:

Everybody should be very familiar with the movie <Titanic>. At the beginning of the movie, the Titanic goes towards the sea slowly. The view is so beautiful. But imagine that, the model of Titanic never sailed in the real sea. They are all made of 3D animations. Most water in this movie is simulated but not real. This is only one profile that shows importance of water simulating. See figure 1.1. [2]



Figure 1. 1 Titanic and simulated ocean water

1.1 background

In computer graphics, people keep on simulating the real world around us by computers. But sometimes, some phenomenon is not as simple as it looks like. We can not find a simple model to describe it. In this situation, we have to track the physics reason and describe it from the physics way inside. For example, the motion of objects and affection between objects. Together with the improvement of hardware and processing ability, computers can manage more and more complicated equations and operations. Water simulating is one aspect of these.

More and more people get interested in water simulating in the area of computer graphics recent years. The real time simulating of open-water has became very widely used and also important in games, virtual wars, movies and other areas. And physics based animations becomes an important part of computer graphics.

1.2 Purpose and research questions

1.2.1 Purpose

Since fluid dynamic especially water animation keeping a high progress speed in our everyday use, this thesis mainly emphasized different methods that used in water simulating, three main methods are discussed and the one most realistic and dynamic

is implemented. We also compared differences between these methods, tried our best to find the most proper method in different domains.

1.2.2 Research questions

How to achieve water simulation by sine wave summing approach?

How to implementing FFT to water modeling for generating a more effective water surface?

How to simulate realistic water by using N-S Equation?

Which method is appropriate in different situation?

1.3 Difficulties

The shape of water is stochastic, dynamic. So you can not render it use the static geometry models with triangles. Every frame should be refreshed.

The waves of water are generated by all kinds of forces integrated together. These forces are changing and so waves of water are changing with them. The motions of these waves are very complicated.

Liquid mechanics based wave models are usually very complicated differential equations. People cannot gain a nice effect simulating use our ordinary computers. Computers needed to work with these things are workstations

Colors in water surface are mainly defined by reflection and refraction in the surface. The proportion of reflection and refraction will change with the different angles of view.

1.4 Actuality in water simulating

There are many kinds of method to simulating water. They can be sorted into two main ways: Literature [6] use a set of linear wave to give an expression to the wave function, then compose high fields of water surface, and use particle system to simulate spray of water. Literature [7] based on FFT (fast Fourier transform) models, it can simulate water with small waves in a very nice effect. Movie <Water World> and <Titanic> are using FFT models. However, FFT can only be processed in CPUs and cannot use the function of GPUs. There are also someone use the Fourier correlative method in this field.

One easier way is to pile up different sine wave together. It is a Fourier correlative method. It piles up different sin waves with different frequencies, swings to generate water waves that we need. Rapid speed is the strongpoint of this method.

The other Physics based methods. These are usually using the Navier-Stokes equation. They set up wave models using the classical hydrodynamics, and then make shapes of water using the result that calculated from the equation. It generated automatically when boundaries and initiatory conditions are given. Therefore, shapes of waves are very similar with the real physical phenomenon. The only limit is that it

is very difficult to process the equation.

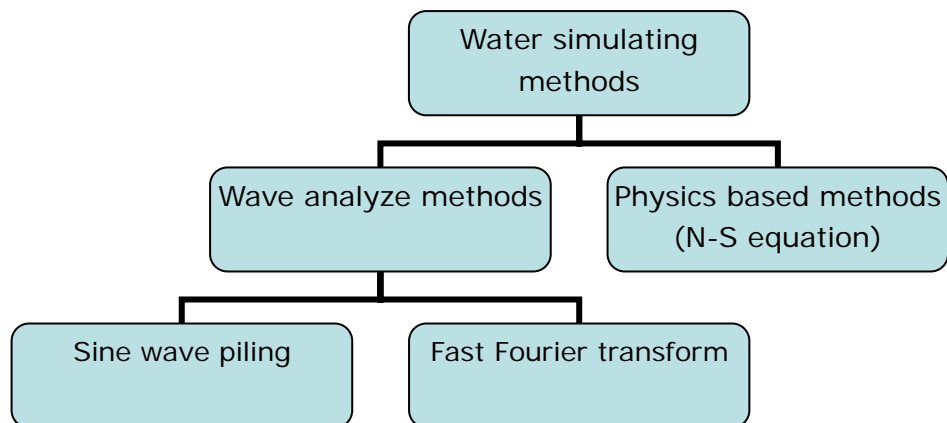


Figure 1. 2 it shows different methods nowadays in this field

1.5 Emphases

Different methods of water animation are the emphases of this thesis. In all kinds of methods of water simulating, the most realistic one is physics based, Navier –Stokes Equation is the most comprehensive method to express characters of water. On the other hand, it not easy to solve N-S equation, a more different part is using N-S equation to the real water simulating. These are the emphases of this thesis

1.6 Technical approach

1.6.1 Video Cards

When working with computer graphics, video cards are very important. Cards in different period use different technologies. The newest card may made by the newest technology that support many new functions. Whose are what we can use for our water simulate.

Cards with Vertex Shader and Pixel Shader

Generate the animations of water on PCs is very hard several years ago. Since NVIDIA video card Geforce 256 comes out in, these things became possible and easier. The function of real time rendering is more and more powerful. Geforce 256 is the first video card that was called GPU (graphics processing unit). The revolution of this card was integrated the T&L (Transform and Lighting) part in GPU. Therefore, this decreased a lot of the CPU's works. In addition, the function of parallel process breaks the choke point of slow speed in complicated processes. Then, NVIDIA issued Geforce 3. It leaded a new generation of video cards in that period. Geforce 3 is the first programmable GPU, with unattached *vertex shader* and *pixel shader*. They can be used to simulate very complex animations. But those are directx 8 and 9 generation.

Most of computers now are using this kind of video cards.

Vertex Shader is a way that GPU built up the three dimensional vertices. Vertex is a basic element in different geometrical graphics. As we know, three vertices can compose a triangle, and some different number of triangles can compose different objects with different complexity. When the 3 dimensional engine transmitting objects on the picture to GPU, it actually transmitting vertices of these objects. Every vertex of this kind include a lot of information: X, Y, Z these kind of 3 dimensional coordinates, colors, and also it's normal, texture information and so on. [3]

The main tasks of Vertex Shader is to process those all kinds of information about the vertices. After processed by Vertex Shader, we can get changed vertices with illuminate. The next step is cutting out, to cut vertices which out of scene. And the back, to cut vertices in the back which can not be seen.

Then, Pixel Shader will process the final color of every vertex combine with information about color, ray and textures. The last step is to change the real 3D scenes to the 2D scenes and display them on our screen. [2]

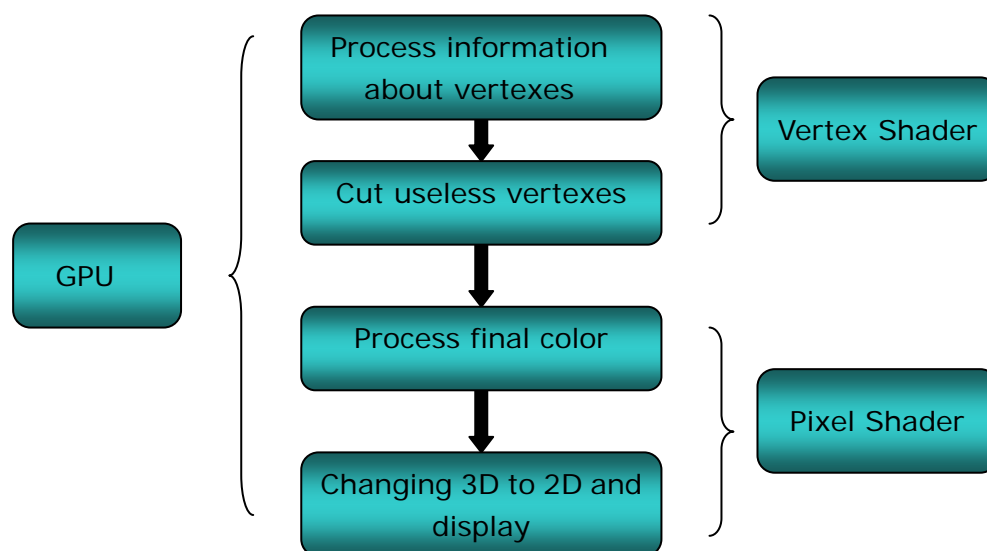


Figure 1. 3 this sketch map shows how vertex shader and pixel shader works in GPU

New technology- Unified Shader processors & DirectX 10

Before directx10 comes out in Jan of 2007, most of GPUs are worked with the structure of Vertex shader and pixel shader. GPUs with different numbers of Vertex shaders and pixel shaders depend on designers. However, in most of situations, vertex shaders and pixel shaders are not full used. For example, some games with very complicated models need very high vertex resource. But the other way, some games put it's emphases on pixel rendering to make a better visual effect need high pixel resource.

Recently, after directx 10 offered by Microsoft. Nvidia has offered it's newest card which brings a new generation. The most innovation is replaced Vertex Shader and Pixel Shader with Unified Shaders. So it can adequately use the resource of GPU.

[4] [5]

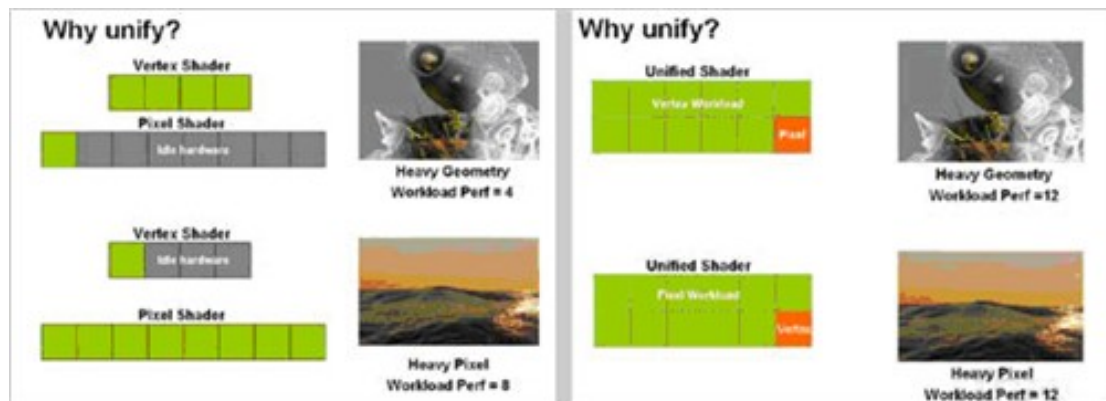


Figure 1. 4 Image left shows Vertex shader and Pixel Shader is not full used in different situations. Image right shows how unified shader works.

Since the second technology is the latest. Most of video cards on our computers are all directx 9 cards. So we are still using Vertex shader and pixel shader.

1.7 Outline of the thesis

In chapter 1, we introduced methods that we use in water simulating. We also listed out questions that we have researched in this thesis, they sequenced by how difficulty the question is. That is also how the content of this thesis arranged.

Technical background is introduced first in every chapter. The second method of chapter 2 is based on the first one, they are all wave based.

In chapter 3, we first introduced basic mathematics and physics we have used in the thesis, then it's content about Navier-stocks equation, after that, we come to the water simulating method.

2 Wave based water simulating

Wave based simulating is one of the most important methods that in fluid simulating. In this chapter, we have two methods about this.

2.1 Introduction

In this chapter, we will discuss two algorithms, which are wave based. Why we call it wave based is because they use sine wave or other waves to simulate water. There are different kinds of algorithms in water simulating. These two are typical and most ordinary ones.

2.2 Sine wave summing

This section is about the sine wave piling algorithm. This is a simple algorithm. It is a Fourier composing method to pile up several sine waves with different frequency, amplitude. The excellence of this method is that it is very quickly. It uses the parallel processing function of GPU to achieve the modeling processes. Only the dynamic waves are not enough. What we also need a special model of illumination, or the water will not be very animat. In this section, we will use vertex shader to implement piling some sine wave together to simulate the dynamic water surface, use the cube map for environment around, and use the pixel shader to accomplish the special illuminating model.

It includes the introduction about the basic of sine wave; and then will discuss water wave modeling; Bump mapping for tiny waves; cube map; and so on.

2.2.1 Sine wave

Sine wave is one of the simplest waves. As we see below, here are some parameters about sine wave equation:

Amplitude of these waves (A): half of the length between wave crest to trough;

Wavelength (L): distance between two wave crests;

Spatial angular frequency (w): direction of spatial angular frequency is the same with the wave diffuse direction, and the quantity is relative with the wavelength L: $|w|=2\pi/L$;

Speed of waves (s): distance of the waves moved in one second;

Temporal angular frequency (wt): $wt=S*2\pi/L$;

Direction of waves (D): the direction of the wave crests.

Initiatory phase (FI): the initiatory phase of waves;

Then, the sine wave equation can be written in these parameters like this:

$$Y(x, y, t) = A * \cos(w * (x, y) + wt * t + FI); [9]$$

We can classify sine waves to directional waves and circular waves. If the shape of a wave is parallel, it will be a directional wave, otherwise if it is circularities. It will be a circular wave. In general, it is proper to use the directional waves to simulate a large water surface drove by wind force. However, for some pint-size lakes, for example if you throw a small stone into it, it will excited some circular waves. In this instance, the circular waves are more appropriate. The main part is directional waves, because they are more familiar in our everyday life and circular waves are very simple, we just focus on the main part.

2.2.2 Water wave modeling

Water wave modeling is one of most important part in water simulating.

From the equation that has described before, the sine wave piling equation can be also writed like:

$$H(x, y, t) = \sum_{i=1}^n A_i \times \sin((D_{ix} \times x + D_{iy} \times y) \times 2\pi / L_i + t \times S_i \times 2\pi / L_i)$$

in this equation, A_i is the amplitude of number i sine wave;

L_i is the wavelength of that wave;

D_{ix} , D_{iy} are direction of it;

S_i is speed of it;

t is time.

This equation actually defined the height fields of the wave at a time t . We can get the movement of water wave when t changes continuously. Implement this equation to the function of displacement, and then use it affect the surface to get the dynamic effect of waves.

Theoretically, every undee wave can be generated by piling a great deal of sine waves. But if we want to render the water in real-time. A small quantity of waves (like 4 or 5) are better.

The speed is quick in this way. However, the real-time water produced by this method is very coarse. It is only a profile of water surface. For fetching up this shortcoming, the method here is *bump mapping*. Map the *bump mapping* to the profile of water surface to describe tiny waves on the surface (see figure 2.1). This can improve a lot of third dimension.

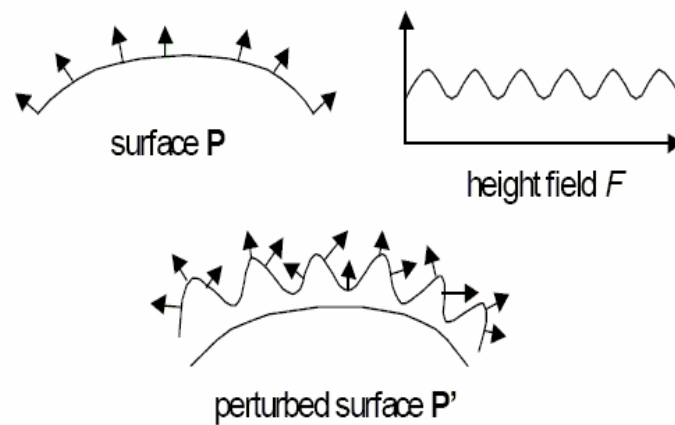


Figure 2. 1 bump mapping

2.2.3 Bump mapping

“Bump mapping is a computer graphics technique where at each pixel; a perturbation to the surface normal of the object being rendered is looked up in a heightmap and applied before the illumination calculation is done. The result is a richer, more detailed surface representation that more closely resembles the details inherent in the natural world.” [12]

Here is an example of bump mapping, see figure 2.2. Image *A* is the one we start with; Image *B* was painted black (deeper) lines; Image *C* with white (higher) lines; And image *D* is how the picture looks with both sets of lines over the face.



(A)



(D)



(B)



(C)

Figure 2. 2 start image, deeper and higher lines for bump used on the picture, and picture after bump map.

Here come some more details about bump mapping. First is to convert the bumps on the bump-map into vectors for each pixel. See figure 2.3, in left figure, it is a zoomed-in view of bump-map, the lighter pixels stick out more than the dark ones. A vector must be computed for each pixel. These vectors represent inclines of the surface at a pixel. See the figure on the right. It represents this. The little red vectors point in the downhill direction.

