Master Arbeit

July 31, 2012

Real-Time Water Simulation

Rahul Mukhi

of Delhi, India (09-743-113)

supervised by

Prof. Renato Pajarola

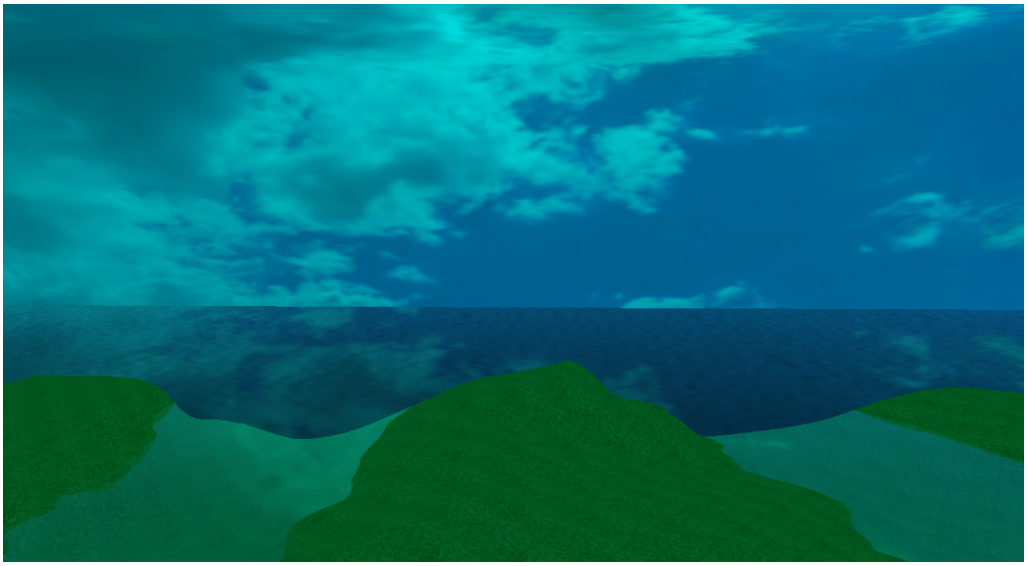
Stefan Geiger

University of Zurich

Department of Informatics

Master Arbeit

Real-Time Water Simulation



University of Zurich
Department of Informatics

GIANTS SOFTWARE

Master Arbeit

Author: Rahul Mukhi, rahul.mukhi@uzh.ch

Project period: 1st Feburary 2012 - 31st July 2012

Department of Informatics, University of Zurich

Acknowledgements

Firstly, I would like to thank **Prof. Dr. Renato Pajarola** and **Mr. Christian Ammann** for allowing me to work on my master thesis at GIANTS Software. A special thanks to **Mr. Stefan Geiger** for his guidance and help throughout the project. The team at GIANTS Software provided a great working atmosphere making it a great place to work for me. This work is the result of the valuable advice, encouragement and support that they have all given me in this project.

Abstract

In this work I present an implementation of water simulation techniques to run in real time with user interaction for video games. The technique combines local shallow water simulation exploiting rigid body interaction via user with computationaly cheaper FFT water simulation method.

The scope of my work is to implement a solution that achieves plausible looking but not realistic physically accurate results. It includes generation of waves created by the interaction of rigid body with water surface, reflecting boundaries for vertical walls and absorbing boundaries between shallow water equations and FFT implemented domain. The rendering of water includes reflection and refraction with alpha blending using fresnel term. The parameters can be tuned to achieve desired effects as per the requirements.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Motivation

Real time water simulation plays a very important part in creating fun virtual worlds in gaming. This visual effect has been used in various games eg. Far Cry, Crysis, World of Warcraft, Battlefield and many more (Figure 1.1). It is also extended to other real time applications like virtual reality and simulator applications.

Figure 1.1: Water scene in Crysis game

The problem of simulating and rendering water in all these applications is that water scenes are very wide and detailed extending upto horizon. In the past, water was simply rendered on a plane surface using textures [1] but as the computational power has increased new methods have been tried to give a more plausible look for water scenes. Due to the computational complexity, realistic fluid simulations are executed offline. For the real time applications, approximated methods are used that can produce plausible results.

1.2 Real-time physics

Contrary to off-line physical simulation where the top priority is realistic visual quality, real-time physics are a part of interactive systems that run at a fixed frame rate, usually between 30 and 60 frames per second (FPS). This frame rate gaurantees a smooth experience to the user with the application. The time required per frame is then from 15 to 30 milliseconds. But the main part of this goes to main features like rendering and artificial intelligence. Physics is left only with few milliseconds and achieving a realisitic visual quality within that time constraint is very difficult.

One solution can be to use off-line methods by decreasing their resolution (eg, the grid), but this result is very blobby and unimpressive. Therefore, new algorithms have been and are being designed in order to solve the problems of real time physically-based simulations in many fields. The trade off is always between the time taken and visual quality produced by the method. It is always desired to have the best visual quality and features within the given time constraint.

1.3 Scope of this work

The purpose of this master thesis is to achieve plausible results for water simulation and rendering in real time by combining two approaches, namely shallow water equations(SWE) and fast fourier transform(FFT) based approaches.

The rest of the work is presented as follows:

Chapter 2 discusses the fluid simulation techniques and related work.

Chapter 3 describes the implementation and results.

Chapter 4 is about the conclusion and directions for future work.

Chapter 2

Water Simulation

Research in the field of fluid simulation has been done for the past three decades and has a long history. The rich and complex behaviour of liquids and gases is the motivation for the effort that has been put in this field by many researchers. Water, like the other fluids, also obeys the same laws of fluid mechanics. The rest of the chapter will describe the theory and methods relevant for real time water simulation.

2.1 Navier-Stokes equation

The motion of fluids is described by the Navier-Stokes equations in terms of pressure and continous velocity field. There are many sources available that describe the full derivation of these equations. A detailed description can be found in SIGGRAPH 2007 Fluid Simulation course notes [2].

The NS equations are defined by two conservation equations that fulfill the incompressible flow of a Newtonian fluid. The first equation also known as the **mass continuity equation** is the following.

$$\nabla . u = 0 \tag{2.1}$$

where u is the velocity field.

The fluid is assumed to have a constant density. This equation means that the overall mass of a quantity will not change when that quantity is advected in the velocity field. The mass of fluid has to be kept constant otherwise it will increase or decrease with the course of time. So it means that the amount of fluid flowing into any volume in space must be equal to the amount flowing out.