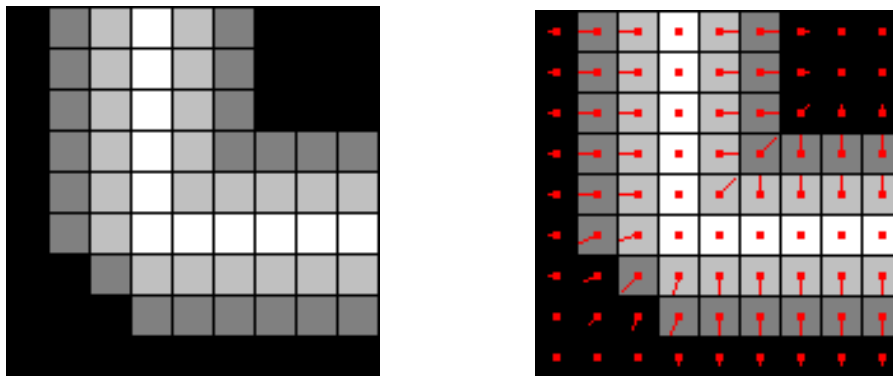


Figure 2. 3 detail about bump mapping

One widely used method to calculate these vectors is to calculate the X and Y gradient of a pixel:

$$x_gradient = pixel(x-1, y) - pixel(x+1, y);$$

$$y_gradient = pixel(x, y-1) - pixel(x, y+1); [10]$$

We will need to adjust the normal vector of the polygon at that point with these two gradients. See figure 2.4; it's a polygon with original normal vector n . There are also vectors that are going to be used to adjust the normal vector for this pixel. They must be aligned with bump-map for the polygon to be rendered correctly. I.E. the

vectors are parallel to the axes of the bump-map.

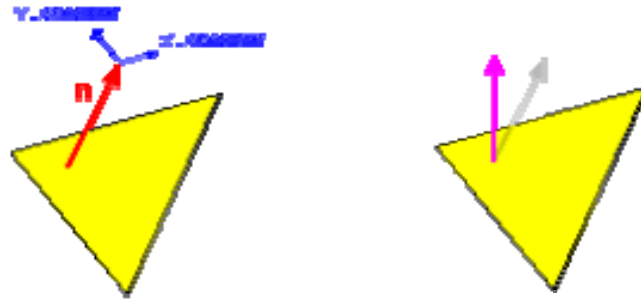


Figure 2. 4 Polygon with vectors.

Then see the figure 2.5; here are bump map and polygon. They all show U and V vectors. After the simple adjustment, we can see the new Normal vector is:

$$New_Normal = Normal + (U * x_gradient) + (V * y_gradient);$$

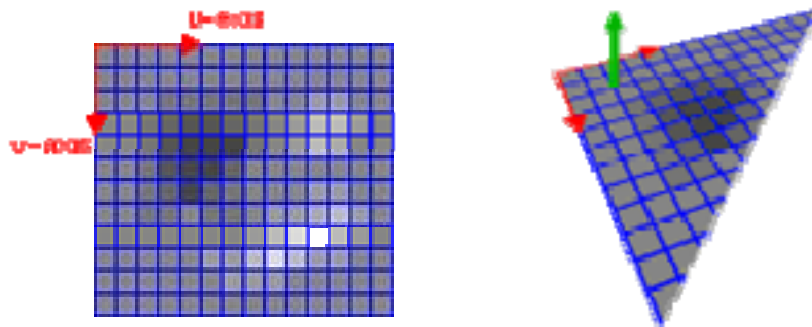


Figure 2. 5 bump map, and the polygon

After that, we can calculate the brightness of the polygon at that point with this New_Normal vector.

2.2.4 Rays for water

All GPUs now are all support the Cube Map. It usually used to implement the object surfaces' reflection of surroundings. See figure 2.6, on the left is an example. Here the Cube Map is used to simulate the sky in the water world. A set of cube map images form the faces of a cube.

The cube map is composed by 6 foursquare 2D textures, each of them denotes one face of the cube. The coordinates of textures are defined as 3D vectors, which point to a point of the cube surface from the center. See figure 2.6; on the right is the whole processes of the cube mapping. The incident ray I arrive the surface of object from the point of eye, the normal vector of point O is N; Then the incident ray is reflected by the surface of the object, a reflection R arise. Then, the angle of incident is the same as reflection angle, so the reflected ray is:

$$R = 2N(N * I) - I;$$

This reflected ray intersected with the cube map at the point E. Therefore, the reflected color of O on the object surface is the function of texture value at point E. In

addition, the reflected ray R is apparently the 3D texture coordinates of the cube map.

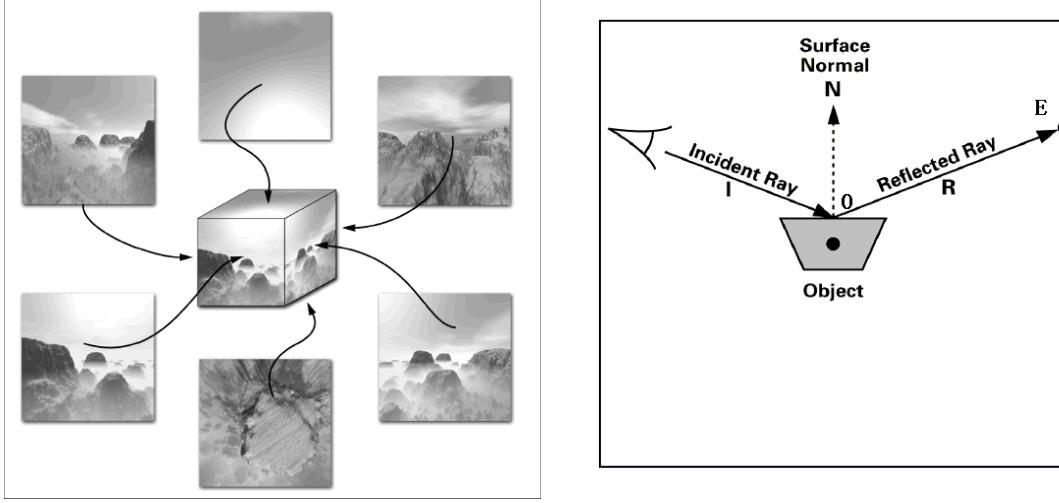


Figure 2. 6 an example of Cube Map

As a transparent object, the color of diffuse reflection about water $C_{diffuse}$ is composed by two parts that mixed together with different percents. One is the reflection from the sky C_{sky} , and the other one is refraction from the rays under water $C_{underwater}$. The mixing percents are determined by the Fresnel coefficient:

$$C_{diffuse} = (Fresnel \times C_{sky} \times S_{sky} + (1 - Fresnel) \times C_{underwater} \times S_{underwater}) \times \max((N \cdot I), 0)$$

S_{sky} , $S_{underwater}$ are all proportional coefficient in this equation.

Color that created by the reflected rays from sky can gained though 3D texture coordinates. It is easy to search the cub map if we know 3D texture coordinates. Just suppose the color of refraction ray from underwater is green. The value bound of Fresnel coefficient is $[0.0, 1.0]$. If the Fresnel coefficient is 0.8; Then the percents of reflected rays is 80%, and rays arrive the surface of water from refraction underwater is 20%; If the angle of normal vector and line of sight approaches 90 degree, the Fresnel is the largest; if it approaches 0 degree, the Fresnel is the smallest. Fresnel coefficient can be calculated from this equation:

$$R(\beta) = 1 - \cos(\beta); (\beta \text{ is the angle of normal vector and line of sight.})$$

When the sun shining above the surface of water, high light will be yielded. A simple method implemented here to calculate the high light $C_{specular}$ of the water. Since the green element in the texture is the most at the place of sun in a cube map. To gain a gloss map, what we need to do is just calculate the eighth power of the green channel. See figure 2.7; it is a gloss map.



Figure 2. 7 an example of gloss map

Then we can confirm the high light C_{specular} of every point on surface of object though the gloss map:

$$C_{\text{specular}} = C_{\text{waterhighcolor}} * (C_{\text{skygreen}})^8 * \max((N * I), 0) * S_{\text{specular}};$$

In this equation: $C_{\text{waterhighcolor}}$ is white; C_{skygreen} is the value of green channel; S_{specular} is the proportional coefficient.

In conclusion, the color of water:

$$C_{\text{water}} = C_{\text{diffuse}} + C_{\text{specular}} + C_{\text{ambient}};$$

Here C_{ambient} is the color of surroundings; $C_{\text{ambient}} = C_{\text{sky}} * S_{\text{ambient}}$; S_{ambient} is the proportional coefficient; Figure 2.8 is a image of the final water.



Figure 2. 8 the final water

2.3 Fast Fourier transform in simulating water.

In this section, a more effective method for water modeling will be introduced. It is FFT method. We will start from the Gerstner wave because the FFT method is based on it. It is a set of Gerstner waves summing together and processed with fast Fourier

transform

Sine wave summing that we discussed before is only a very simply method in water animation. The more effect way is Gerstner wave. Although the Gerstner wave is no more complicated than the sine and cosine waves, the undee wave is more like the water. For sine and cosine wave, the radians of wave crests and troughs are even. In fact, wave crests of water waves are sharp and troughs are saponaceous. See figure 2.9; image left is a sine wave and right is two Gerstner waves with different amplitudes and wavelengths.

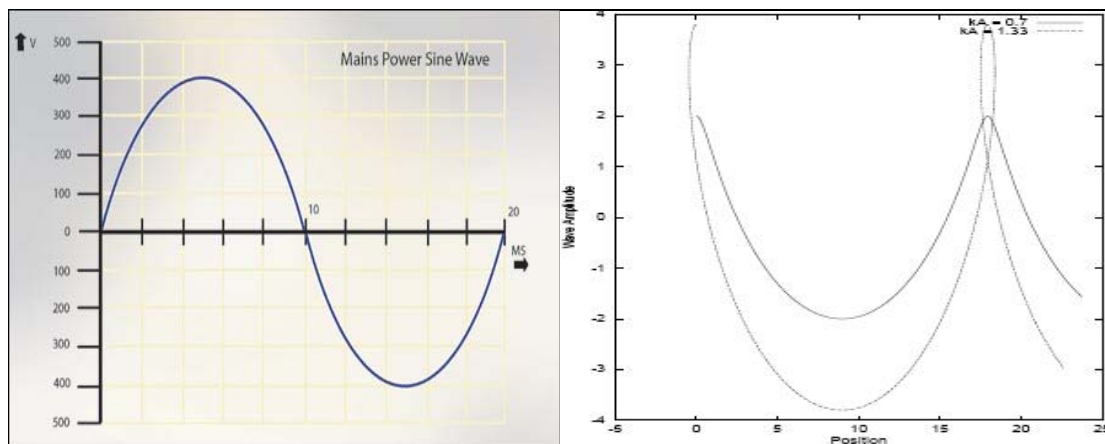


Figure 2. 9 sine wave and Gerstner wave.

2.3.1 Introduction of Fourier transform and Fast Fourier transform.

Fourier transform was first provided by a France mathematician whose name is Fourier. That is why it is called Fourier transform. It is a kind of integral transform. It has a lot of applications in physics; acoustics; statistics; optics; and many other areas. In mathematics, it is a certain linear operator. Here is a brief introduction of Fourier transform:

Fourier analysis was first provided as a tool to analyze physical problems. The contrary transform of it is very easy to get, and the form of inverse transform is almost the same as Fourier transform. [8]; Fourier transform can express some certain functions which fit some given qualification to trigonometric functions (sine or cosine functions), or the linear integration of their integral. In different researching areas, it has a couple of transformations. For example, discrete Fourier transform, continuous Fourier transform and so on. [8]

The Fourier transform in discrete form can be implemented in a very fast way. It is called fast Fourier transform (FFT). FFT owns an important position in simulating water.

2.3.2 Gerstner-wave

The first time that Gerstner-wave was used to water simulating is about 200 years ago, which describes the surface in terms of the motion of individual points on the surface.

We suppose P is a point that on the surface of water. Its position is (x_0, y_0, z_0) when no waves passed by. Then a single wave passed by whose amplitude is A, the point will be displaced to:

$$(x, z) = (x_0, z_0) + Q \cdot A \cdot \sin(w \cdot (x_0, y_0) + wt \cdot t + FI) \quad [1]$$

$$y = A \cdot \cos(w \cdot (x_0, y_0) + wt \cdot t + FI) \quad [7] \quad [2]$$

Amplitude of these waves (A);

Spatial angular frequency (w): quantity is relative with the wavelength L:
 $|w| = 2 \cdot \pi / L$;

Temporal angular frequency (wt): $wt = S \cdot 2 \cdot \pi / L$;

Initiatory phase (FI): the initiatory phase of waves;

Notice that Q is for controlling how cliffy the water is. It works together with the amplitude of the wave. QA should not be the value that $QA < 1$. For the values that $QA < 1$, waves will turn more and more cliffy then QA approaches 1. If QA has values that $QA > 1$, loop forms at the top of water will generated. That's not realistic, here comes the reason, see figure 2.10; [14] [15]

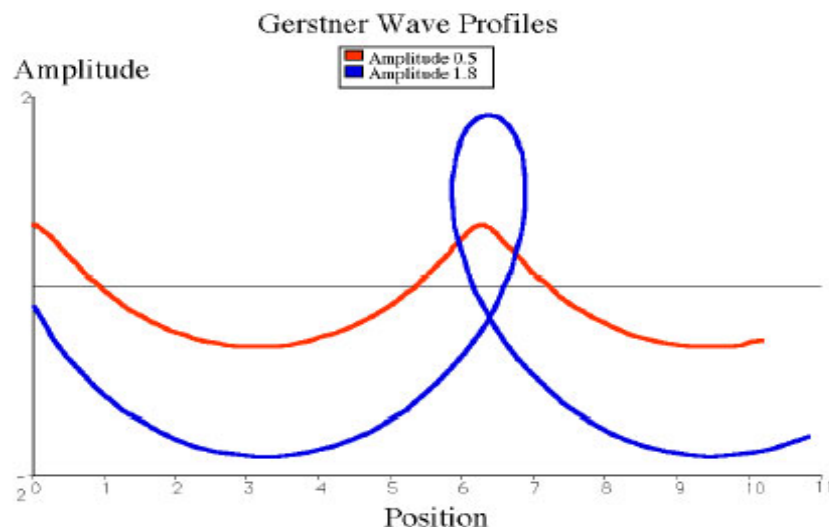


Figure 2. 10 graph of Gerstner wave with different amplitudes

Gerstner wave has the undulating not only in the upright direction but also horizontal. When the amplitude has the value that bigger than 1, for example 1.8 in the figure 2.9 , it will bring loop forms.

2.3.3 FFT method which based on Gerstner wave

Gerstner waves we mentioned above are still very limited. They are only a single

wave horizontally and vertically. It is not enough if you want to render a more effective water to use it in films. Then you can use the FFT method to get a more complex profile by summing a set of waves.

First, we analyze the one dimension situation, check the water motion in a one-dimension region which length is L_s , see figure 2.11 below;

Two preconditions here are need:

- The motion of water in this region has the spatial periodicity with the cycle L .
- We need a set of Gerstner wave summed together to compose the animation. Because of the wave motion in this region has the periodicity, so the spatial cycle L_s should be the integral multiple of wavelengths that summing together. See figure 2.11 below:

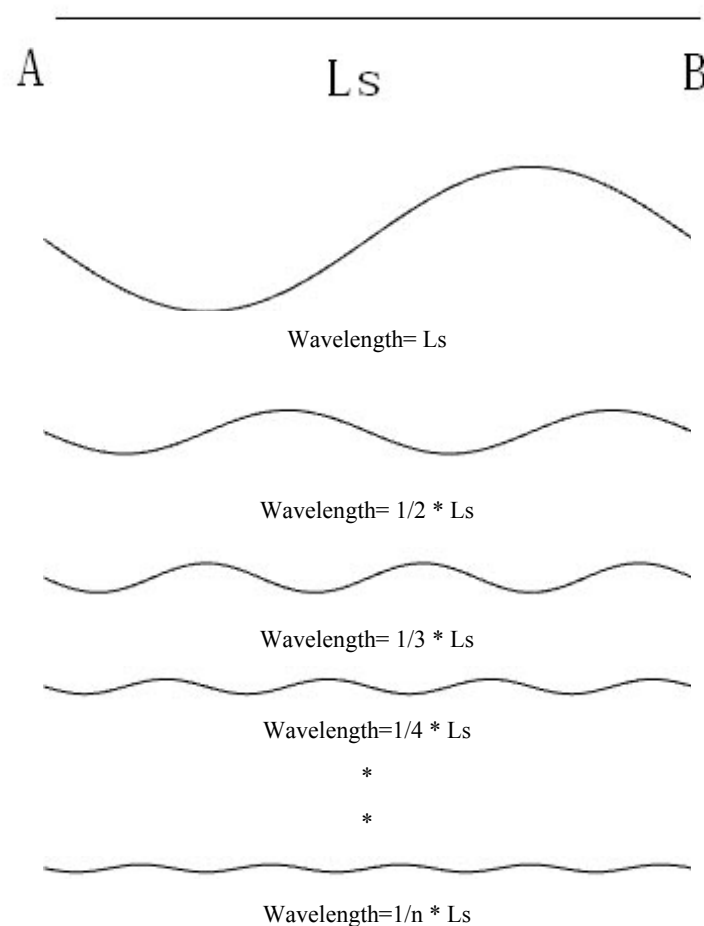


Figure 2. 11 region with length L_s and waves with different wavelengths

For Gerstner wave, then wavelength L is set, the special angular frequency w and temporal angular frequency can be written like this:

$$w=2*\pi / L; \quad [3]$$

$$wt = \text{sqrt} (g*w); [4]$$

Here the $L= L_s / K$; K is plus integer.

Then we come to the academic part of the FFT equation for water:
Summing a set of sine waves together, then we gain a composite wave:

$$y = \sum_{i=1}^n A_i \cos(\omega_i \cdot x - \omega_i t + FI_i) \quad [6]$$

Using the Euler's formula, we can change the formula to the form of Fourier transform:

$$y = \text{re}(\sum_{i=1}^n A_i (\cos(\omega_i \cdot x - \omega_i t + FI_i) + i * \sin(\omega_i \cdot x - \omega_i t + FI_i))) \quad [7]$$

That is:

$$y = \text{re} \sum_{i=1}^n A_i \exp(i * (\omega_i \cdot x - \omega_i t + FI_i)) \quad [8]$$

It can also be written:

$$y = \text{re} \sum_{i=1}^n A_i \exp(-i\omega_i t + iFI_i) \exp(i\omega_i x) \quad [9]$$

Together with the initial phase and amplitude, we can calculate the water wave though Fast Fourier Transform.

After the one-dimensional situation, here comes the two-dimensional part. Pick out a rectangular with the length L_x and width L_y like figure 2.12.

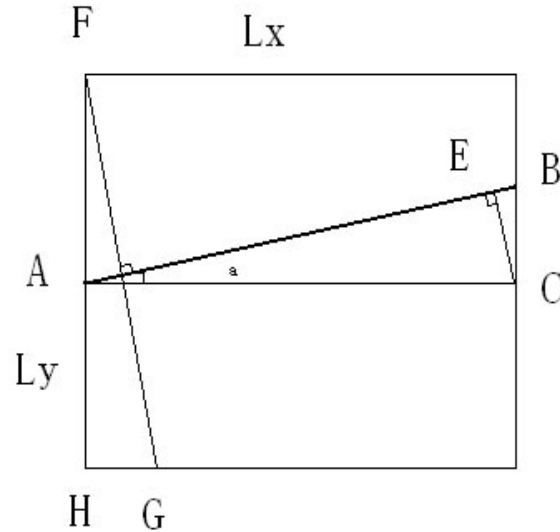


Figure 2. 12 two-dimensional region

The preconditions are almost the same with the one-dimensional:

- The motion of water in this whole region has the spatial periodicity with the cycle L_x in x direction and L_y in y direction;
 - Use a set of Gerstner-Wave summing together to compose the wave animation;
- Consider a sine wave that diffusing along AB in figure 2.12s; if we want this wave

has the spatial periodicity of L_x in the x direction, the difference of phases between point A and point C should be the integral multiple of 2π . Because CE is perpendicular to the wave direction AB, C, E located at the same wave front, they have the same phase. Therefore, the difference of phase of A and E should be integral multiple of 2π . Then the length of AE should be integral multiple of wavelength L .

The length of AC is L_x , so the length of AE is $L_x \cos(a)$, it is integral multiple of wavelength L , so:

$$L_x \cos(a) = k_1 L \quad [10]$$

k_1 is integer here; The wave has the spatial periodicity with cycle L_y in the y direction. We can get:

$$L_y \sin(a) = k_2 L \quad [11]$$

k_1 and k_2 are all integers (can be minus); So for a discretionary, we can get a Gerstner-wave with wavelength L and direction $(\cos(a), \sin(a))$;

When k_1 and k_2 are set, we can get the solution of (10) and (11), so:

$$\tan(a) = k_2 L_x / (k_1 L_y) \quad [12]$$

$$a = \arctan(k_2 L_x / (k_1 L_y))$$

$$a = \pi + \arctan(k_2 L_x / (k_1 L_y))$$

Here we set the bound of a – $\pi < a \leq \pi$. a has two values here, their discrepancy is 180 degree, that means there are two Gerstner-waves on this line, and their directions are opposite. We can calculate the value of wavelength L through a :

$$L = 1 / \sqrt{(k_1/L_x)^2 + (k_2/L_y)^2} \quad [13]$$

Then we can get a couple of spatial angular frequency:

$$w = (2\pi k_1 / L_x, 2\pi k_2 / L_y)$$

$$-w = (-2\pi k_1 / L_x, -2\pi k_2 / L_y)$$

From the relation of diffuse, we can get the temporal angular frequency:

$$w_t = \sqrt{g|w|}$$

So we can get a couple of Gerstner-Wave which were set by (k_1, k_2) from the upwards:

$$Y_1 = A_0 \cos(w \cdot (x, y) + w_t t + \phi_0)$$

$$Y_2 = A_1 \cos(w \cdot (x, y) - w_t t + \phi_1) \quad [14]$$

At last, we pick up several this kind of Gerstner-Wave summing together, the

range of k_1 and k_2 is: $-N/2 < k_1 < N/2$; $-M/2 < k_2 < M/2$; this value range makes us get waves in as more directions as possible. The equation of composite wave is:

$$y = \sum_{k_1=-N/2}^{N/2} \sum_{k_2=-M/2}^{M/2} (A_{0k_1,k_2} \cos(w^*(x,y) + wt^*t + FI0_{k_1,k_2}) + A_{1k_1,k_2} \cos(w^*(x,y) - wt^*t + FI1_{k_1,k_2}))$$

Use Euler's formula here again, we can change the equation to the form of two-dimensional Fourier transform:

$$y = re \sum_{k_1=-N/2}^{N/2} \sum_{k_2=-M/2}^{M/2} (A_{0k_1,k_2} * \exp(i*wt^*t+i*FI0_{k_1,k_2}) + A_{1k_1,k_2} * \exp(-i*wt^*t+i*FI1_{k_1,k_2})) \exp(i*w^*(x,y))$$

This is the final equation to process waves in the whole region through two-dimensional Fourier transform.

3 Physics based water simulating

What we talk about above is based on the common shape of water, and people try to simulate water with wave and present these waves using mathematic equation. People have work out many efficient ways to deal and process equations of wave. FFT is a good example, we present wave in time field. Since it very hard to analyses and solve under time field, we translate it into frequency field.

It is work and can run fast with a nowadays' ordinary computer. This is also have a nice result and present very well when simulate water especially when simulate sea water which means large body of water. However people don't satisfy with this, they need more real water, not a simulating water.

That is why we do research on water and what we try discover. Since water is so real in our days life. We are so familiar with it. Once it a little artifact behaves, it will be enlarge and found out by our sharp eye.

With these reasons, we start on our way to the world of water. The way is directly lead to physics characteristic of water. We will show the mystery side of water and how we simulate it with totally real way. At the end of our way is a new world of water, and you cannot tell whether it is real not artifact.

3.1 Introduction

Here what we will talk about is all base on physic. How to present water is not a problem, people have already found the equation to present water. There are two basic way to present water in mathematic equation.

Lagrange's way: Describing every finite point path, this way can't describe parameters in space directly. This is also not suit for describing transformation of fluid.

Euler's way: Describe all points' parameters at one certain time. The equation is less complex; it can show parameters in space directly. It suit for describing motion of fluid. This is most common way in transformation in hydrodynamics.

The most complete equation to describe phenomena of fluid is N-S equation(NSEs: Navier-Stokes Equation). This equation is follow Newton's second law. N-S equation is popularly used in CFD(Computational fluid dynamics) and fluid simulation. Our physics based water simulate is also use N-S equation to describe the state. N-S equation try to describe the velocity change with time, it follows momentum conservation. The velocity change with time is effected by force, advection, pressure and viscose.

The main problem is how to solve this complex equation efficiently. Mathematicians have work many years on this. Till now, nobody can found the exact solution of N-S equation, this is a mathematic problem worth \$1,000,000 which provided by CMI(Clay Mathematics Institute) in May,2000.

We are not focus on how to solve the equation accurately, our problem is how to solve it use computer since we are computer scientists not mathematicians. How to solve this problem, we will talk about the details in 3.3 Navier-Stokes equation.

Anther problem is how to use Navier-Stokes present water. Great computer scientist have create several ways to do this. The most efficient one is Semi-Lagrange's method. This method use particles to present water, use MAC grid to deal with advection and boundary condition. This is a very good way since MAC grid is very easy to process advection and boundary condition. It's is also very easy to present water with particles.

Water simulate is just one kind of fluid dynamic, there are other fluid dynamic such as fire, smoke, cloud, fog and so on. They are all can be work out with the same equation N-S. Compare with water, they can be simulated by changing some part of N-S equation.

What confuse us is how to solve N-S on computer in a stable and efficiently way. Stam have developed a stable way to solve N-S equation. However, nobody can efficiently work N-S out on a common computer, especially 3 dimensional N-S equation.

We need to work hard on way to simulate water, since it is a long and hard way.

3.2 Previous Work

At first, in order to present fluid animation, [18] Kass solve height field to present water. [19]O'Brien use height field to present the surface of water, and suppose there are many pipes between nodes, he used these pipes to calculate the volume of box.

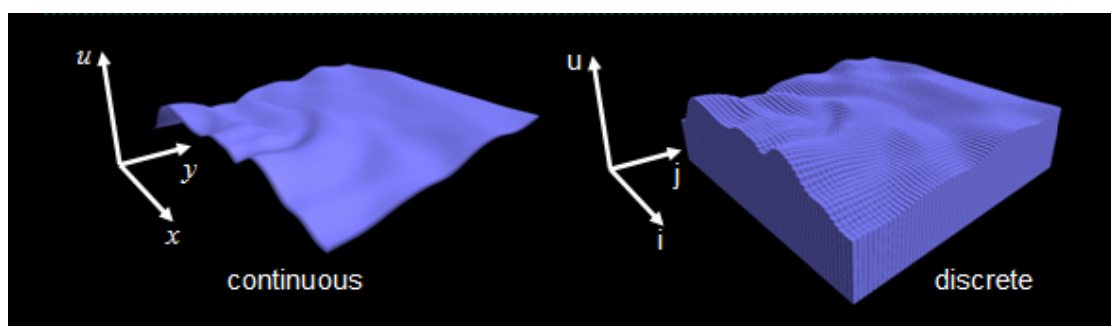


Figure 3. 1 Height field (from SIGGRAPH 2006 Reel Time Fluids in Games)

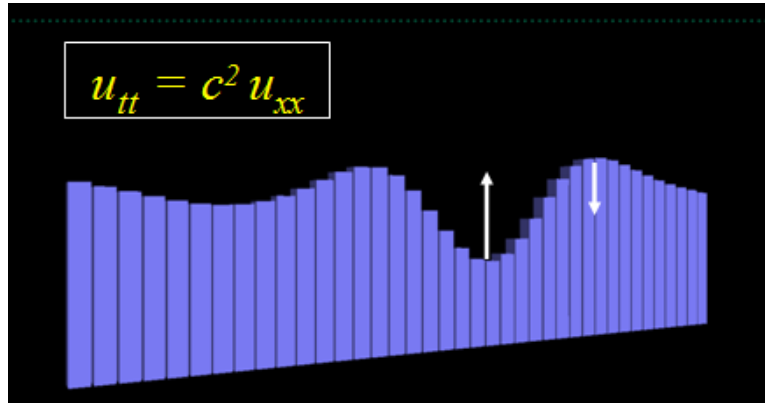


Figure 3. 2 Pipe
(from SIGGRAPH 2006 Reel Time Fluids in Games)

He simulated iterations with objects by add force. The splash was made by particles, which is more simulate with physics. [20] Chen use two dimensional N-S equation to velocity field, then get height field by solving Bernoulli pressure equation to present the surface of fluid. The advantage of using height field is fluid simulating is just 2 dimensional, this avoid the complex calculation of 3 dimensions way. However, the result is not so good.

The first time to use 3 dimensional N-S equation to simulating fluid is Foster[21], they use MAC(Marker and Cells) to solve fluid, but they use an explicit way, time step must satisfy CFL(Courant-Friedrichs-Lewy Condition). [16]Stam use Semi-Lagrange method to solve advection, and use an implicit way. Since then, it is more and more popular using N-S equation to simulate 3 dimensional fluid.

3.3 Basic Mathematics and Physics

In order to simulate physics based water, we need to learn some basic mathematics and physics. Since we are computer scientists, we just need to know how to use mathematics and physics. About the detail, we do not have to prove them.

In the notes that we will use below indicate the time step in the superscript, and the spatial indices in the subscripts.

For example: $P^n = P^{n+1}$ means P at time n equals the P at time $n+\Delta t$.

$P_{n,j} = P_{n+1,j}$ means the P in cell (i,j) equals the P in cell(i+1,j)

$P^n_{i,j} = P^{n+1}_{i+1,j}$ means the P in cell(i,j) at time n equals the P in cell(i+1,j) at time $n+\Delta t$.

(here P is Pressure of cell)

This is used quite often in our thesis, so please remember them in your mind.

3.3.1 Mathematical Background

Basic concept of mathematical

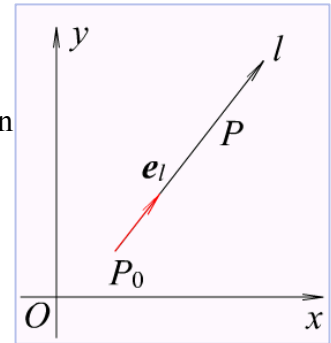
Directional Derivative:

Suppose function $z=f(x,y)$ has definition at the neighbor domain of point $P_0 (x_0,y_0)$, l is a radial line with the start point $P_0 (x_0,y_0)$ on plane xOy , the directional vector is $e_l=(\cos\alpha, \cos\beta)$. Assume $P(x_0+t\cos\alpha, y_0+t\cos\beta)\in U(P_0)$, if the limit :

$$\lim_{t\rightarrow 0^+} \frac{f(x_0+t\cos\alpha, y_0+t\cos\beta)-f(x_0,y_0)}{t}$$

exist, then we call this limit is the directional derivative of function $f(x,y)$. Marked with symbol:

$$\frac{\partial f}{\partial l}\bigg|_{(x_0,y_0)} = \lim_{t\rightarrow 0^+} \frac{f(x_0+t\cos\alpha, y_0+t\cos\beta)-f(x_0,y_0)}{t}.$$



Directional derivative is the change speed of function $f(x,y)$ at point $P_0 (x_0,y_0)$ along direction l .

If function $z=f(x,y)$ can be differentiated, the directional derivatives of this function at this point along any direction $l (e_l=(\cos\alpha, \cos\beta))$ are exist, and:

$$\frac{\partial f}{\partial l}\bigg|_{(x_0,y_0)} = f_x(x_0,y_0)\cos\alpha + f_y(x_0,y_0)\cos\beta.$$

Gradient:

Suppose function $z=f(x,y)$ has continually first-order partial derivative at the plan region D , so for every point $P_0(x_0, y_0)\in D$, there is a certain vector $f_x(x_0, y_0)\mathbf{i}+f_y(x_0, y_0)\mathbf{j}$, this vector is called the gradient of this function $f(x,y)$ at point $P_0(x_0, y_0)$, it marked as $\mathbf{grad}f(x_0, y_0)$, that is: $\mathbf{grad}f(x_0, y_0)=f_x(x_0, y_0)\mathbf{i}+f_y(x_0, y_0)\mathbf{j}$.

The relationship of Directional Derivative and Gradient:

If function $f(x,y)$ can be differentiated at point $P_0(x_0, y_0)$, $e_l=(\cos\alpha, \cos\beta)$ has the same direction with l , so:

$$\begin{aligned} \frac{\partial f}{\partial l}\bigg|_{(x_0,y_0)} &= f_x(x_0,y_0)\cos\alpha + f_y(x_0,y_0)\cos\beta, \\ &= \mathbf{grad}f(x_0, y_0) \cdot \mathbf{e}_l \\ &= |\mathbf{grad}f(x_0, y_0)| \cdot \cos(\mathbf{grad}f(x_0, y_0) \wedge \mathbf{e}_l). \\ \frac{\partial f}{\partial l}\bigg|_{(x_0,y_0)} &= |\mathbf{grad}f(x_0, y_0)| \cdot \cos(\mathbf{grad}f(x_0, y_0) \wedge \mathbf{e}_l). \end{aligned}$$

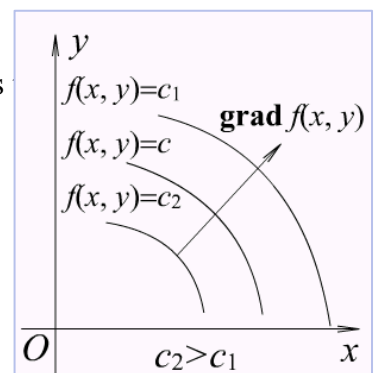
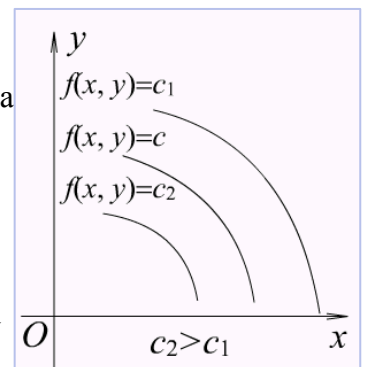
The gradient of function at one point is such a vector, this vector has a direction of the largest directional derivative, its module is the value of directional derivative.