The second equation is about the conservation of momentum. They are derived form Newton's second law of motion and accounts for the motion of the fluid through space along with internal and external forces acting on the fluid. It specifies how u, the velocity field changes over time.

$$\frac{\partial u}{\partial t} + u.\nabla u = \frac{-1}{\rho}\nabla p + v\Delta u + g \tag{2.2}$$

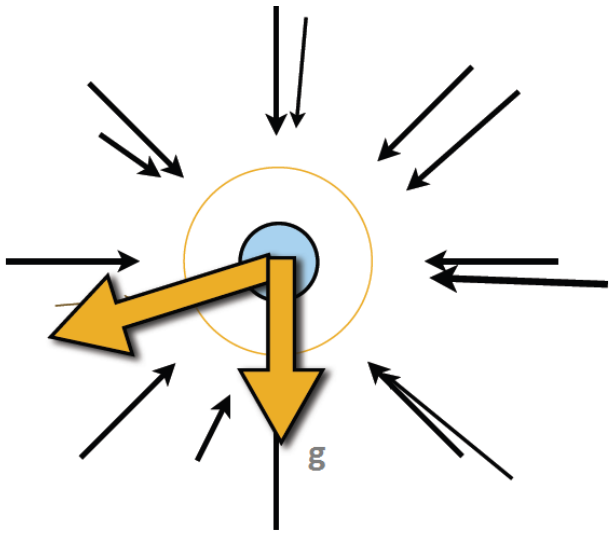


Figure 2.1: Fluid blob

where,
u is velocity field.
p is pressure.
v is viscosity.
g is gravity.
ρ is density.

The term on the left hand side is known as the advection term and describes the local fluid acceleration and convection. It deals with the advection of the velocity of the fluid with itself. The right hand terms take account of acceleration from the following factors. The first term ($\frac{-1}{\rho}\nabla p$) on the right hand side is the gradient of pressure that make the fluid flow away from high pressure regions. The second term ($v\Delta u$) is the viscosity term. Since the viscosity of fluids like air and water is very low, this term is neglected. The third term is the force exerted by gravity on the fluid.

Figure 2.1 shows a fluid blob and the forces acting on it from various directions.

2.2 Lagrangian vs Eulerian approach

The Lagrangian approach also know as the particle model and Eulerian approach also known as the grid model are two spatial discretization methods to solve the Navier-Stokes equations. Figure 2.2 and 2.3 show the difference between the two approaches.

In Lagrangian approach the fluid medium is divided into small blobs or particles that are followed through time. The particles are labelled by a time-dependent vector field and they carry physical attributes. In Eulerian approach there are fixed spatial locations (like a grid) and the fluid is observed as it passes through that location. This approach determines how quantities for that fixed location change with time.

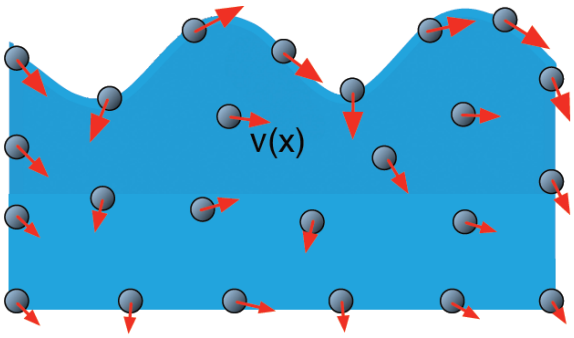


Figure 2.2: Lagrangain approach

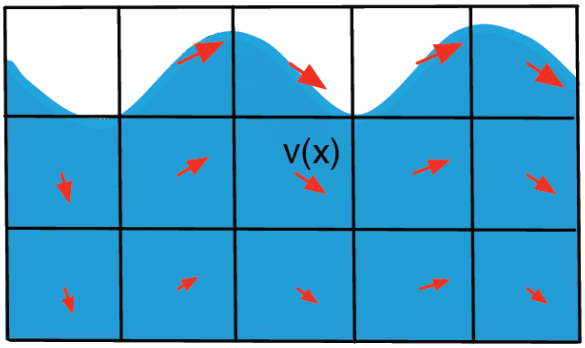


Figure 2.3: Eulerian approach

Both the approaches have different outcomes. In Eulerian approach it is not possible to achieve splash and droplets, which can be done with the Lagrangain approach. A more smooth surface is obtained by using grid model instead of particles. Incompressiblity can be achieved in grid model but not in particle model.

2.3 Related Work

The first work of numerically solving the full three-dimensional Navier-Stokes equations to generate complex water effects in the field of computer graphics was done by Foster et. al [3]. Since then a lot of improved methods have been introduced. Stam [4] introduced a semi-Lagrangian integration scheme that provided a gaurantee stability to the advection step. [5] and [6] introduced a new approach known as level-set based simulations of liquids.

Various adaptive mesh techniques have been also used to increase the temporal and spatial resolution to work in the contraints imposed by CPU time and memory space. The idea is to place more number of grids cells in visually interesting regions. [7] exploits camera's viewing properties to the adaptive mesh techniques for further optimization. Other researchers have proposed adaptive grids such as octrees, tetrahedral grids, voronoi cell grids, tall cell grids etc. for sloving this problem.

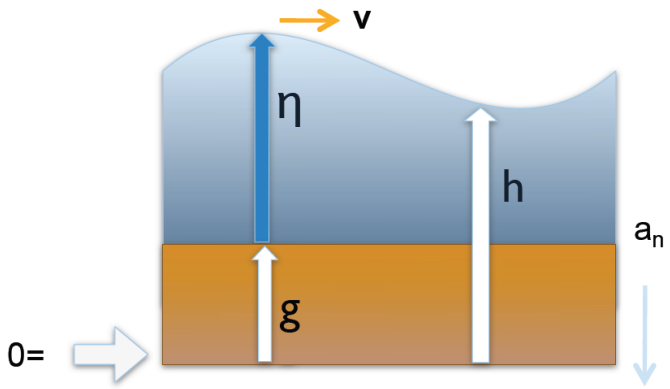


Figure 2.4: Representation of terms in shallow water equations

Kass and Miller [8] first presented their work related to shallow water equations and height-field. Their use in computer graphics field was highlighted by [9]. Hess [10] proposed methods for two way water - rigid body coupling and wet-dry region tracking.

Tessendorf [11] used stastical based model for modeling ocean waves based on the Fast Fourier Transformation(FFT) to create anumation of large bodies of water. The simulation is bound but periodic. This method can generate high resolution and large scale water animation but they can not model the interaction with solid objects easily.

2.4 Shallow water equations

The shallow water equations are a simplified version of Navier-Stokes equations that describe the dynamics of fluids. They reduce the problem from three-dimensional fluid motion to two-dimensional description. The general characteristic of shallow water flows is that the vertical dimension is much smaller than the typical horizontal level. Real Time Physics class notes [12] give detailed derivation of shallow water equations from Navier-Stokes equation.

The basic version of SWE that can be derived from Navier-Stokes equations can be written as

$$\frac{\partial \eta}{\partial t} + (\nabla \eta)\mathrm{v} = -\eta \nabla.\mathrm{v} \tag{2.3}$$

$$\frac{\partial \mathrm{v}}{\partial t} + (\nabla \mathrm{v})\mathrm{v} = a_n \nabla h \tag{2.4}$$

where,
h is the height of fluid above zero-level.
g is the height of ground above zero-level as shown in figure 2.4.
η is the height of fluid above ground, $\eta = h - g$.
v is the velocity of fluid in horizontal plane.
a_n is the vertcial acceleration of fluid (gravity).