For function $z=f(x,y)$, the curve $f(x,y)=c$ on plane xOy is called contour line of function $z=f(x,y)$.

If f_x, f_y is not zero at the same time, then the normal vector at any point on contour line $f(x,y)=c$ is:

$$\mathbf{n} = \frac{1}{\sqrt{f_x^2(x_0,y_0) + f_y^2(x_0,y_0)}} (f_x(x_0,y_0), f_y(x_0,y_0)).$$

This shows the direction of gradient $\mathbf{grad}f(x_0, y_0)$ is the same as normal vector of the same point on the contour line.



$$\text{grad}f(x_0, y_0) = \frac{\partial f}{\partial n} \mathbf{n}.$$

Divergence:

Gaussian formula:

Suppose a closed spatial region Ω is made up by continually closed surfaces, function $P(x, y, z)$, $Q(x, y, z)$, $R(x, y, z)$ on region Ω has first step differential derivative, then we have:

$$\iiint_{\Omega} \left(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z} \right) dv = \oint_{\Sigma} P dy dz + Q dz dx + R dx dy,$$

Or

$$\iiint_{\Omega} \left(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z} \right) dv = \oint_{\Sigma} (P \cos \alpha + Q \cos \beta + R \cos \gamma) dS,$$

This can be written for short:

$$\iiint_{\Omega} \left(\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z} \right) dv = \oint_{\Sigma} v_n dS,$$

Here $v_n = \mathbf{v} \cdot \mathbf{n} = P \cos \alpha + Q \cos \beta + R \cos \gamma$.

Divergence:

Suppose some vector field is given by $\mathbf{A}(x, y, z) = P(x, y, z)\mathbf{i} + Q(x, y, z)\mathbf{j} + R(x, y, z)\mathbf{k}$, and P, Q, R has first order continually partial derivative, so we call

Is divergence of the vector field \mathbf{A} , and it's marked as $\text{div} \mathbf{A}$, that is:

$$\text{div} \mathbf{A} = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}.$$

Curl:

Suppose scale the closed curve to a certain point, then the area Δs closed by curve L will be reduced, $\oint_L \vec{A} \cdot d\vec{l}$ will reduce also, the ratio of these two has a limit value, it marked as:

$$\lim_{\Delta s \rightarrow 0} \frac{\oint_L \vec{A} \cdot d\vec{l}}{\Delta s}$$

That is the limit of average stream of circle per union area.

$$\text{rot} \vec{A} = \nabla \times \vec{A} = \lim_{\Delta s \rightarrow 0} \frac{\oint_L \vec{A} \cdot d\vec{l}}{\Delta s} \hat{n}$$

This is called the rot or rotation of vector field $\vec{A}(\vec{x})$

Temporal Discretization and Spatial Discretization:

When we try to solve some equation on computer, what should we do first?

We need to convert mathematic problems to computer compatible problems, so we can solve them with powerful computer. Since our computer just can solve discrete numbers, we have to convert every thing into discrete things.

For exam, if we want to simulate water, we need to discrete time and spatial, that is because our N-S equation contains temporal variables and spatial variables. We will talk about these variables in detail later in section 3.4. Now we know the problem,

how can discrete them?

Temporal Discretization [22]:

We look at some possible discretizations for the time derivatives in the above convection equation. The equations below are called semi-discrete because we have discrete only the time derivatives.

- forward Euler:
$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + \vec{V}^n \cdot \nabla \phi^n = 0$$
- backward Euler:
$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + \vec{V}^{n+1} \cdot \nabla \phi^{n+1} = 0$$

The forward Euler and backward Euler are both first order accurate, meaning that the error in the discretization is $O(\Delta t)$.

Spatial Discretization:

Spatial discretization is much easier than you think usually we discretize spatial into a grid. For example, here is a grid in 2 dimensions. The center of cell is pressure, the edge is velocity.

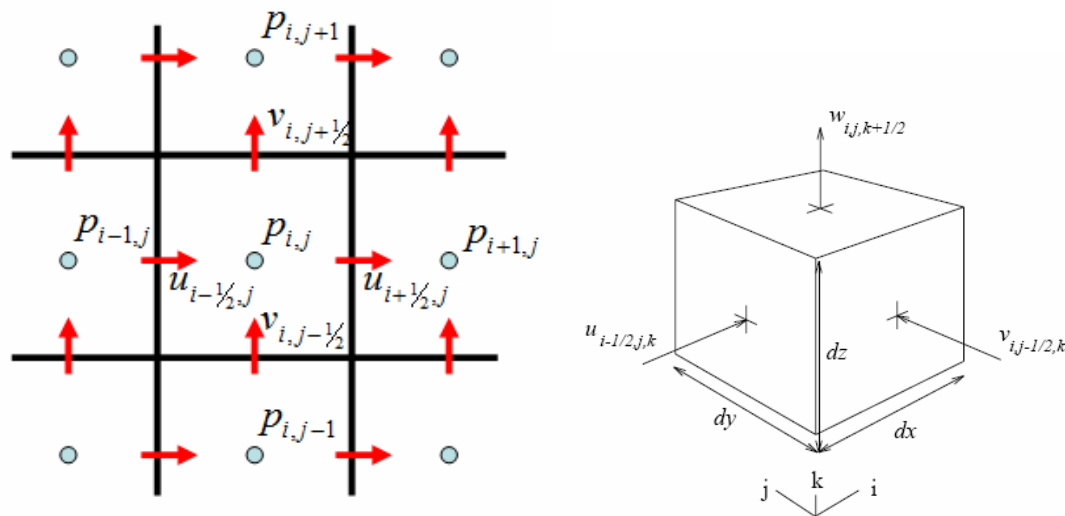


Figure 3.3 left one: MAC Grid in 2D

(From SIGGRAPH 2006 Slides Fluid Simulation for computer Animation The basics of fluids); right one: Location of staggered velocity components on a typical cell(from [16])

Why we need a MAC grid? What can we get from it?

We convert the continually world to a MAC grid, then we can process them with a array on computer. When we have discrete grid, we reduce the storage, calculation and that is also easy to control. In 3d, we have a single cell shown the right picture. The pressure is in the center and velocity on the surface. This is good for

add force, advection and dealing with boundary condition.

Vector Calculus Operators Used in Fluid Simulation[23]:

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

Figure 3. 4 Vector Calculus Operators

Symbol ∇ (often pronounced “del”), this is well known Laplace operator. ∇ stands for one kind of operator, which is partial derivative. If we do partial derivative to a scalar, that means solve gradient of this scalar. For example: ∇P , here P is the pressure, which is a scalar. However, ∇P is vector. Since we do partial derivative to a vector, that means solve divergence of a vector. For example: $\nabla \cdot \mathbf{u}$, here \mathbf{u} is the velocity, which is a vector. However, $\nabla \cdot \mathbf{u}$ is scalar. ∇^2 is Laplacian operator. If ∇^2 equals zero, that is $\nabla^2 \Phi = 0$. This is a Laplacian equation. If ∇^2 equals constant, that is $\nabla^2 \Phi = c$. That is Poisson equation, it is also called pressure equation. We will use it later.

In this table, there is a column named Finite Difference Form, here δx , δy is the width and height of each grid. When we deal with our water simulating, we usually use a square MAC grid. That means we have $\delta x = \delta y$. So we have a ∇^2 Laplacian simplifies to:

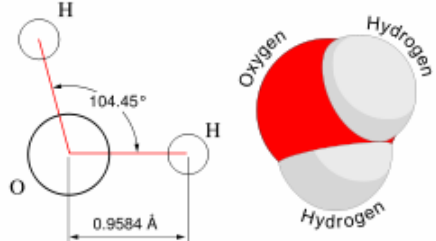
$$\nabla^2 p = \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{(\delta x)^2}$$

This is very useful in our later calculation, this is not only reduce the solving equation, but also good for array storage and union process.

3.3.2 Physics Background

Basic introduction of water:

Chemical and physical properties

Water	
 <p>Water is the base of human life, and an abundant compound on the earth's surface.</p>	
Information and properties	
Systematic name	water
Alternative names	aqua, dihydrogen monoxide, hydrogen hydroxide, (more)
Molecular formula	H ₂ O
Molar mass	18.0153 g/mol
Density and phase	0.998 g/cm ³ (liquid at 20 °C) 0.92 g/cm ³ (solid)
Melting point	0 °C (273.15 K) (32 °F)
Boiling point	100 °C (373.15 K) (212 °F)
Specific heat capacity	4.184 J/(g•K) (liquid at 20 °C)

Water has a chemical formula H_2O : one molecule of water has two hydrogen atoms covalently bonded to a single oxygen atom. Water is tasteless, odourless liquid at ambient temperature and pressure.

Water sticks to itself(cohesion) because it is polar. Water also has high adhesion properties because of its polar nature.

Water has a high surface tension caused by the strong cohesion between water molecules.

Water can be ice, vapor and liquid. Water is in sea ocean, rivers, lakes and anywhere. Ice is usually on mountains and snow on the ground when winter comes. Vapor is always goes up, you can see it in the sky, such as cloud.

Water has so many properties, however what we care most are only four characteristics:

- viscosity $\nu = 10^{-6} \frac{m^2}{s}$
- surface tension $\sigma = 7 \cdot 10^{-4} \frac{kg}{s^2}$
- gravitational constant $g = 9.81 \frac{m}{s}$
- density $\rho = 1000 \frac{kg}{m^3}$

Figure 3. 5 Water Properties

These are four items in N-S equation, we will talk about them in detail in section 3.4 N-S equation.

Fluid dynamic:

What is fluid? May be you can say something like water is fluid. That's right. The exactly definition is fluid can't resist shape changing under the force(the volume don't change). Fluid have a certain volume and difficult to compress. In 1826 Robert Brown found that farina move randomly on the surface of water.

Fluid has viscosity inside. When two neighbor fluid move against each other, there be inside friction. The friction inside fluid was affirmed by Newton(I.Newton, 1687). 100 years later, this was proved by Coulomb(C.A. Coulomb, 1784).

Coulomb hang one piece of thin circle board in liquid with a metal string. Release the circle board after turning some angular. The board will swing with the force metal string. Since liquid has viscosity inside, the swing will be attenuated until stopping. Coulomb test three kinds of board, ordinary board, was board and sand board. The result is very similar.

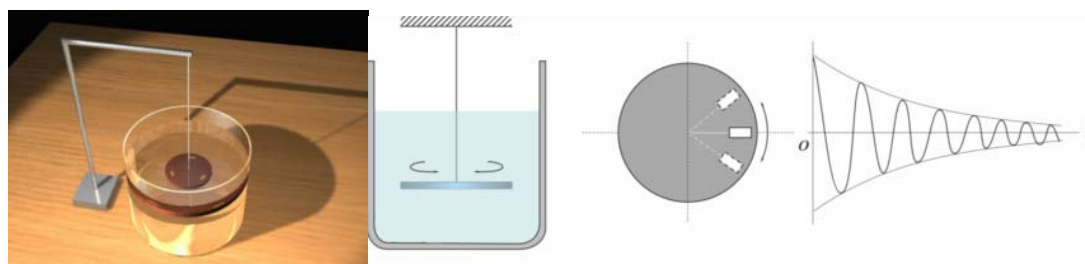


Figure 3. 6 Coulomb's experiment equipment and the attenuation result
(from ShangHai University course "fluid dynamics" webpage:

<http://em.sjtu.edu.cn/jingpin/ziyuan/jiaoan.htm>

The kinds of board has the same time of attenuation. Coulomb make a conclusion:
The reason of attenuation is not the friction between board and liquid, but the friction
inside of liquid.

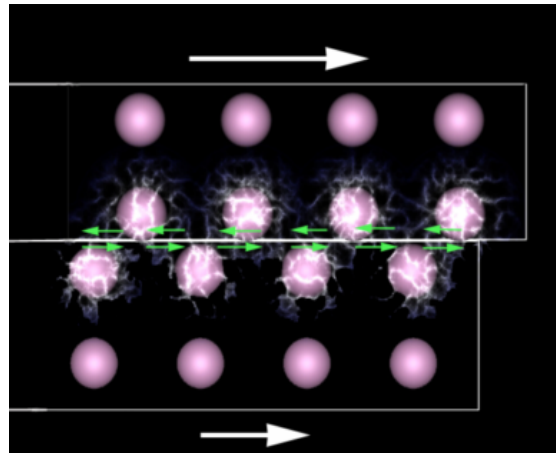


Figure 3. 7 molecule cohesion

(from ShangHai University course "fluid dynamics" webpage:

<http://em.sjtu.edu.cn/jingpin/ziyuan/jiaoan.htm>

The friction inside liquid is the appearance of the changing between the
molecule's cohesion and molecule's momentum of two layer of fluid.

When two layer liquid move against each other, the distance of two layer's
molecule become larger and larger. This course the attractive force, this kind of force
called molecule's cohesion.

There are two ways of mathematic to analysis fluid. They are Euler's way and
Lagrange's way. Euler's is one kind of method based on grid, it describe all the
parameters of all mass point at one certain time. You can fix your point in space, then
measure stuff as it flows past. Lagrange's way is based on particles. It describe some
mass points' trace independently, so it is more complex. This method treat the world
like a particle system, it label each speck of matter, track where it goes(how fast,
acceleration).

Compare:

Lagrange's way	Euler's way
Describe limit mass points' traces in the space, expression is complex	Describe all mass points parameters at one time, expression is easy
It can't tell spatial parameters directly	It shows spatial parameters directly
Not suit for characteristic of fluid movement changing	Suit for characteristic of fluid movement changing
This is an import method	The most common way in fluid dynamics

Figure 3. 8 Comparison Lagrange's way and Euler's way

3.4 Navier-Stokes Equation

This is the most important part of fluid dynamic. Navier-Stokes is derived from Newton's second law, that is very famous $F=ma$. I think you can understand it without my explanation. We use many pages to describe and explain N-S equation. Maybe you will feel confused in this part, don't worry about it. You just need to know how to use it in our water simulating, since we are not experts in mathematics and physics. Whatever we will explain as clearly as possible. We will also keep it easy. If you still feel confused and don't know what this part is talking about, you can return to the physics and mathematics part or learn some more basic knowledge about it. If you are really interested in water simulation, I think you can learn some from CFD(Computational fluid dynamics). Let us start our challenge.

3.4.1 N-S Equation introduction

[wikipedia webpage: http://en.wikipedia.org/wiki/Navier-Stokes_equations]

The Navier-Stokes equation is named after Claude-Louis Navier and George Gabriel Stokes. It can describe the motion of fluid substances such as gases and liquids. These equations establish that changes in momentum in infinitesimal volumes of fluid are simply the sum of dissipative viscous forces(similar to friction), changes in pressure, gravity, and other forces acting inside the fluid. The equation was first derived from Newton's second law in several hundreds years ago. This is not hard to derive N-S equation, the problem is how to solve this equation. Clay Mathematics Institute used a \$1,000,000 for prize to encourage mathematicians to work hard on this. However this problem doesn't solve accurately, people only can use computer to simulate an imprecise result.

N-S equations are one of the most useful sets of equations, because they describe the physics of a large number of phenomena of academic and economic interest. They may be used to model weather, ocean currents, water flow in a pipe, flow around an airfoil(wing), and motion of stars inside a galaxy. As such, these equations in both full and simplified forms, are used in the design of aircraft and car, the study of blood flow, the design of power stations, the analysis of the effects of pollution, etc. Coupled with Maxwell's equations they can be used to model and study magnetohydrodynamics.

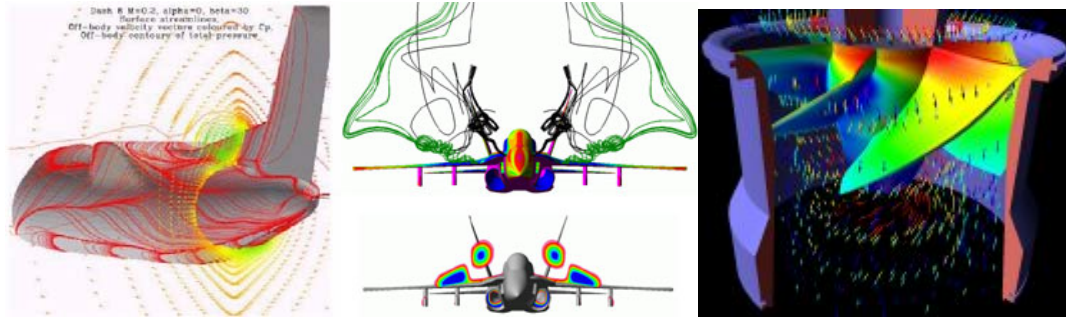


Figure 3. 9 Navier-Stokes application (From google pictures)

N-S equations are not so widely used, because of complex calculation. Today people only can solve N-S on super computer, I think in the further, it will be more and more widely used in many differently areas. With the power of N-S, it will serve people quit well.

3.4.3 N-S Equation Expression

Symbols

We use consistent mathematical notation through out the chapter. *Italics* are used for variables that represent scalar quantities, such as pressure, p . **Boldface** is used to represent vector quantities, such as velocity, \mathbf{U} .