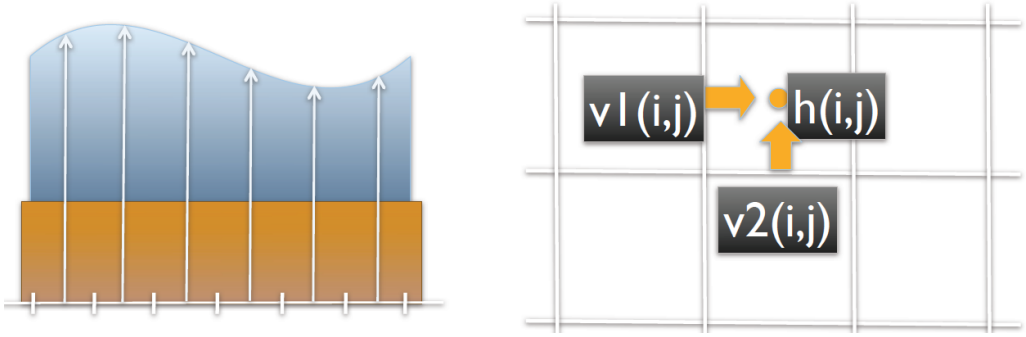


Figure 2.5: Discretization of SWE

t is the time.

A basic solver of the above SWE can be written as

$$\partial\eta/\partial t + (\nabla\eta)\mathrm{v} = -\eta\nabla.\mathrm{v} \tag{2.5}$$

$$\partial\mathrm{v}_1/\partial t + (\nabla\mathrm{v}_1)\mathrm{v} = a_n\nabla h \tag{2.6}$$

$$\partial\mathrm{v}_2/\partial t + (\nabla\mathrm{v}_2)\mathrm{v} = a_n\nabla h \tag{2.7}$$

In these equations, the left side accounts for the advection for advection with the velocity field v and the right side computes an additional acceleration term. To solve these equations many integration schemes like explicit or implicit Euler can be used. Semi-lagrangian scheme [4] is a good alternative as it provides an unconditional stability to the solution.

To solve the equations, first the space is discretized into a staggered grid as shown in figure 2.5 with specific number of grid cells in x and z directions. Each gridcell has a square form with Δx edge length. The gravity force, a_n is acting downwards in negative y direction. This coordinate system is analogous to the left hand coordinate system in OpenGL implementation. Also, a fixed single time step Δt is considered to solve the equations. The speed of waves created on the height field are dependent on the height of water, gravity and fixed time step.

The staggered grid contains pressure at the centre of the grid cell and the velocity components are located at the centre of each edge. This grid approach prevents instability and is used for many fluid solvers. The derivatives are computed with finite differences. The basic algorithm for an update of SWE equations is the following.

SWE-update-step (η, v, g)
1. η = Advect(η, v)
2. v_1 = Advect(v_1, v)
3. v_2 = Advect(v_2, v)
4. Update-height (η, v)

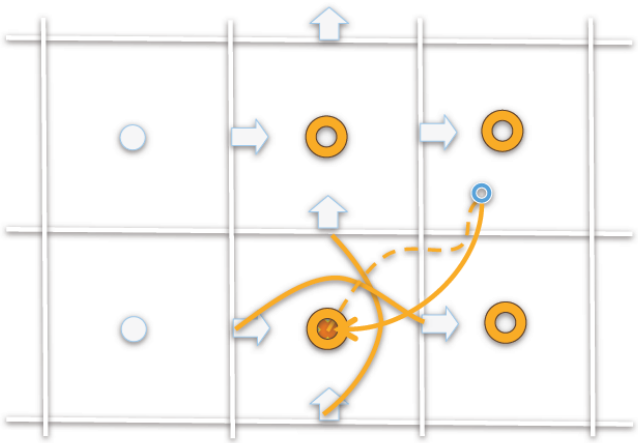


Figure 2.6: Backtracing in staggered grid

5. $h = \eta' + g$
6. Update-velocities(h, v_1, v_2)

In order to advect the height and velocities a temporary array has to be stored for the grid which then copies the values to the respective fields. This is because the Advect function returns a value that is assigned back to the original input grid.

The semi-Lagrangain method performs a backwards trace of an imaginary particle in order to compute advection at each grid cell. This is done by calculating the position x^{t-1} = **x** - Δt.v(**x**) by tracing a particle at this position backward in time. The value to be advected, s, is updated with the value at x^{t-1}, so the new value is s(**x'**) = $s(x^{t-1})$.

Advect(s,v)
1. **for** j = 1 **to** n_2-1
2. **for** i = 1 **to** n_1-1
3. **x** = ($i.\Delta$x, $j.\Delta$x)
4. **x'** =**x** - Δt.v(**x**)
5. $s'(i, j)$ = interpolate(s, **x**)
6. **endfor**
7. **endfor**
8. **return**(s')

Since **x'** can be located anywhere in the grid, it usually requires an interpolation to compute the value of s at that position. Bi-linear inerpolation ensures stability as the interplolated values are bounded by its source values from the grid. Note that the backtracing of the particle is also clamped to the spatial region within the grid. The quauntity v(**x**) requires the averaging of two neighbouring velocity components. For each advection step it requires 3 different velocity interpolations to be done, two for horizontal components of velocities and one for height.

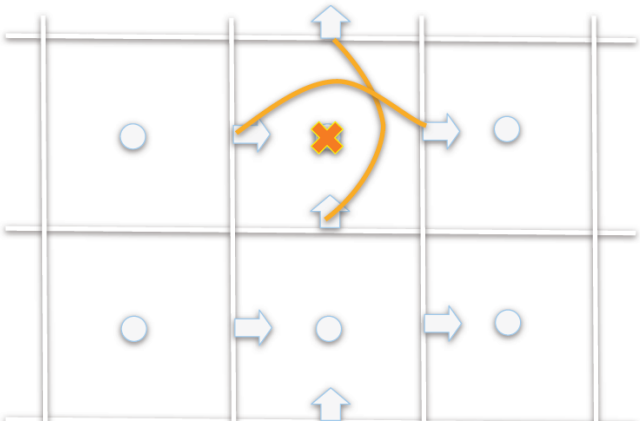


Figure 2.7: Height update by divergence of velocity fields

Update-height(η,v)
1. **for** j = 1 **to** n_2-1
2. **for** i = 1 **to** n_1-1
3. $\eta(i,j)$ -= $\eta(i,j).(\frac{v_1(i+1,j)-v_1(i,j)}{\Delta x} + \frac{v_2(i,j+1)-v_2(i,j)}{\Delta x})\Delta t$
4. **x′** =**x** - Δt.v(**x**)
5. **endfor**
6. **endfor**
7. **return**(η')

For the acceleration term for velocity update, the gradient of the overall fluid height is taken. But in this case, the total height above the zero-level is used instead of the fluid height above the ground level. This is because when the fluid is on an inclined plane, there has to be an induction even when the fluid height (η) is constant. Otherwise on an inclined plane the divergence of η will be zero. The parameter a is the gravity acting on the fluid.