\mathbf{U} : velocity with components(u,v,w)

ρ : fluid density

P : pressure

\mathbf{G} : acceleration due to gravity or animator

μ : dynamic viscosity

\mathbf{U} is the fluid's velocity. \mathbf{U} is a vector, so it has a direction and magnitude. The velocity of a MAC grid is always present as 2d: $\mathbf{U}(u,v)$, for 3d: $\mathbf{U}(u,v,w)$. u is the part magnitude of \mathbf{U} in x axis direction, v is stands for y direction and w for z direction.

ρ is fluid density, water has a density of 1000kg/m^3 .

P is pressure of water. We usual present pressure in the center of cell.

μ is dynamic viscosity. This is not so hard to understand, for example, molasses and maple syrup flow slowly, but rubbing alcohol flows quickly. We say that thick fluids have high viscosity. Viscosity is used to measure how resistive the flow of fluid.

\mathbf{G} is the internal force. Here \mathbf{G} stands for gravity, because our water just have a gravity work on it.

Two important Assumption

Incompressible

Real fluids are compressible, but we have told you that fluid's volume doesn't change when pressure increase. That's not correct, liquids change their volume as well as

gases, otherwise there would be no sound underwater. There are micro gap in fluids' molecules, this make fluid compressible. However this is nearly irrelevant for animation. The gap is so small, that doesn't affect our water simulation.

When we simulate water, we have an incompressibility assumption. This will give our simulation great convenient. What's more we get anther equation, that makes our Navier-Stokes equations could solved.

What does incompressible mean? For example, when you fix your eyes on a volume of space, the volume contain some water. When some water fluid, there is must be the same water fluid out. This means the divergence is zero:

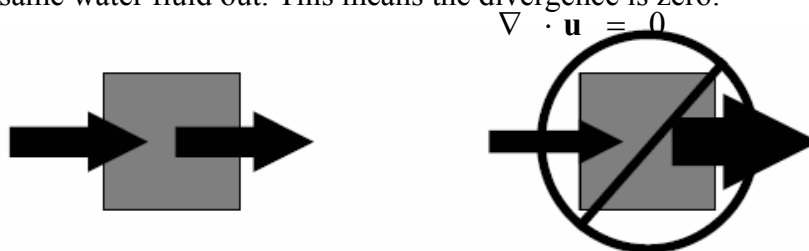


Figure 3. 10 Incompressible equation(divergence free) from: Mark Harris/
GPGPU tutorial Siggraph 04

Homogenous:

Homogenous means density of our fluid is a constant. That is ρ equals const. When the ρ change with space and time. If ρ change with time and space, that is not a water simulate but smoke, fire or anything else. Our water has a continuously ρ , and don't change with t grows.

This is also another important assumption. This makes our Navier-Stokes much more easily to work out.

2D Navier-Stokes and 3D Navier-Stokes Equations

Our water simulation is base on physics. Our simulation equations contain two parts, momentum equation and incompressibility condition equation(mass conservation equation).

The momentum equation is derivative from famous Newton's Second law: $F=ma$. This is a complex equation, we will explain by four steps: advection, diffuse, projection and force. About the incompressibility equation, since it is easy one, there is no explanation.

Incompressibility condition: $\nabla \cdot \mathbf{u} = 0$

Momentum Equation: $\mathbf{u}_t = -(\mathbf{u} \cdot \nabla) \mathbf{u} + \nu \nabla^2 \mathbf{u} - \frac{1}{\rho} \nabla p + \mathbf{f}$
 \mathbf{U} : velocity P : pressure \mathbf{F} : force ν : viscosity ρ : density

Notation: $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$

$$\nabla^2 = \nabla \cdot \nabla$$

Expanding the partial derivative operator:

Conservation of mass:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

Conservation of momentum:

$$\begin{aligned} \frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} &= -\frac{\partial p}{\partial x} + g_x + \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \\ \frac{\partial v}{\partial t} + \frac{\partial vu}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} &= -\frac{\partial p}{\partial y} + g_y + \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2} \right) \\ \frac{\partial w}{\partial t} + \frac{\partial wu}{\partial x} + \frac{\partial wv}{\partial y} + \frac{\partial w^2}{\partial z} &= -\frac{\partial p}{\partial z} + g_z + \nu \left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} \right), \end{aligned}$$

Here u, v, w is velocities in axis x, y, z direction; P is local pressure; G is the gravity, this can be any internal force work on the fluid; V is viscosity.

3.4.4 Get Momentum Equation from Newton's Second Law

Our N-S equation is just an expanding of $F=ma$. It's is more complex and contain many items which can affect our fluid. We will add them and explain one by one.

In order to simulate water, we modeling water as many blobs. Every blob has a mass m , a volume V , velocity \mathbf{u} . This is kind of modeling is very suit for our intuitively simulating.

Du/Dt is the acceleration of blob, and then we have:

$$m \frac{D\mathbf{u}}{Dt} = \mathbf{F} \quad (\text{here } \mathbf{u}, \mathbf{F} \text{ with a } \mathbf{v} \text{ on its head is stands for vector})$$

Since we have an easy form $f=ma$, why we need a partial derivative equation. That is because we don't have an acceleration in our system. Instead, we just only have u and t , so use a partial derivative form. U partial derivative for time t is acceleration. Newton's second law is a classic physics' formula, it is widely used in so many field. Today we will use it to create our Naiver-Stokes. First, I want to tell you, our derivate is not accurate in mathematics. This kind of derivate is just based on physics. However it is much easier to understand it in physics way.

Forces on Fluids

We still use the blob(a water blob). When a blob of water float in front of you, it

floats in the air. That means no force work on it. On the earth, there are gravity exist. The blob of water goes down by gravity. This kind of gravity has a acceleration g , so the gravity work on blob is mg . This also can be treated as a body force. Add this gravity to our Newton's second law.

$$m \frac{D\mathbf{u}}{Dt} = m\mathbf{g} + \mathbf{K}$$

Since our water blob is not exist lonely, it exerts contact forces on its neighboring blobs. This will be transferred by internal pressure.

Pressure

The pressure is one kind of normal contact force. Pressure is created by molecules' collision. When molecules move around actively, this will create great pressure. As the temperature goes high, our fluid becomes hot. The more temperature it has, the higher pressure it create.

When pressure works, our blobs push against each other. When the pressure around are all the same, there is no force. We present this with gradient.

We multiply the pressure gradient with the volume. This is much like multiply our pressure with normal vector over the blob's surface. Pressure always work on the surface when interact with neighbors.

$$m \frac{D\mathbf{u}}{Dt} = m\mathbf{g} - V\nabla p + \mathbf{K}$$

Viscosity

Viscosity is little strange to understand, you can imagine our blob is not fluid, but a solid blob. When other blobs pass through, there is contact friction. This is the sticky force(viscosity).

In our water simulating, we just want our velocity slow down. A water can never be keep on flowing all the time. We control water, let the velocity field be diffused.

$$m \frac{D\mathbf{u}}{Dt} = m\mathbf{g} - V\nabla p + V\mu\nabla^2\mathbf{u}$$

The Continuum Limit

Since we have the rudiment of N-S equation, we need to improve it. If we do nothing about the equation, we will get $F=ma$ the same as $0=0$. We divide both side of the equation by mass, then we have:

$$\frac{D\mathbf{u}}{Dt} = \mathbf{g} - \frac{V}{m}\nabla p + \frac{V}{m}\mu\nabla^2\mathbf{u}$$

Our fluid density is $\rho = m/v$, so we have:

$$\frac{D\mathbf{u}}{Dt} + \frac{1}{\rho} \nabla p = \mathbf{g} + \frac{\mu}{\rho} \nabla^2 \mathbf{u}$$

Then we get our Navier-Stokes equation, this is the standard form of the equation. We will use this form to solve our velocity field.

3.4.5 Solve Equation Gradually.

The N-S equations can be solve accurately, this is useful in CFD. The help of super computer can do the huge calculation. However, we are feel interested in the evolution of the flow over time, we need an incremental numerical solution. This solution is now possible to solve incrementally by using numerical integration techniques.

Since N-S contain four items, we must divide it into parts. Then solve N-S equation with simple forms gradually. This method was created by Stam [16] in 1999. Stam's method is one kind of stable fluids technique. In 1996 Nick foster and Dimitrix Metaxas[24] also develop one method to solve N-S equation. They discrete both eth equations and the environment that they want to model. However their method has a big problem: unstable. With the time step grows big, the computed solution will be unbounded. That's because their used explicit method(Euler method). This method cost low computation, but unstable. However Stam's method used implicit(Backwar Euler method), this method can perfectly solve the unstable problem though it was more expensive.

Stable is very important in our simulation. If the result is unstable our water will not correct. All we have will change to be nothing. The next section, we will show you how to keep our fluid stable.

Stability of a Numerical Method

Suppose we have partial derivative equation : $du/dt = -\lambda u$ ($\lambda > 0$),

initial condition: $u(0) = 1$. The exact solution is $u = e^{-\lambda t}$.

When we use Euler method, we can have an explicit discrete equations:

$$\begin{aligned} (u_{i+1} - u_i)/\Delta t &= -\lambda u_i, u_i \approx u(i\Delta t) \\ u_{i+1} &= (1 - \lambda\Delta t) u_i, u_0 = 1 \\ u_i &= (1 - \lambda\Delta t)^i \end{aligned}$$

When we use Backward Euler method, we get implicit equations:

$$\begin{aligned} (u_{i+1} - u_i)/\Delta t &= -\lambda u_{i+1} \\ u_{i+1} &= (1 + \lambda\Delta t)^{-1} u_i, u_0 = 1 \\ u_i &= (1 + \lambda\Delta t)^{-i} \end{aligned}$$

In general the Backward Euler way will cost more calculation than explicit method.

If the computed solution bounded, we can say this is a stability method. Let's the results of two method.

For explicit: $u_i = (1 - \lambda \Delta t)^i$, this is stable when $|1 - \lambda \Delta t| \leq 1$, that is $\Delta t \leq 2/\lambda$. When λ is large, time step will be small. If time step is large, it will become unstable.

For implicit: $u_i = (1 + \lambda \Delta t)^{-i}$, it is stable when $|1 + \lambda \Delta t| \geq 1$, that is unconditional stable! That's what we needed. Large time step is also stable.

The Helmholtz-Hodge Decomposition

Ordinary vector shows that any vector v that can be decomposed into several vectors whose sum is v . For example, a vector can be presented as $v = (x, y)$ in Cartesian grid. We also can write this vector as $v = xi + yj$, here i and j are unit vectors aligned to the x axes and y axes.

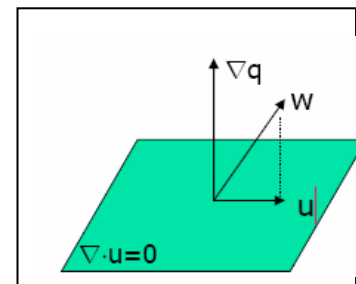
We will use the same way to decompose a vector into a sum of several vectors. We not only decompose vectors, but also vector field. This is what we really need. Assume there is a plan in space, the region of this plan is D . This plane boundary ∂D is smooth, that mean our region is differentiable). The normal direction is n .

Helmholtz-Hodge Decomposition Theorem

Any vector field w on region D can be uniquely decomposed into form:

$$\mathbf{W} = \mathbf{u} + \nabla p$$

Here u is divergence free ($\nabla \cdot u = 0$) and is parallel to ∂D , p is a scalar field; When u is parallel to boundary, $u \cdot n = 0$. N is the normal vector.



How to proof this is a very complex mathematic problem, we just use Helmholtz-Hodge decomposition without proof. You can find detail and a proof about this theorem in Section 1.3 in Chorin and Marsden 1993.

This is the most important part of solving N-S equation. This decomposition change our velocity field into a divergence free field and the gradient of a scalar field. With this, we get **two powerful realizations** [23].

First Realization

There are three computations to calculate the velocity field: advection, diffusion, and force. After these three computations, we will get a new velocity field. This field \mathbf{W} is nonzero divergence. What we really need in N-S equation is a divergence-free velocity. At this moment, we can use Helmholtz-Hodge Decomposition. The divergence-free velocity can be corrected by subtracting the gradient of the pressure field.

$$\mathbf{U} = \mathbf{w} - \nabla p$$

Second Realization

How can get the pressure field? We still can use $\mathbf{W} = \mathbf{u} + \nabla p$. Helmholtz-Hodge decomposition is so powerful that if we add partial operator to both sides of this equation, we

get:

$$\nabla \cdot \mathbf{W} = \nabla \cdot (\mathbf{u} + \nabla p) = \nabla \cdot \mathbf{u} + \nabla^2 p.$$

Our N-S equations have a mass conservation equation $\nabla \cdot \mathbf{u} = 0$, then we get:

$$\nabla \cdot \mathbf{W} = \nabla^2 p,$$

This is a Poisson equation (we have already talk about it at 3.3.1.3) of pressure field. This is Poisson-pressure equation in physics. If we can get the velocity field \mathbf{W} , then we can calculate our pressure field by Poisson-pressure equation. Since we have p , we can get our final \mathbf{u} by using $\mathbf{W} = \mathbf{u} + \nabla p$.

Now it is time to find a way to compute \mathbf{W} . If we want to project a vector \mathbf{w} to a unit vector \mathbf{i} , we can using dot product. The projection can be presented as $\mathbf{w} \cdot \mathbf{i} = |\mathbf{w}| \cdot |\mathbf{i}| \cdot \cos \theta$. The dot product is one kind of projection operator for vectors which project themselves onto some directional vector.

Now, we use the Helmholtz-Hodge Decomposition to define a Projection operator \mathbf{P} , \mathbf{P} is a such a kind of projection, which Projects a vector field \mathbf{W} onto its divergence-free component, \mathbf{u} .

That means $\mathbf{P}(\mathbf{W}) = \mathbf{P}(\mathbf{u}) = \mathbf{u}$. If we apply \mathbf{P} to $\mathbf{W} = \mathbf{u} + \nabla p$, we can get:

$$\mathbf{P}(\mathbf{W}) = \mathbf{P}(\mathbf{u}) + \mathbf{P}(\nabla p).$$

Since $\mathbf{P}(\mathbf{W}) = \mathbf{P}(\mathbf{u}) = \mathbf{u}$, we have $\mathbf{P}(\mathbf{W}) - \mathbf{P}(\mathbf{u}) = \mathbf{P}(\nabla p) \rightarrow 0 = \mathbf{P}(\nabla p)$.

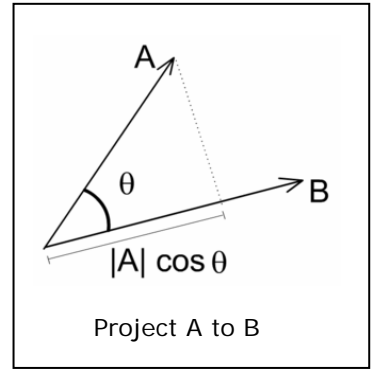
We apply projection operator to both side of N-S equation.

$$\mathbb{P} \frac{\partial \mathbf{u}}{\partial t} = \mathbb{P} \left(-(\mathbf{u} \cdot \nabla) \mathbf{u} - \frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F} \right)$$

Because \mathbf{u} is divergence free, then $\mathbf{P}(\partial \mathbf{u} / \partial t) = \partial \mathbf{u} / \partial t$, and $\mathbf{P}(\nabla p) = 0$. Our N-S has a form:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P} \left(-(\mathbf{u} \cdot \nabla) \mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{F} \right)$$

Now we can solve our Navier-Stokes equation. At the right side of the equation, there are only three items. We will solve N-S with three steps: advection (A), diffusion (D) and add force (F). We already have our strategy, let review how the steps we have done. First, we add force to our equation, here we will use a gravity which always goes down. Second we advect our water(3.4.5.3). Third we diffuse the velocity field. The last step is projection. After projection, we will get a nonzero divergence velocity field \mathbf{W} , then we use $\nabla \cdot \mathbf{W} = \nabla^2 p$ to calculate out our pressure p . Since we have the pressure field, we work out $\mathbf{u} = \mathbf{W} + \nabla p$. The \mathbf{u} is found.



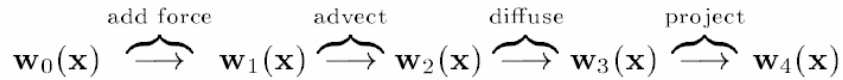


Figure 3. 11 Stam[16] “Stable Fluid “ chapter: Method of Solution

Advection

The Material Derviative:

Suppose we have fluid moving in a velocity field \mathbf{u} . This fluid carry some quality q . In space position \mathbf{x} , the fluid at \mathbf{x} has a quality $q(\mathbf{x},t)$. How fast does the fluid change?