Update-velocities(h, v_1, v_2, a)
1. **for** j = 1 **to** n_2-1
2. **for** i = 2 **to** n_1-1
3. $v_1(i,j)$ += $a(\frac{h(i-1,j)-h(i,j)}{\Delta x})\Delta t$
4. **endfor**
5. **endfor**
6. **for** j = 2 **to** n_2-1
7. **for** i = 1 **to** n_1-1
8. $v_2(i,j)$ += $a(\frac{h(i,j-1)-h(i,j)}{\Delta x})\Delta t$
9. **endfor**
10. **endfor**

Note that the velocities and height at the boundaries are not updated as their derivatives can't be calculated. Special boundary conditions are required for the border cells to achieve different effects. It is not necessary that the boundary cells only lie at the border of the grid. If their are steep walls in the middle of the grid then the cells have to be labelled to know if they are a part of simulation domain or a boundary.

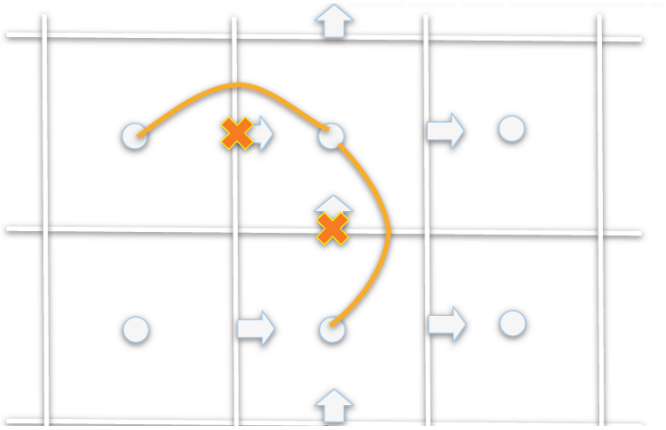


Figure 2.8: Velocity update by gradient of fluid height

2.5 Boundary conditions

Boundary conditions are an important part of water simulation as it gives more realistic effect when water interacts with walls or landscape. The main types of boundaries related to fluid simulation are namely reflecting, absorbing, free surface and periodic. Reflecting boundaries are used to model a wall that reflects incoming waves. Absorbing boundaries give an effect of water surface in which the waves just leave the spatial domain and dont come back. Free surface boundaries are used in the applications like a flowing river where the water is flowing through a landscape. Periodic boundaries are not used much as they have an effect of leaving through one side and entering the domain from the other side.

2.5.1 Reflecting Boundaries

Reflecting boundaries set the value of velocity component orthogonal to the boundary line to zero to avoid any flux through the wall. The velocity component along the boundary remains the same and the waves are allowed to move tangentially. The fluid height at the boundary is mirrored with the inner layer. This can be put as following for boundary perpendicular to x-axis on the left border.

$h'(0,j)' = h(1,j)$
$v_1(1,j) = 0$

The reason $v_1(1,j)$ is set to 0 because as $v_1(0,j)$ is not accessed during the computation step.

2.5.2 Other Boundary Conditions

For absorbing boundaries perfectly matched layer [13] and Higdon boundary conditions [14] can be used. Implementing absorbing boundaries is difficult than reflecting boundaries and they are usually used when the computational domain of SWE is coupled with some other domain at the boundaries of SWE spatial domain. A cheaper way is also be damp the velocities and height at the borders or apply a heuristic that weighs the velocities and height from SWE and the other domain.

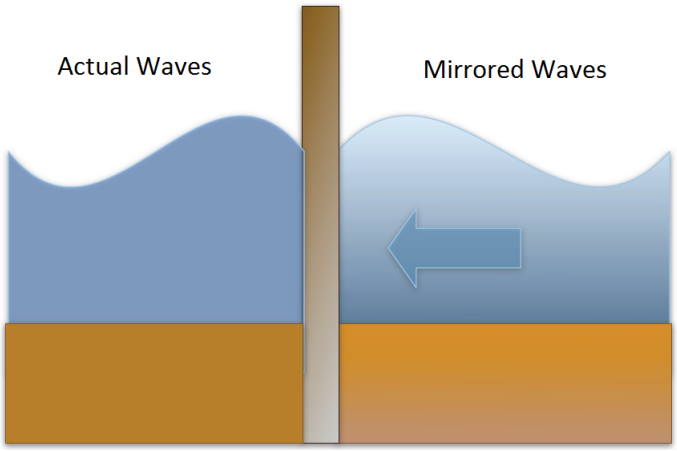


Figure 2.9: Reflecting boundaries

For the free surface boundaries where the complete domain is not fluid, like a river, and consists of arbitrary terrain, the cells have to distinguished as fluid or dry. The computation has to be then done only for the fluid filled cells and then the boundaries have to be recalculated as dry or fluid cells. It is to be noted that the height of fluid in all the cells must not go below zero. Dry cells have a fluid height of zero and not negative.

2.6 Rigid Body Interaction

Rigid Body Interaction is a two way coupling between the object and the fluid where the fluid affects the object and vice-versa. It plays a very important part in games and simulators as the interaction between the object and fluid gives the user the ability to interact with the fluid domain. SWE have the ability to form a rigid body coupling with the object where the heightfield is affected by the motion and weight of the object and the fluid exerts buoyant force on the object causing it to float.

Figure 2.10 shows the effect that object has on the fluid heightfield. In this general case the object affects 2 cells on the fluid domain. As the object moves down(due to gravity) by a distance d, the water is distributed to the nearby cells. Since the height of the nearby cells is more, a wave flows outwards in the fluid domain. The following are the forces acting on object due to its interaction with the fluid.

First is the buoyant force that acts on the body due to the volume of water displaced by the object.

$$F_{buoyancy} = -g\rho V \tag{2.8}$$

where,
$F_{buoyancy}$ is the bouyant force acting on the rigid body.
g is the force of gravity.
ρ is the density of fluid.
V is the volume of of rigid body below water surface.