That is our material derivative Dq/Dt . By chain rule we can get:

$$Dq(\mathbf{x},t)/Dt = \partial q/\partial t + \partial q/\partial \mathbf{x} \cdot d\mathbf{x}/dt = \partial q/\partial t + \nabla q \cdot \mathbf{u}$$

That is :

$$Dq/Dt = \partial q/\partial t + \mathbf{u} \cdot \nabla q$$

In three dimensions:

$$Dq/Dt = \partial q/\partial t + \mathbf{u} \cdot \nabla q = \partial q/\partial t + u \cdot \partial q/\partial x + v \cdot \partial q/\partial y + w \cdot \partial q/\partial z$$

Suppose our fluid has a color variable (RGB vector) \mathbf{C} , then we have a quality \mathbf{C} material derivative:

$$D\mathbf{C}/Dt = \partial \mathbf{C}/\partial t + \mathbf{u} \cdot \nabla \mathbf{C}$$

This is ok even if the vector field is velocity itself:

$$D\mathbf{U}/Dt = \partial \mathbf{U}/\partial t + \mathbf{u} \cdot \nabla \mathbf{U}$$

This means the velocity field carries itself moving around. Maybe this is a little wilder, but it is not so hard to understand. Suppose there is a boat which has a velocity with \mathbf{u} , the boat is carried by water. The water also has a velocity $\mathbf{u}_{\text{water}}$. Here the boat can be treated as our quality. This quality is carried by $\mathbf{u}_{\text{water}}$. If the boat’s velocity is the same as the water velocity, that is velocity field advection.

To compute the advection of water, we should update the velocity field at each position. An easy way is using Euler’s method.

$$\mathbf{U}(t+\Delta t) = \mathbf{U}(t) + \mathbf{U}(t) \cdot \Delta t$$

This is an explicit(forward) integration of differential equation. As we told before, a forward differential equation is not stable when using a large time step. So we use the method provided by Stam 1999[16]. We don’t calculate the velocity field by add the change in Δt with the old velocity, but trace back.

$$\mathbf{U}(\mathbf{x}, t+\Delta t) = \mathbf{U}(\mathbf{x}-\mathbf{U}(\mathbf{x},t).\Delta t, t)$$

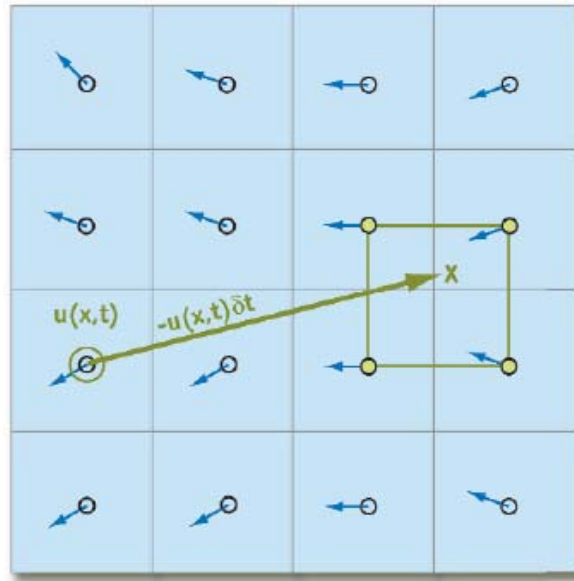


Figure 3.12 Computing Fluid Advection[23] The implicit advection step traces backward through the velocity field to determine how qualities are carried forward.

In position x , we want to determine the velocity U at time t . Along the velocity field we trace back for a distance of $u(x,t)\Delta t$, in Figure 3.12 shows by a green X in a rectangular. We use a bilinear interpolation to calculate the velocity at X .

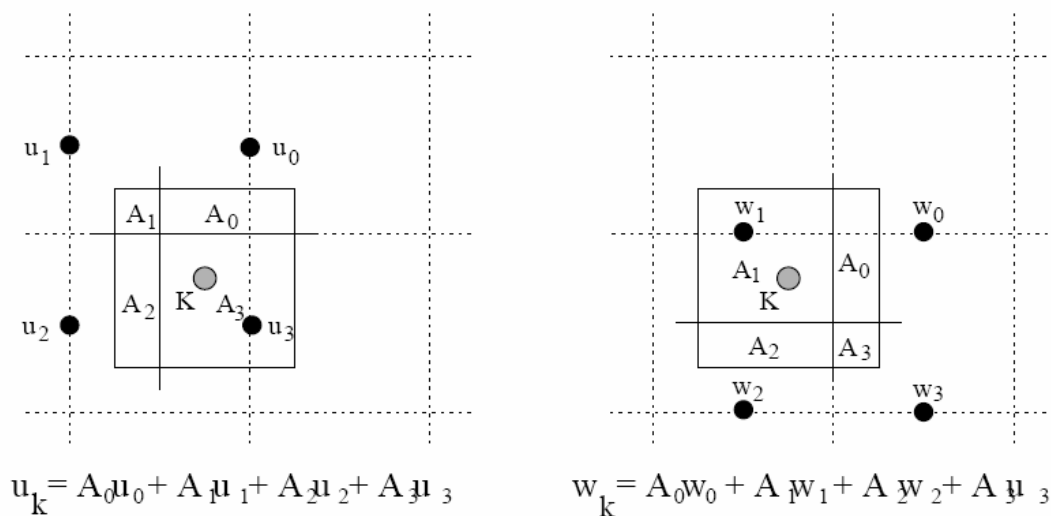


Figure 3.13 bilinear interpolation [24]

We will talk about MAC grid in detail in 3.5, here you just need to know all of our velocities are on the edges. For a point K , if we want to determine it's velocity from around, we can use bilinear interpolation. We will find the nearest four points, then average the velocity by the position of K .

Viscous Diffusion

We have already explained what is viscous. Viscous will slow down our fluid. A

partial differential equation for viscous diffusion is:

$$\partial u / \partial t = \nu \nabla^2 u$$

For a simple way to discrete this partial differential equation we can use forward form. The explicit form is:

$$U(x, t + \Delta t) = u(x, t) + \nu \Delta t \cdot \nabla^2 u(x, t)$$

In this equation, ∇^2 is the discrete form of the Laplacian operator, it an explicit Euler method, which is unstable for large values of Δt and ν . Therefore, we use an implicit formulation of equation as we do in advection.

$$(I - \nu \Delta t \cdot \nabla^2) u(x, t + \Delta t) = u(x, t)$$

Here I is a identity matrix, $|I| = 1$. This implicit formulation is unconditional stable for arbitrary time steps and viscosity. This is much like a Poisson pressure equation, we will use the same code in our implementation to solve both of them.

Solution of Poisson Equations

There are two Poisson equations in our N-S equation: the Poisson-pressure equation and the viscous diffusion equation. Usually Poisson equation is popular in physics. How to solve Poisson is a mathematics problem. Scientists have developed many iteration technologies to solve this problem.

Poisson equation is a one kind of matrix equation with the form of: $Ax = b$. Here x is the vector which we want to solve (p or u), b is constant vector, and A is a matrix.

Name	Cost	Comments
Gaussian elimination	N^3	Use only for very small N (test code)
Jacobi/SOR relaxation	N^2	Easy to code but slow
FFT/cyclical reduction	$N \log N$	Use when no internal boundaries
Conjugate gradient	$N^{1.5}$	Use when internal boundaries
Multi-grid	N	Slower than FFT in practice. Hard to code when internal boundaries present

Figure 3. 14 Linear Solvers From Jos Stam Stable Fluids ppt SIG 2004

Here is table list of methods to solve linear equation. Usually we use Gaussian elimination or Jacobi relaxation to do this. Though it is not fast, it is easy for coding. There are several ways to do this. Multi-grid has a high efficiency, but hard to code and very difficult to understand.

Poisson pressure equation and Diffusion equation is a little different, but we using

the discrete equation and rewritten in the form:

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

Here α and β are constants. The two equations have different x , b , α , β . In the Poisson-pressure equation, x represents p , b represents $\nabla \cdot \mathbf{W}$, $\alpha = -(\Delta x)^2$, and $\beta = 4$. For the viscous diffusion equation, both x and b represent u , $\alpha = -(\Delta x)^2/\nu \Delta t$, and $\beta = 4 + \alpha$.

We will use the same code to solve the two equations, so we formulate the equations in the same way. To solve the equations, we simply run a number of iterations. Usually we run 20 times iteration. The more times your run, the more accurate your calculation will be. We need to find a balance point between accurate and high efficient.

3.5 Water Simulating Strategy

There are many ways have been created till now, we use a classic way created by Ronald Fedkiw[25]. There are five steps for water simulating. First, we will introduce these five steps, and then is the MAC Grid, boundary condition and level set. At this part, we will use all the things that we have introduced.

Water simulating used the same the way to solve velocity field as fire, smoke, and sand. That is because they are all fluid, so we can use N-S equation to describe fluid. Though they all use N-S, they have many differences among them. Water has a simple form of N-S equation, but the simulating strategy is more complex. First, we need to model the static environment as a MAC grid. Then model water volume using a combination particles. The next step is iteration. In the iteration, there are several steps. We use our N-S equations to update the velocity field. Then use advection to move our velocity moving around. Update the position of the water by the new velocity field. This is not very clearly description, we just want you have a basic understanding what will we do in this chapter.

There are many details need to consider. For example the boundary condition, it seems very easy. However when designing it, we found things work not so easy.

3.5.1 MAC Grid

MAC grid(Marker-and-Cell) is one kind of method to discrete space. It is a powerful tool in our water simulating. It's the basic platform, which our water can show on. With MAC grid we can easy translate particles velocity to grid, and process velocity field on the MAC grid. This is not only make our calculation simpler, but also make our data storage less complex. In this chapter, you will why we use a MAC Grid, and how to use a MAC grid.

Splitting Momentum

In N-S momentum conservation equation, we have many items. It will be very painful to handle them at the same time. Therefore, we split up conservation equation into several items. Then integrate them one by one. This will be more easily to design code. This can make our architecture more modularize.

For example: a differential equation:

$$dq/dt = f(q) + g(q)$$

We will solve this differential equation gradually. First, solving $F(q, \Delta t)$, that is solving $dq/dt = f(q)$ for time Δt . Second, solving $G(q, \Delta t)$, that is solving $dq/dt = g(q)$ for time Δt . We add them together, assume $q^* = \text{SolveF}(q^n, \Delta t)$ and $q^{n+1} = \text{SolveG}(q^*, \Delta t)$

Up to $O(\Delta t)$:

$$\begin{aligned}\frac{dq}{dt} &\approx \frac{q^{n+1} - q^n}{\Delta t} \\ &= \frac{q^{n+1} - q^*}{\Delta t} + \frac{q^* - q^n}{\Delta t} \\ &\approx g(q) + f(q)\end{aligned}$$

Let's review our N-S equation ($\partial \mathbf{u} / \partial t = -\mathbf{u} \cdot \nabla \mathbf{u} + \mathbf{g} - 1/\rho \cdot \nabla p$), we have three items on the right side of equation: advection, force and pressure.

First item: **advection** $\partial \mathbf{u} / \partial t = -\mathbf{u} \cdot \nabla \mathbf{u}$

- Move the water through its velocity field

Second item: **gravity** $\partial \mathbf{u} / \partial t = \mathbf{g}$

Third item: **pressure** $\partial \mathbf{u} / \partial t = -1/\rho \cdot \nabla p$

A Simple Grid

We have already finished our splitting in time. Now it's time to splitting space. We start with a fixed Eulerian grid.

Eulerian grid is easy to set up and approximate spatial derivatives. It is particularly good for the effect of pressure. However, we do not use it. Because Eulerian grid is not good for advection. We will use particle to transfer velocity field instead of this to fix this problem.

We can set all the variables at the nodes of a grid. This is easy to simulate, but this will caused some big problems. For example: in 1D our mass conversation equation is $\partial u / \partial x = 0$.

At the grid point is: $\mathbf{u}_{i+1} - \mathbf{u}_{i-1} / 2\Delta x = 0$. The velocity at the grid point isn't involved. This will make our mass conservation equation is useless. The only solution is $\partial u / \partial x = 0$. That means $u = \text{constant}$.

Therefore, we need find a new way to do this. A simple grid cause big disaster.

Staggered Grid

If we skip over grid points, we will get nothing. If we can use the grid point, the problem can be solved. In order to achieve this, we stagger the grid. We use the halfway point between grid points instead of the point in the center of grid. In 1D, we can estimate divergence at grid point as:

$$\frac{\partial u(x_i)}{\partial x} \approx (u_{i+1/2} - u_{i-1/2}) / \Delta x$$

So our problem is solved. Here $u_{i+1/2}$ is the velocity between grid i and $i+1$, $u_{i-1/2}$ is the velocity between grid i and $i-1$.

MAC Grid

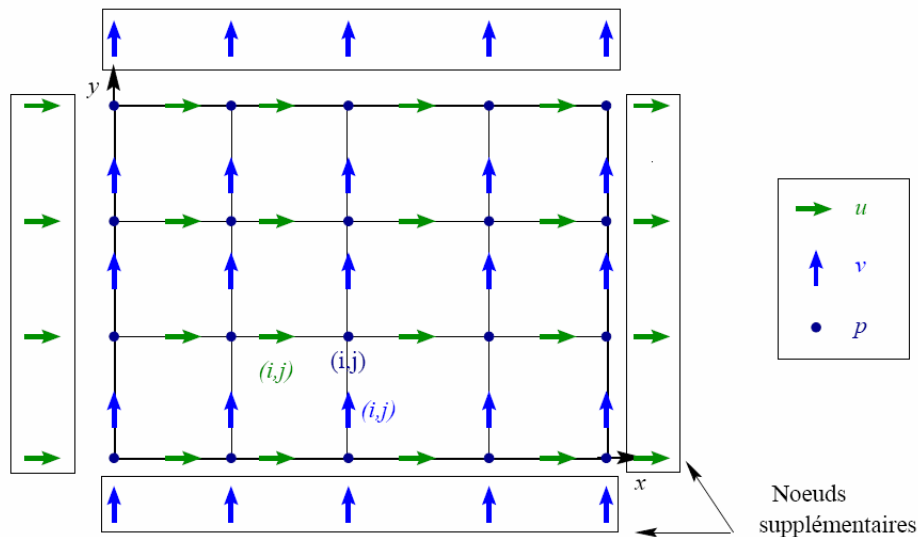


Figure 3. 15 2D MAC Grid From

http://www.lept-ensam.u-bordeaux.fr/aquilon/theses/these_j_breil.pdf page 80

The variables in N-S equation are easily presented in a 2D/3D staggered grid. That is what we want. A 2D MAC grid is shown in the figure Figure 3.15. For grid (i,j) , the pressure $p_{i,j}$ is in the center point. In the graph, a point is a grid center. The green arrow is u velocity and blue arrow is v velocity. For each grid, it has a pressure $p_{i,j}$, a $u_{i,j}$ velocity along X axis and a $v_{i,j}$ velocity along Y axis.

Usually a grid has four edges that mean it has four velocities (two u , two v). However every grid shares its edges with its neighbor. One edge works half for the two grids which share this edge, so one grid actually has only two edges. For each grid, we choose the left edge and the bottom edge for it. That means:

$$u_{i,j} = u_{i-1/2,j}$$

$$v_{i,j} = v_{i,j-1/2}$$

MAC Grid data storage

Since we have MAC grid, we need to have a data structure for storing our variables.

For 2D we have 3 variables, pressure, u velocity and v velocity. For 3D we have 4 variables. Besides p , u and v , we have a w velocity along Z axis.

We don't use a struct (p, u, v) to store our data. We use array instead of struct. When we process our data with N-S equation, we use advection, diffusion and add

force gradually.

We can use the same code to process pressure field and velocity field. Therefore, array is much easier than struct to do this, especially for large $N \times N$ grid.

3.5.2 Particle-in-Cell Methods

There are two ways to simulate water, one is Lagrange and the other is Euler.

Lagrangian method is base on particles, so it is good for solids and bad for fluids. However, Eulerian method is base on grid.

This makes it good for fluids. Therefore, we create a new way, **semi-Lagrangian** method.

We use particles to handle advection. This is Lagrangian method. However, we use grids to handle interactions efficiently. This is Eulerian method. We put these two kinds of methods together.

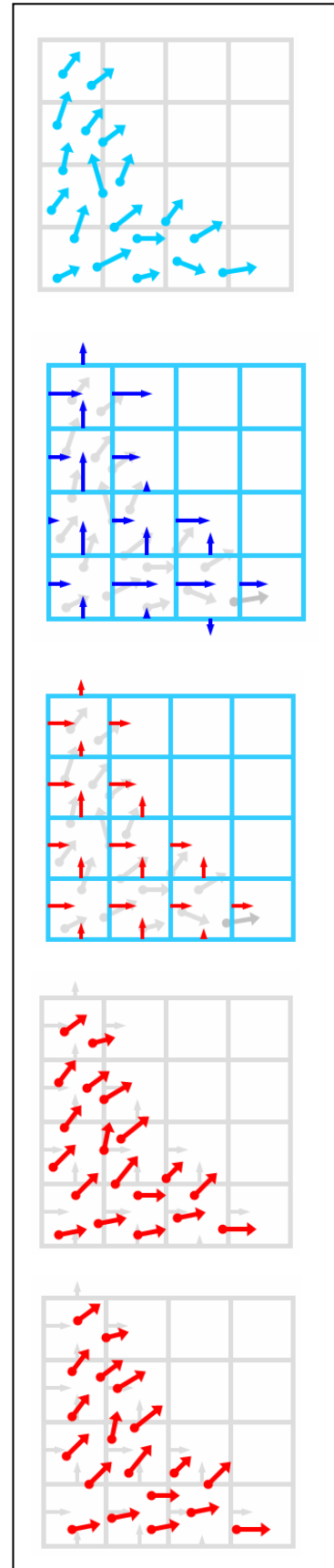
- Transfer velocity to grid
- Solve forces on grid
- Transfer back to particles
- Move particles

Figure 3. 16 Particle-in-Cell Method (on the right)

First, we start with particles. Usually we put eight particles in one grid. In the pictures, we don't have so many particles. That is just illustrating for clear. The particles have initial velocities. In pictures, velocity shows by an arrow with direction and magnitude. Particles will move along the direction of velocity. The length of arrow determines how fast particles move.