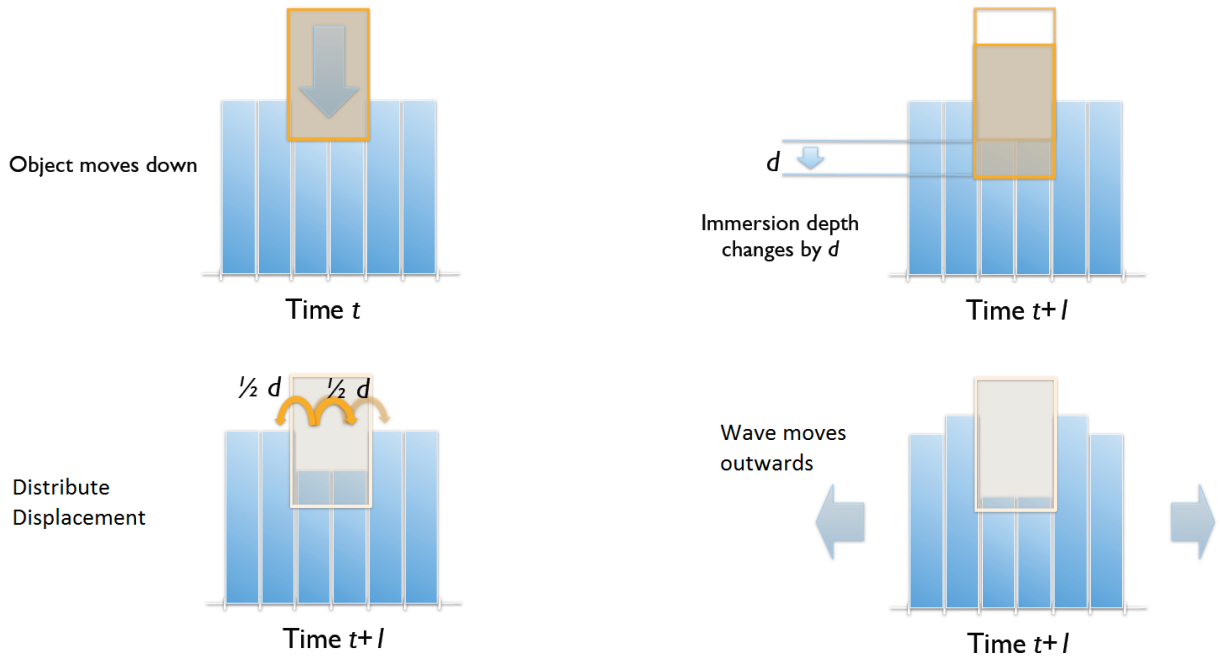


Figure 2.10: Rigid Body Interaction

Second is the drag force that acts due to the resistance of motion of the object in the fluid environment.

$$F_{drag} = -\frac{1}{2} C_D A^{eff} |\mathbf{v_{rel}}| \mathbf{v_{rel}} \tag{2.9}$$

where,
F_{drag} is the drag force acting on the rigid body.
C_D is the drag coefficient.
A^{eff} is the effective area of solid in fluid.
$\mathbf{v_{rel}}$ is th relative velocity of object with respect to the fluid.

Third is the lift force that is perpendicular to the flow direction.

$$F_{lift} = -\frac{1}{2} C_L A^{eff} |\mathbf{v_{rel}}| (|\mathbf{v_{rel}} \times \frac{\mathbf{n} \times \mathbf{v_{rel}}}{|\mathbf{n} \times \mathbf{v_{rel}}|}) \tag{2.10}$$

where,
F_{lift} is the lift force acting on the rigid body.
C_L is the lift coefficient.
A^{eff} is the effective area of solid in fluid.
$\mathbf{v_{rel}}$ is th relative velocity of object with respect to the fluid.

n is the normal.

It is to be noted that for complex objects these forces are calculated by dividing the object into simple triangles and integrating forces over them.

2.7 Fast-fourier Transform based wave simulation

The fast fourier transform based wave simulation has been described by [11] and [15]. By decomposing the wve height-field as the sum of sine and cosine waves it is easire to build a statistical model of ocean waves. [11] has used the FFT decomposition for this as it is a faster way to do that. This decomposition is done as follows

$$h(\mathbf{x}, t) = \sum_j \tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k}.\mathbf{x}) \tag{2.11}$$

where $h(\mathbf{x}, t)$ is the wave height at horizontal position **x** = (x, z) and t is the time. **k** is a two-dimensional vector (k_x, k_z) where $k_x = 2\pi n/L_x$ and $k_z = 2\pi m/L_z$ with n and m with bounds $-N/2 \leq n < N/2$ and $-M/2 \leq m < M/2$. The height field is generated at these discrete points given by $\mathbf{x} = (nL_x/N, mL_z/M)$

Statistical analysis demonstrate that the wave height amplitudes $\tilde{h}(\mathbf{k}, t)$ are nearly statistically stationary, independent, gaussian fluctuations with the following spatial spectrum where $\left\langle \right\rangle$ denote the data-estimated ensemble averages.

$$P_h(\mathbf{k}) = \left\langle |\tilde{h}^*(\mathbf{k}, t)|^2 \right\rangle \tag{2.12}$$

Phillips spectrum model for wind-driven waves in a fully developed sea is given by

$$P_h(\mathbf{k}) = A\frac{\exp(-1/(kL)^2)}{k^4}|\hat{\mathbf{k}}.\hat{w}|^2 \tag{2.13}$$

where $L = V^2/g$ is the largest waves arising from a continous wind of speed V and g is the gravitational constant. $\hat{w}$ is th direction of wind and A is a numeric constant.

Using the phillips spectrum and gaussian random numbers the fourier amplitudes of wave height filed can be generated by

$$\tilde{h_0}(\mathbf{k}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{P_h(\mathbf{k})} \tag{2.14}$$

where ξ_r and ξ_i are ordinary independent draws from a gaussian random number generator, with mean 0 and standard deviation 1.

Finally, the Fourier amplitudes of wave field realization at time t with a given dispersion relation $w(k)$ are given by

$$\tilde{h}(\mathbf{k}, t) = \tilde{h_0}(\mathbf{k}) \exp\{iw(k)t\} + \tilde{h_0}^*(-\mathbf{k}) \exp\{-iw(k)t\} \tag{2.15}$$

This creates a random ocean wave heightfield.

Chapter 3

Implementation and Results

3.1 Working environment

All the implementations were done on HP Pavillion dv6-6170us notebook with following specifications

- **Processor** - Intel Core i7-2630QM.
- **RAM** - 6.00 GB.
- **Display** - AMD Radeon HD 6770M.
- **Operating System** - Windows 7 64-bit.

In addition to the following language, tools and libraries, other base and utility classes from GIANTS software have also been used.

- **Languages** - C++, GLSL.
- **Development Environment** - Microsoft Visual Studio 2010 Ultimate.
- **Libraries** - glut, glew, fftw, libpng.
- **Operating System** - Windows 7, 64-bit.

Shallow water equations have the advantage of simulating water in a more realistic way and being able to interact with objects which is not possible with various other methods that create water waves and other desirable effects. But it is computationaly very expensive to simulate the whole water scene with shallow water equations with a good resolution. Therefore, in games where there is always a time constraint for physical simulation, it is not possible to simulate the whole scene with shallow water equations and achieve a smooth interactive frame rate. To overcome this constraint, the shallow water equations have been combined with computationaly cheaper fast fourier transform (FFT) based approach to achieve the desired result at an interactive frame rate.

The whole water scene is divided into 2 parts. One is the shallow water equations (SWE) grid and the rest is the fast fourier transform (FFT) plane. The SWE grid always remains in the front of the camera where the details of object interaction like wave generation etc. are desired. The crucial thing is the stitching of these two parts together so that the user does not feel the boundary or discontinuity between them. The following describes the implementation of both the techniques and other desired effects for water simulation in real time.