Second, translate particles' velocities to grid. Since it is not so convenient to deal with force and interaction on particles, we translate velocity field. On grid we can do this easily. We use the same arrows, but with only directions. The arrows are all on edges. As we mention above, each grid has two velocities at left edge and blow edge.

We add fore on MAC Grid, this is very convenient. Since we have velocity field and pressure field, we use the method in section 3.4.5 to solve the N-S equation. Then we get a new velocity field and pressure field on MAC Grid. Here we need set boundaries conditions.



We will talk about how to set boundaries in next section.

Since we have the new velocity field, we need to transfer it back to particles. Actually, we don't translate velocity back to particles, but use particles to get velocities from grid. We use a bilinear interpolation to get particles' velocity. How to use bilinear interpolation, we have already show it in Figure 3. 12 bilinear interpolation.

The last step is using the velocity field to move particles. At the right side is a full description of particle in cell method.

In 3d, we use the same strategy. This method uses both benefits of Lagrangian and Eulerian method. It is not only easy to deal with force and boundary condition, but also convenient to process advection. This semi-Lagrangian method is though being the first choice to simulate fluid.

3.5.3 Marker particles and Boundaries condition

Marker particles

It is very easy and good to track fluid position in 2D grid with local fluid velocity, when we use mass less marker particles. Since we use mass less marker particles, it will be possible to interact with boundary or splash and flow freely. We mark grid only with three state: empty, full, surface.

- A grid containing no particles is Empty
- A grid containing at least one particle that is adjacent to an Empty grid is surface grid
- A grid containing at least one particle that is not a Surface grid is a Full cell.

With marker particles, we can highlight the full area of water motion such as rotation and

splashing at a greater resolution than the finite difference mesh. One thing we must take care is particles don't have mass. Particles are just used to define the position of water, they can't effect on the calculation.

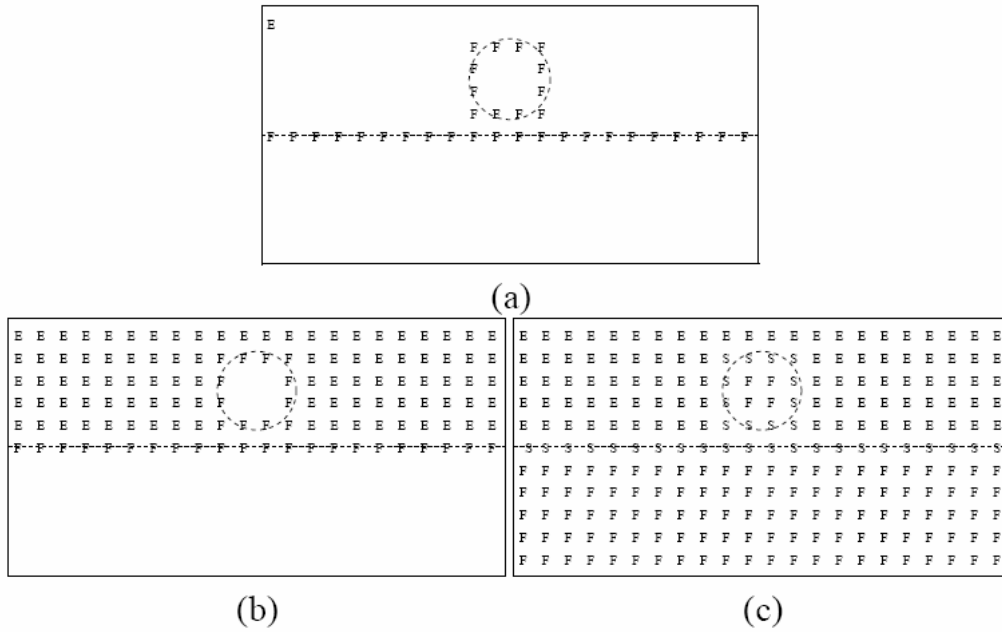


Figure 3. 17 How define the state of each grid, (a) show the surface grids; (b) grow empty grids; (c) growing full grids. (from [24])

Boundary condition

Our water follows no-slip velocity boundary conditions and pure Neumann pressure boundary condition. The no-slip condition indicates that the velocity on the boundaries is zero. The pure Neumann pressure condition requires the normal pressure derivative to be zero at the boundaries.

For the velocity boundary on the left side, we have:

$$(U_{0,j} + u_{1,j})/2 = 0 \quad (\text{here } 0 \leq j \leq N)$$

N is the resolution of grid. So we get $U_{0,j} = -u_{1,j}$. The pressure equation works out in the same way. We use the forward difference approximation of the derivative:

$$(P_{0,j} - P_{1,j})/\Delta x = 0 \quad (\text{here } \Delta x \text{ is the length of grid})$$

That means the pressure of boundary is the same as inside.

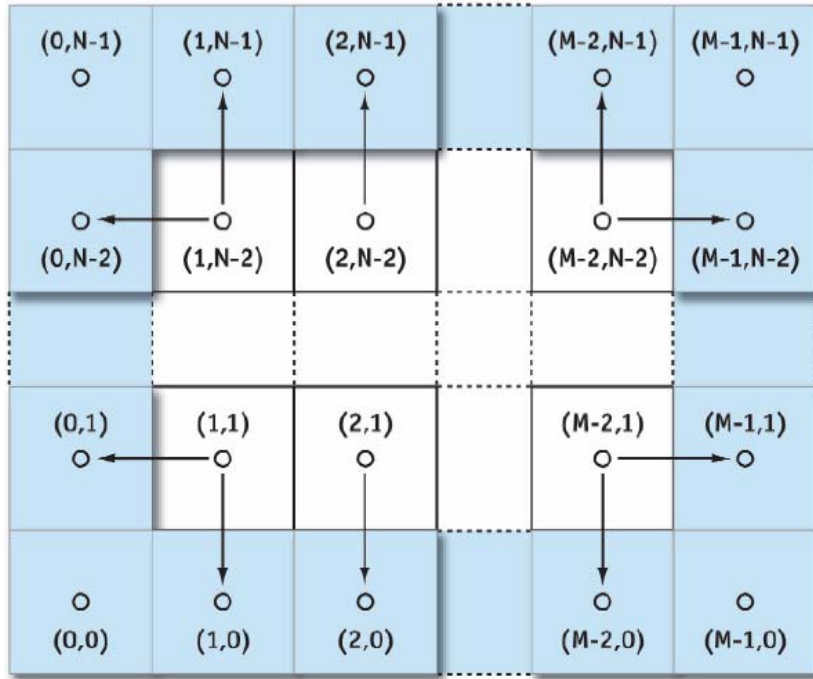


Figure 3. 18 Boundary Conditions on MAC Grid (from [23])

We boundary blow((1,0) (2,0) (M-2,0)) for illustration. For the pressure, we set them the same as line((1,1) (2,1) (M-2,1)). For the velocities of u (velocity along x direction) we don't change it. We still use the u velocity of line((1,1) (2,1) (M-2,1)). However, we use the opposite v velocity. If the velocity of (1,1) is $v_{1,1}$, then we have $-v_{1,1}$ at grid (1,0). For the line((0,1) (0,2) (0,N-2)), we use the same strategy as we use in line((1,0) (2,0) (M-2,0)) to solve pressure. The only difference are we use the same v (velocity along y direction) and opposite u (velocity along x direction). For the four corner grids, we average from nearby grid. For example, $p_{0,0} = (p_{0,1} + p_{1,0})/2$. The velocity is the same.

3.6 Implementation

3.6.1 Platform

Develop PlatForm:

We developed our program with VS.net 2005, you need to download glut.h file from <http://www.opengl.org>.

For informamtion about VS.net 2005, you can find it on page:

<http://msdn.microsoft.com/vstudio/express/visualc/>

You can search how to set include head file in VS.net 2005. For detail information about windows 32 project,

You can search on MSDN for help.

We use opengl 2.0 to accelerate aimation. The high perfomance and strong power provided by opengl 2.0 give our simulating

Operating System:

Operating System which we used is Window Vista, we have already test our program under Windows XP.

For Windows 2000, Windows 2003, Windows 98, we don't suggest you run our program on them.

3.6.2 Opengl Structure

We separate our opengl part from windows 32 programming. We use ordinary window 32 application structure to process input, output devices and windows message transfer. For opengl part, we use “BEGINNING OpenGL GAME Programming” as our reference book. This book give us a vivid structure of Window 32 project with Opengl. Chapter 2 give us a very good direction of programming. This chapter tells us how to create a simple opengl application. The demo application code separate windows main function and opengl code. In this way, you can clearly read the code, a good structure is also powerful for extension and reusing. For more information can be found on page: <http://glbook.gamedev.net/>.

Opengl Structure

```
initial() {  
    InitialData();  
    IntialOpengl();  
}
```

```
render() {  
    renderopengl();  
    renderwater();  
}
```

```
CG::initialdata() {  
    particlesintial();  
    gridinitial();  
    velocityiitial();  
    solidboudaryinitial();  
}
```

```
CG::renderwater() {  
    showgrid();  
    showsolidboundary();  
    showparticles();  
    showvelocityfield();  
    showwaterboundary();  
    showwatervolume();  
}
```

Figure 3.6.2 CG.cpp opengl structure

3.6.3 System structure

This system is a little complex. Maybe you will feel confuse about it, since it is the version of 1.0. We create this system structure for the first time. Therefore, many things need to improve.

We start our system with black circle point, and then initial our data. Our data contain four fields, pressure field, velocity field, velocity divergence field, and curl field. Our MAC grid is $(n+2)*(n+2)$. n is the resolution. We use two arrays to store our particles' coordinate.

When initial data, we initial four fields, and particles' coordinates. Particles' velocities is set to be zero. Then we show our water volume by the particles' coordinates. Here we use grid mesh to determine the shape of water. Since, it is a little complex to use particles with lever set method. This part is done by opengl. After showing, we translate particles' velocities to grid. We use an easy way to do this job. We calculate out the position of each particle, and then add the velocity to correspond grid. When all particles have done, we use a loop to calculate all grids' average velocity. This process is done by dividing the particles' number in the grid. When we get the average velocity, we set particles' number in grid to zero. This translation is the first translation of velocity between particles and grids. We will use another method to translate velocities back. Since we get the velocity on MAC grid, we can do the most important part.

Our water simulating is all about solving N-S equation. In our system structure map(Figure 3.6.3), this process is only labeled with a rectangle. This single process contains four steps: add force, advection, and diffusion (section 3.4.5). We also get rid of boundary condition (section 3.5.3.2) here. We name this part with a name Solving Velocity field (section 3.6.5). Actually solving velocity field contain pressure field processing, divergence field and curl field calculation. All these filed is used to calculate velocity field, so we call them together by name solving velocity field. After a series of steps of calculation, we set MAC grid with new velocity field.

Each grid has a velocity on edges. We need to translate velocity back to particles for further moving. Here we use linear interpolate method (Figure 3. 12 bilinear interpolation) to calculate particles' velocity from MAC grid. Once particles update velocities, we move them with new velocities. Here we don't use new velocity to replace old velocity, but add the new velocity to the old one. After moving particles, we render our water again with particles' position.

Then we start another loop again. This is main structure of our system. First, start and initial data. After set particles' velocities, we start a loop. In this loop we translate velocity to grid, process velocity field and translate it back. Set particles' velocity and move particle. We always use coordinates of particles to show volume of water.

For detail of system structure, we will have another data structure and solving velocity filed to build structure that is more constructive.

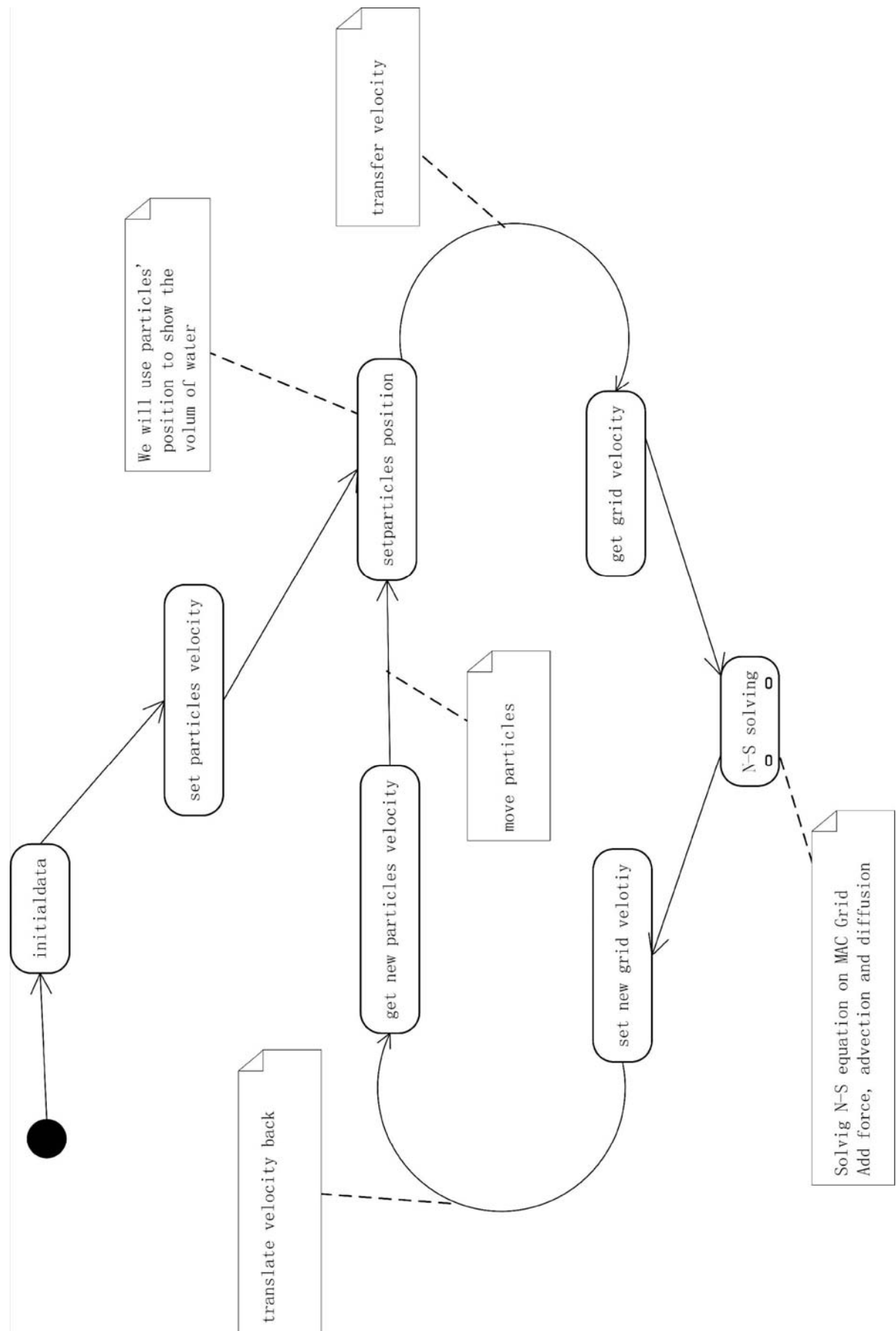


Figure 3.6.3 System structure

3.6.4 Data Structure

We use MAC grid to solve velocity, particles to trace volume of water. In MAC grid, we have velocity field with direction x, y, pressure filed in the center of each grid.

Here, we use a grid of 60*60, so our array size is 62*62. The extra 62*62 – 60*60 grids is used for deal with boundaries.

Array d store buoyancy filed, dOld is a temp container for buoyancy filed. The velocity filed are u, v. When add fore or swap our velocity filed, we use uOld and vOld. Array curl stores rotation of water. If we want to check the type of grid, we can find information in full array.

```
void FluidSolver::reset(void)
{
    d      = new float[size];
    dOld = new float[size];
    u      = new float[size];
    uOld = new float[size];
    v      = new float[size];
    vOld = new float[size];
    curl = new float[size];
    full = new bool[size];

    visc = 0.0; //This is viscosity coeffience
    diff = 0.0; //This is diffusion coeffience

    for (int i = 0; i < size; i++)
    {
        u[i] = uOld[i] = v[i] = vOld[i] = 0.0f;
        d[i] = dOld[i] = curl[i] = 0.0f;
        full[i]=false;
    }
}
```

For particles, we use a strut to present it. In our water simulating, we store our particles in particles2 array.

```
//Particles(2d) (x,y) is coordinate; (u, v) is velocity
typedef struct{
    float x, y;
    float u, v;
}particles2;
```

When we access our grid cell information, use two loops iteration .

```
for (int i = 0; i <= size; i++){
    for (int j = 0; j <= size; j++){
```


This is easy and fast. For array storage, we use a index of $u[I(i,j)]$ (example: u velocity field), $I(i,j)$ is equal to $i + j * \text{size}$.