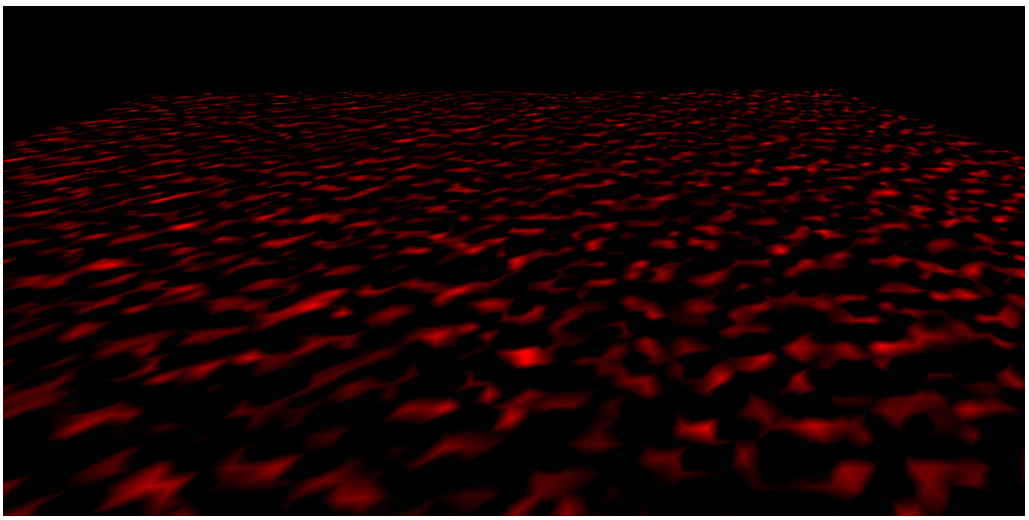


Figure 3.1: Normal infomation in SWE grid

3.2 Shallow water equations (SWE)

Shallow water equations are implemented by using the following data structure and the grid is rendered using VBOs with stream array buffer and static element array buffer (Figure 3.2). The grid resolution is 120x120 with cell edgelength as 1.5. Figure 3.1 shows the normal information of the waves formed by the SWE.

3.2.1 Gridcell data structure

The data structure below defines the parameters for each gridcell in shallow water equations grid.

```
struct GridCell
   {
      float y; // total height of gridCell, (groundHeight + waterHeight)
      State state;
      float xVelocity, zVelocity, temp, waterHeight;
   };
```

Listing 3.1: Gridcell data structure

Each gridcell has the above parameters. The following things should be noticed.

- Groundheight is the height of ground above zero level.
- State consists of water, ground, boundary, near-boundary, object and object boundary states.
- Temporary values of velocities and height while advecting are stored in temp.

The x,z position of each gridcell is calculated by using translate{x,z} (translation variables in x and z directions) and constant gridstart{x,z} (x and z coordinates of gridstart) variables. Translate{x,z} is dependent on the movement of the camera and is passed to the vertex shader to alter the position of the each gridcell for rendering. Ground height for cells is also calculated using their {x,z} position.

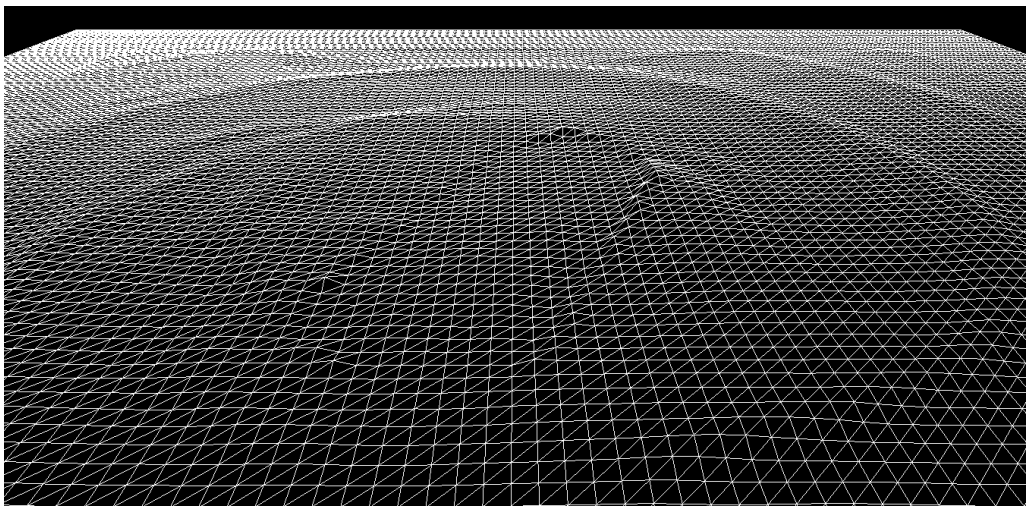


Figure 3.2: Wave simulation in SWE grid with unrendered boat in the middle

3.2.2 Moving SWE grid with Camera

The camera position can be changed as per the requirements but in the implementation the centre of the boat is taken as the origin of rotation of camera. The view vector of the camera always has its x and z coordinates as the same as the centre of the boat. So, wherever the boat is translated, view vector changes. The position of camera maintains a constant distance with the view vector thus following the boat.

For SWE grid to be always in front of the camera, view vector of the camera is used. Figure 3.3 and Figure 3.4 show the difference in the position of the SWE grid with the movement of the camera. Grid cells in scene 2 take the parameters of the grid cells that were at that position in scene 1. If there were no grid cells present at that position in scene (for example at the boundary region), then new cells are created and assigned states according to the groundheight at their and their neighbour's positions. For the new cells that are created, they are given a velocity of zero in both directions and a random height so that it matches the FFT domain at the boundaries.

3.3 Fast fourier transform (FFT)

FFT approach is implemented differently from shallow water equations. In this approach a grid-size of 64x64 is taken and a normal map is created using the same which is used in the fragment shader to distort the reflection map and calculate fresnel term for alpha blending. Figure 3.5 shows the FFT plane and the wave simulation using the normal map. The FFT plane is also translated with the SWE grid in the same way in the vertex shader. The height of FFT plane remains constant throughout.

The fftw library has been used to decompose the wave heightfield and the real part of the decomposed heightfield at that location is used as the height at that location. Since FFT plane is rendered using a normal map, the normals are calculated using height information at that point and its neighbours. The normals are then used to create a normal map which is used in the fragment shader to perturb the reflection map giving an impression of moving water.

Figure 3.3: SWE grid position in scene 1

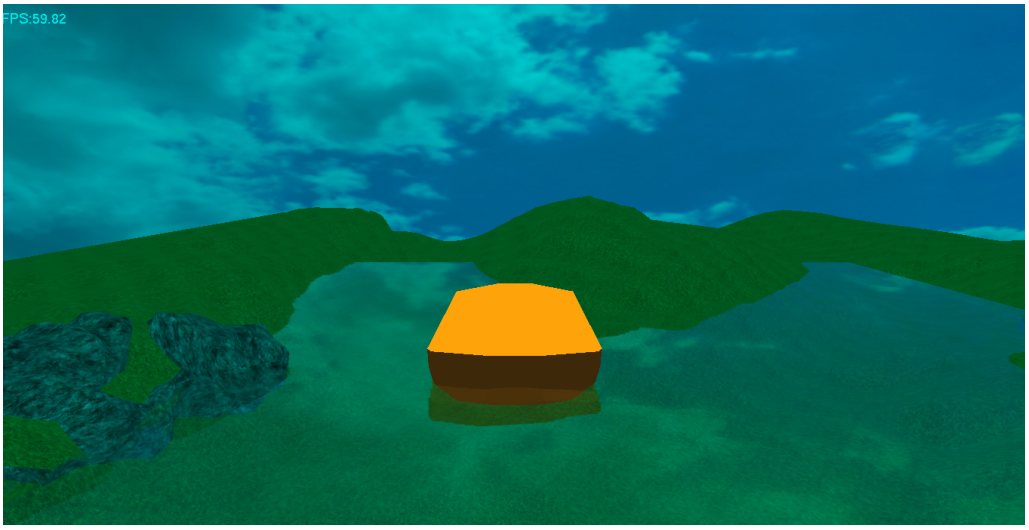


Figure 3.4: SWE grid position in scene 2

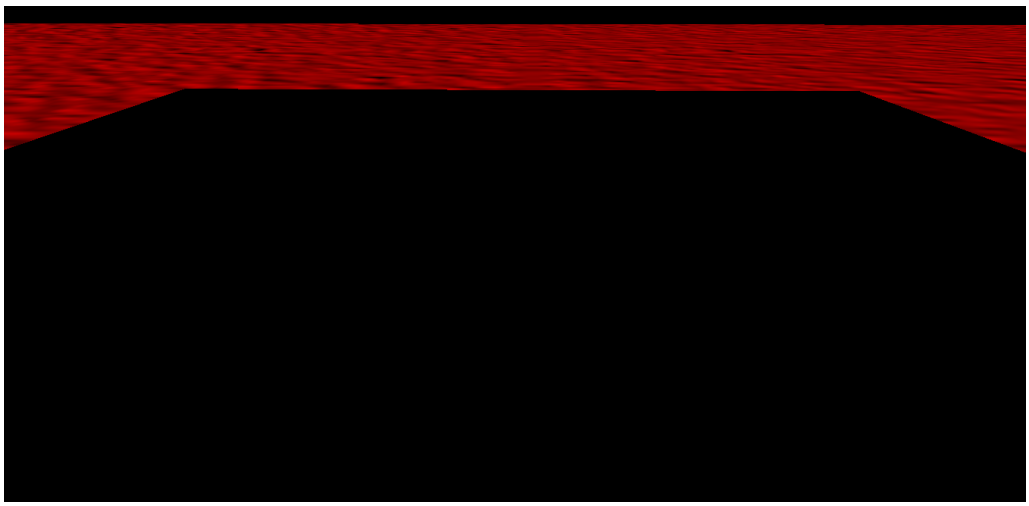


Figure 3.5: Wave simulation for FFT

3.4 Stitching SWE grid and FFT plane

Figure 3.6 shows the wave simulation by SWE and FFT. Since the normals are generated from different approaches, they have to be blend together to give a smooth appearance between the two. Figure 3.7 and Figure 3.8 show the non-stitched and stitched parts. In Figure 3.7, at the centre of the scene one can see the discontinuity between the two. This is without the blending of the normals. Morever, the animation also looks different as once can figure out that the waves generated are from different sources as they have different appearnace.

In Figure 3.8, the discontinuity is not there. This is the result of blending FFT normals from the FFT normal map with the normal at the gridcell. The weighing for the blending of two normals is done according to the distance of the gridcell fron the centre of the SWE grid. The more the distance from the centre of SWE grid, the more is the weight of FFT normal. So at the boundary cells of SWE grid the weight of FFT normal is 1 which decreases towards SWE grid centre.

```
float dx, dz; // variable weights

if(Xvertex<0.0f) // Xvertex is x coordinate of the vertex
   dx = (0.0f - Xvertex)*inv_gridLength;
else
   dx = (Xvertex - 0.0f)*inv_gridLength;

if(Zvertex<0.0) // Zvertex is z coordinate if the vertex
   dz = (0.0f - Zvertex)*inv_gridLength;
else
   dz = (Zvertex - 0.0f)*inv_gridLength;
```

Listing 3.2: Stitching SWE and FFT

As shown in listing 3.2, dx and dz are the two variables that measure the distance of x and z coordinates of the SWE grid vertex from the centre. The centre of the grid lies at (0,0). The values

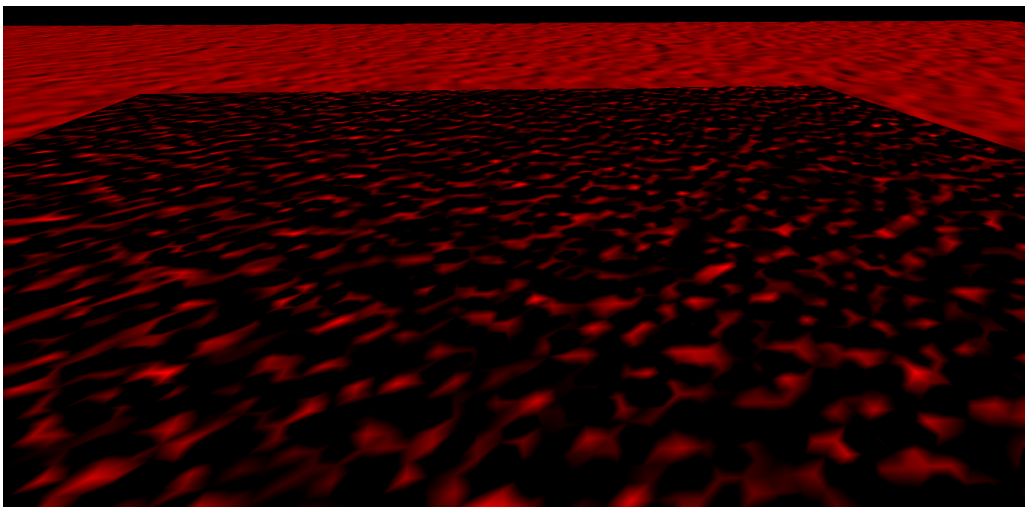


Figure 3.6: FFT and SWE

of dx and dz are bound to [0, 1/2]. Whichever of the two is greater is taken as the weighing factor for blending the normals from SWE and FFT domain at that point.

Also, there is another important point to be noted while stitching the boundaries. The coordintaes at the boundary of SWE and FFT domain should match otherwise there are artifacts such as black pixels and the border between them can be seen. To do this the height of the boundary cells at SWE grid are kept at the same height level of the FFT domain.

3.5 Rigid Body Interaction

3.5.1 Volume submerged calculation

For the object interaction, firstly the object is triangulated if not already. To calculate the buoyant force on the rigid body it is important to know the volume of the object submerged in the water. Volume submerged is calculated by adding the volume contributed by all the triangles inside water. But since a triangle only has area, the volume is obtained by considering a tetrahedron having the 3 vertices of the triangle and the centre of the boat. The tetrahedron volume is given by

$$V = \frac{1}{6}(b \times a).r \tag{3.1}$$

where,
V = Volume of tetrahedron.
b, a and r are the three edges that meet at one vertex.