3.6.4 Data Structure

In our system structure(Figure 3.3.6), Solving velocity is core part. Since we have done the mathematic working in 3.4.5 solving N-S equation gradually, here we just show how to use it in our program. There are four main steps to solving N-S equation(Figure 3.11), add force, diffuse, advection and project. We talk about before, here we just force on how to use these in our programming.

Add force

We use u_{Old} to store gravity, add it to u . Here parameter $x0$ stands for u_{Old} , x for u .

```
void FluidSolver::addSource(float* x, float* x0)
{
    for (int i = 0; i < size; i++)
    {
        x[i] += dt * x0[i];
    }
}
```

Diffuse

As we explain before(3.4.5.5 Solution of Poisson Equations), we use the `linearSolver()` to solve matrix equation of pressure and diffuse equation. Here parameter b is use to define the type of boundary condition. Diff is viscosity coefficient. c is velocity field, $c0$ is not adding force, here we use $c0$ as old u velocity field. After calculation we will get new velocity field in array c .

```
void FluidSolver::diffuse(int b, float* c, float* c0, float diff)
{
    float a = dt * diff * n * n;
    linearSolver(b, c, c0, a, 1 + 4 * a);
}
```

Advection

We use a implicit method to do advection(3.4.5.3 Advection), this method is stable with arbitrary time step. We use bilinear interpolation(3.13 bilinear interpolation) to determine the quality when trace back along the velocity field.

Parameter b is also to set boundary condition, $d0$ is the old quality, d is the new quality. Du and Dv is the velocity field, which we do advection in.

```
void FluidSolver::advect(int b, float* d, float* d0, float* du, float* dv)
```

```

{
    int i0, j0, i1, j1;
    float x, y, s0, t0, s1, t1, dt0;

    dt0 = dt * n;

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            // trace back along velocity field
            x = i - dt0 * du[I(i, j)];
            y = j - dt0 * dv[I(i, j)];
            if (x > n + 0.5) x = n + 0.5;
            if (x < 0.5) x = 0.5;
            i0 = (int) x;
            i1 = i0 + 1;

            if (y > n + 0.5) y = n + 0.5;
            if (y < 0.5) y = 0.5;

            j0 = (int) y;
            j1 = j0 + 1;

            s1 = x - i0;
            s0 = 1 - s1;
            t1 = y - j0;
            t0 = 1 - t1;

            d[I(i, j)] = s0 * (t0 * d0[I(i0, j0)] + t1 * d0[I(i0, j1)])
                        + s1 * (t0 * d0[I(i1, j0)] + t1 * d0[I(i1, j1)]);
        }
    }
    setBoundry(b, d);
}

```

Project

First, we calculate out pressure field by using linearSolver() (3.4.5.5 Solution of Poisson Equations), this the same as we do in diffusion. Then subtract the gradient of pressure(3.4.5.2 The Helmholtz-Hodge Decomposition - First Realization) .

Parameter x, y store new velocity field, p is use for pressure field and dv is for

the divergence field of velocity field. The code contains three parts. First, calculate our divergence field by using old velocity field x, y . Initial velocity field $p=0$. Second, we use linearSolver to get pressure field. At last subtract the gradient of pressure field.

```
void FluidSolver::project(float* x, float* y, float* p, float* div)
{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            div[I(i, j)] = (x[I(i+1, j)] - x[I(i-1, j)]
                           + y[I(i, j+1)] - y[I(i, j-1)])
                           * (-0.5) / n;

            p[I(i, j)] = 0;
        }
    }

    setBoundry(0, div);
    setBoundry(0, p);

    linearSolver(0, p, div, 1, 4);

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            x[I(i, j)] -= 0.5f * n * (p[I(i+1, j)] - p[I(i-1, j)]);
            y[I(i, j)] -= 0.5f * n * (p[I(i, j+1)] - p[I(i, j-1)]);
        }
    }

    setBoundry(1, x);
    setBoundry(2, y);
}
```

LinearSolver

We use 20 times Gaussian elimination (figure 3.14 Linear Solvers) to do solve matrix equation. It is slow but easy to code. We will improve this in the further work.

```
void FluidSolver::linearSolver(int b, float* x, float* x0, float a, float c)
{
    for (int k = 0; k < 20; k++)
```

```

{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            x[I(i, j)] = (a * ( x[I(i-1, j)] + x[I(i+1, j)]
                            + x[I(i, j-1)] + x[I(i, j+1)])
                        + x0[I(i, j)]) / c;
        }
    }
    setBoundary(b, x);
}
}

```

Boundary condition

We use the no-slip boundary condition (3.5.3.2 Boundary condition). Parameter b use for type of boundary condition, because we use different method to set boundary condition. For example, when set boundary of pressure field, we use b=0, when deal with velocity field u, we use b = 1.

```

void FluidSolver::setBoundary(int b, float* x)
{
    for (int i = 1; i <= n; i++)
    {
        x[I( 0, i )] = b == 1 ? -x[I(1, i)] : x[I(1, i)];
        x[I(n+1, i )] = b == 1 ? -x[I(n, i)] : x[I(n, i)];
        x[I( i, 0 )] = b == 2 ? -x[I(i, 1)] : x[I(i, 1)];
        x[I( i, n+1)] = b == 2 ? -x[I(i, n)] : x[I(i, n)];
    }

    x[I( 0, 0)] = 0.5f * (x[I(1, 0 )] + x[I( 0, 1)]);
    x[I( 0, n+1)] = 0.5f * (x[I(1, n+1)] + x[I( 0, n)]);
    x[I(n+1, 0)] = 0.5f * (x[I(n, 0 )] + x[I(n+1, 1)]);
    x[I(n+1, n+1)] = 0.5f * (x[I(n, n+1)] + x[I(n+1, n)]);
}

```

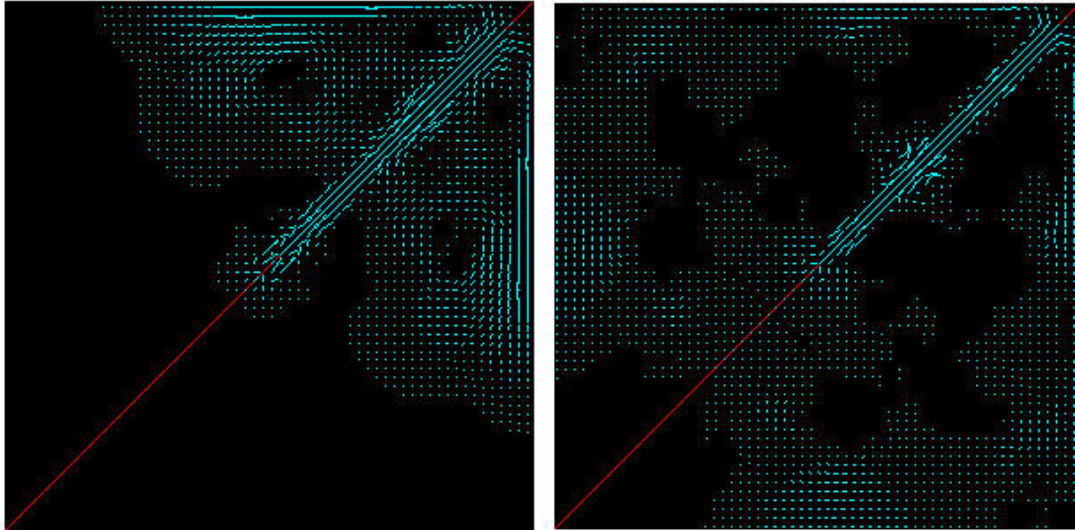


Figure 3. 19 Velocity field

3.6.6 Particles

We solve out the velocity field, then we put particles into the field. Make particles is one kind of Lagrange's method. It's easy to deal with advection. We implement the figure 3.6.3 system structure. For detail, read chapter 3.5.2 particle-in-cell methods.

First, we move particles for 5 times, for each time we use a time step $0.2 \cdot dt$. This is the consideration of stable. Second, we translate particles' velocity to grid. Then solve N-S equation on MAC grid. This part include add force, boundary condition, incompressible, and diffusion. After solving velocity field, we translate velocity back to particles (use weight average).

```
void advance_one_step(Grid &grid, Particles &particles, double dt)
```

```
{
    for(int i=0; i<5; ++i)
        particles.move_particles_in_grid(0.2*dt);

    particles.transfer_to_grid();
    grid.save_velocities();
    grid.add_gravity(dt);
    grid.compute_distance_to_fluid();
    grid.extend_velocity();
    grid.apply_boundary_conditions();
    grid.make_incompressible();
    grid.extend_velocity();
    grid.get_velocity_update();
    particles.update_from_grid();
}
```

This function is very easy, three parameters are grid, particles and time step.

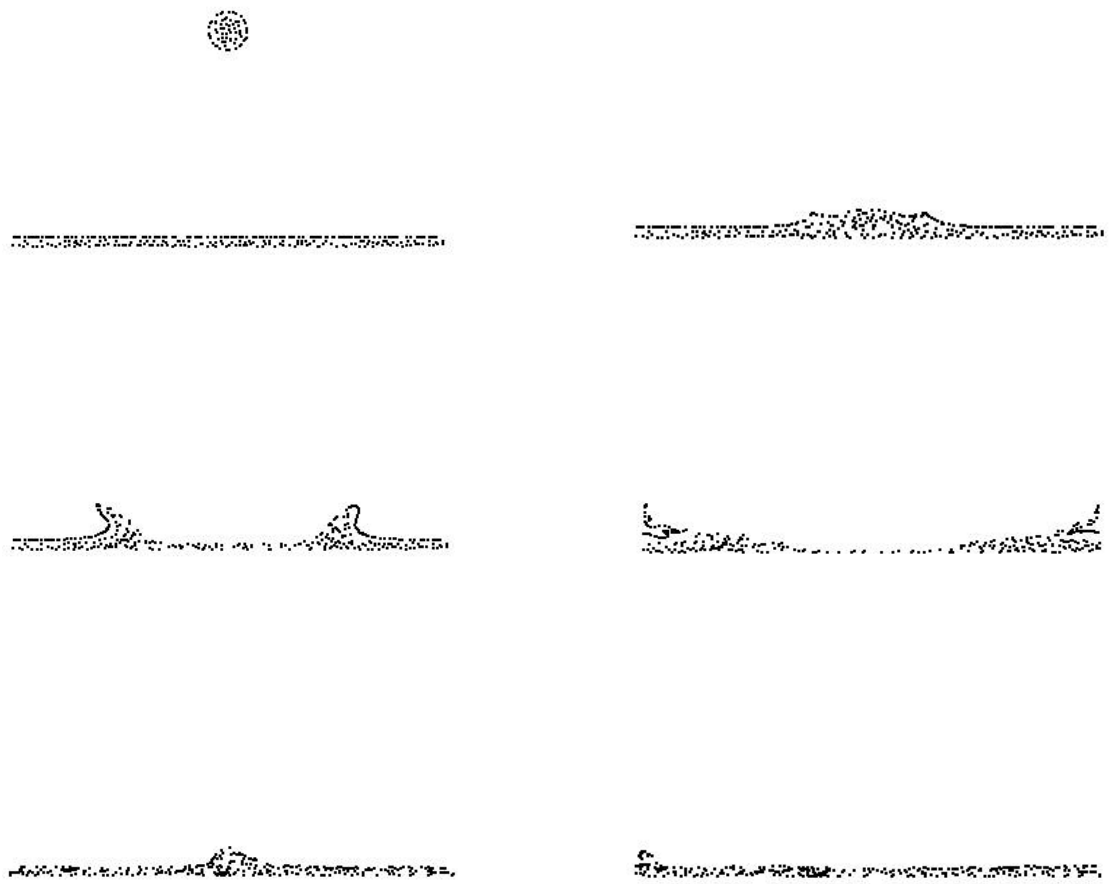


Figure 3. 20 water particles

4 Conclusion

In the beginning, we suppose to implement different methods that we mentioned in the thesis. However, we discover that it is not that easy to do this because we don't have enough time. So we put our energy on the last method---the most difficult one and also most realistic one. We successfully use N-S equations to solve velocity field to simulating water. This is best way to simulating water, it can simulating water so real that you cannot tell the truth since it is physic based. It can perfectly give you a splash and interaction with boundary condition. We use a strategy, which called semi-Lagrange to achieve simulation. Semi-Lagrange use particles to deal with advection, and solve N-S equation on MAC grid. This method make our simulation stable and easily to implement.

As we described before, the method we have implemented is the most realistic, on the other hand, it have some shortcomings. It cost so much time in calculation that it can not simulate water in real-time. Something more, we can not implement water use much particles because of our computer and that's why the water we made is not very realistic.

What we have to investigate more is about how to make this method speed up and cost less time. Together with computers develop day and day, physics based method will be used in movie and game industry.

Compare with masters of water simulating, we just beginners. We have a long way to go on the road of computer graphics.

5 Reference

- [2] Lu Shen, “*ocean water simulating*”, Chinese edition.
- [3] Dwight, Rose, Jay, and Millie, “*Computer Graphics with OpenGL*”, third edition
- [4] Nvidia Corp. website, [online]. Available from:
< <http://developer.nvidia.com/page/home.html> >; [Accessed 23 April 2007]
- [5] Wikipedia, “introduction and history about directx”, Available from:
<<http://en.wikipedia.org/wiki/DirectX>>; [Accessed 1 may 2007]
- [6] Darwyn R. Peachey. “*Modeling Waves and Surf*”. Computer Graphics, 1986, 20(4);
- [7] Jerry Tessendorf, “*Simulating Ocean Waves*”, SIGGRAPH '99 Course Notes & SIGGRAPH'2000: Course Notes 25: Simulating Nature: From Theory to Practice
- [8] Wikipedia, “Fourier transform introduction”, Available from:
<http://en.wikipedia.org/wiki/Fourier_transform>; [Accessed 6 may 2007]
- [9] Sine wave; Available from:
<http://www.sfu.ca/sonic-studio/handbook/Sine_Wave.html> [Accessed 6 may 2007]
- [10] Anders Hast, *Bump mapping*, 2004;
- [11] Bump Mapping; [online]. Available from:
<<http://web.cs.wpi.edu/~matt/courses/cs563/talks/bump/bumpmap.html>>;
[Accessed 8 may 2007]
- [12] Wikipedia, “*Bump mapping*” [online]. Available from:
<http://en.wikipedia.org/wiki/Bump_mapping>; [Accessed 9 may 2007];
- [13] Nvidia developer. “OpenGL cube Map texturing”; Available from:
<http://developer.nvidia.com/object/cube_map_ogl_tutorial.html>;
[Accessed 10 may 2007];
- [14] Gerstner Waves [online]. Available from:
<http://www-viz.tamu.edu/students/jd/658/T_algorithms.html>
[Accessed 13 may 2007]

- [15] Alain Fournier and William T Reeves, "*A Simple Model of Ocean Waves*"; Computer Graphics, Vol. 20, No. 4, 1986.
- [16] Stam, J., "Stable Fluids", ACM SIGGRAPH 99, 121-128(1999)
- [17] Nick Foster and Dimitri Metaxas "Realistic Animation of Liquids" University of Pennsylvania, Philadelphia, PA 19104 June 3, 1996
- [18] Michael Kass, Gavin Miller. Rapid Stable Fluid Dynamics for Computer Graphics. Computer Graphics, Volume 24, Number 4(Proceedings of the 17th annual conference on Computer graphics and interactive techniques)
- [19] James F. O'Brien, Jessica K. Hodgins. Dynamic Simulation of Splashing Fluids. Proceedings of the Computer Animation 1995.
- [20] Jim X. Chen, Niels da Vitoria Lobo, Charles E. Hughes, J. Michael Moshell. Real-Time Fluid Simulation in a Dynamic Virtual Environment. May 1997. IEEE Computer Society Press, Los Alamitos, CA, USA.
- [21] Nick Foster, Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA
- [22] Ron Fedkiw, Numerical Solution of Partial Differential Equations spring 2006 Lecture Notes Lecture 1 webpage: <http://www.stanford.edu/class/cs237c/>
- [23] Mark J. Harris, Nvidia, GPU Gems chapter 38 Fast Fluid Dynamics Simulation on the GPU, University of North Carolina at Chapel Hill
- [24] Nick Foster and Dimitri Metaxas "Realistic Animation of Liquids" Center for Human Modeling and Simulation, University of Pennsylvania, Philadelphia, PA 19104 1996
- [25] Ron Fedkiw, Nick Foster "Practical Animation of Liquids" Stanford University and PDI/DreamWorks



Matematiska och systemtekniska institutionen
SE-351 95 Växjö

Tel. +46 (0)470 70 80 00, fax +46 (0)470 840 04
<http://www.vxu.se/msi/>