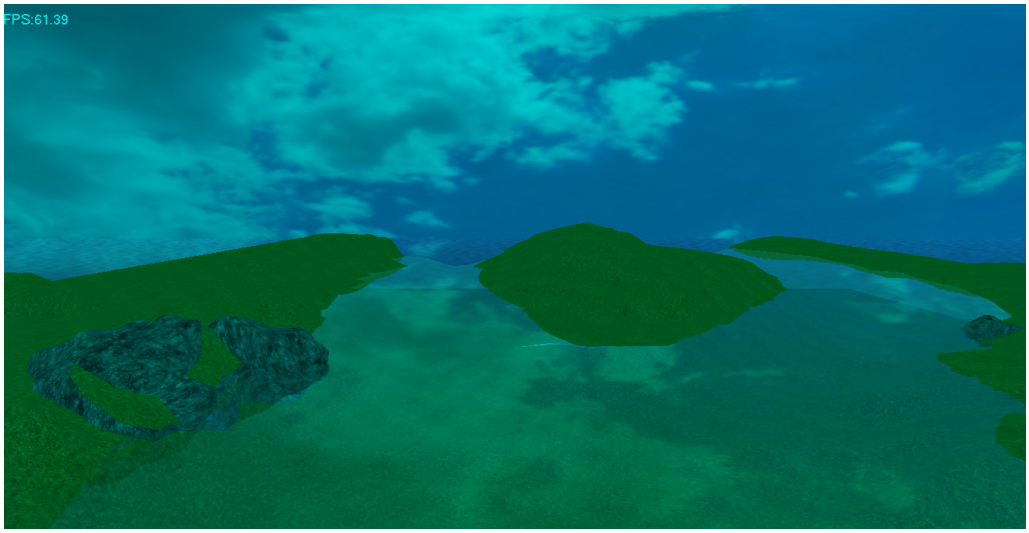


Figure 3.7: Without Stitching

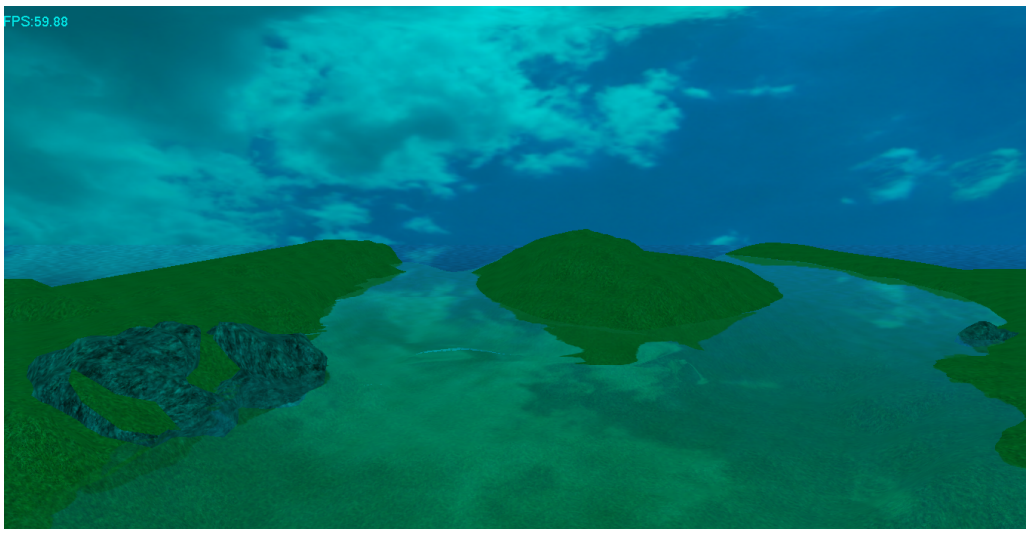


Figure 3.8: With Stitching

There are four cases to be considered.

• If all the three vertices of the triangle in the object are above water plane, then the triangle does not contribute to the volume submerged.
• If all the three vertices are below water plane, then the full triangle contributes to the volume submerged.
• If only one vertex of the triangle is below water plane then the triangle is clipped by the water-plane and the volume of that triangle contributes to volume submerged.
• If two of the three vertices lie below waterplane, then the waterplane clips the triangle leaving a quadrilateral. This quadilateral is then divided into two triangles and the volume by these two triangles contributes to the volume submerged.

[16] gives the detailed algorithm to do the clipping of triangles with the water plane.

3.5.2 Finding SWE grid cells below object

To find the SWE grid cells that are affected by the rigid body, we need to calculate the convex hull of the object. The convex hull of the object is calculated by Graham scam algorithm. The points in the above section that are clipped by the waterplane are taken as input to the graham scan algorithm. After calculating the convex hull points, the bounding box of the points is calculated and the cells within than bounding box are checked if they are object cells or not. After finding the cells which are below the object, their state is marked as 'object'. Cells who have at least one grid cell with 'object' state are marked as 'object boundary'.

After finding the 'object' cells on the SWE grid, it is checked if that grid cell was 'object' before this frame or not. A static int array is used to keep the indices of grid cells with 'object' state in the previous frame. If the grid was not in 'object' state before but is now in the 'object' state, then water is transferred from that grid to the grid cell with the nearest 'object boundary' state. This creates the waves when the object is moved in SWE grid. The water is transferred to nearby cells when the object is moved.

3.6 Boundary Conditions

For reflecting boundaries, like the steep walls in the portscene in the implementation, the cells with state 'boundary' are considered and the height from the 'near boundary' cells is copied to it. Also, the velocity components are given a value of zero depending on the location of the near boundary cell.

For the absorbing boundaries, which are at the border of SWE grid, the are damping cells defined at the border where the velocities and height of the waves are damped to zero at the last border cell in order that no wave reflects back and the use feels that the waves have been absorbed by the outer domain.

For the free surface boundaries, which are near the shore of the river, the cells are marked as 'ground' state and no advection is done on those cells. The cells are checked every frame after moving the camera about their new state, but only for the new cells. The parameters of the old cells are copied to those cells which have the same location as the previous ones.

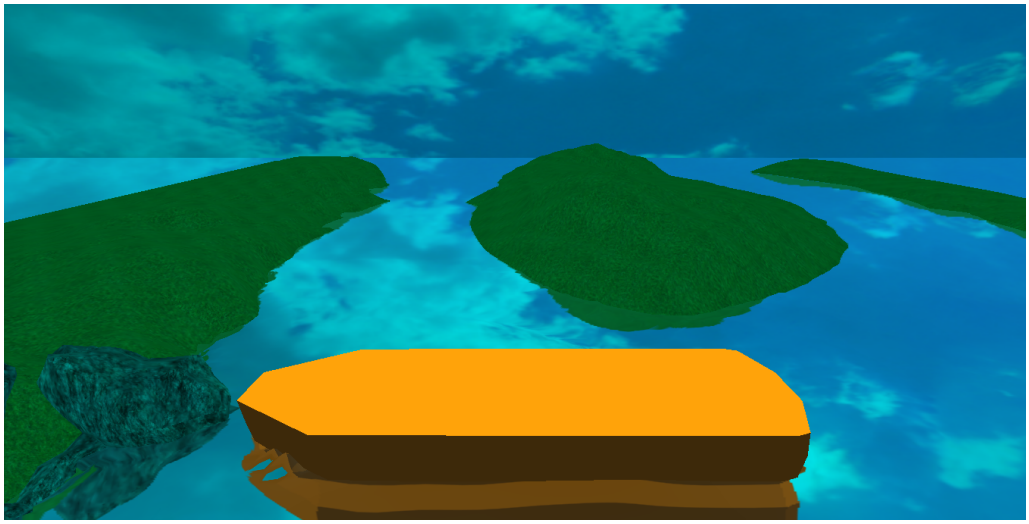


Figure 3.9: Without using Fresnel

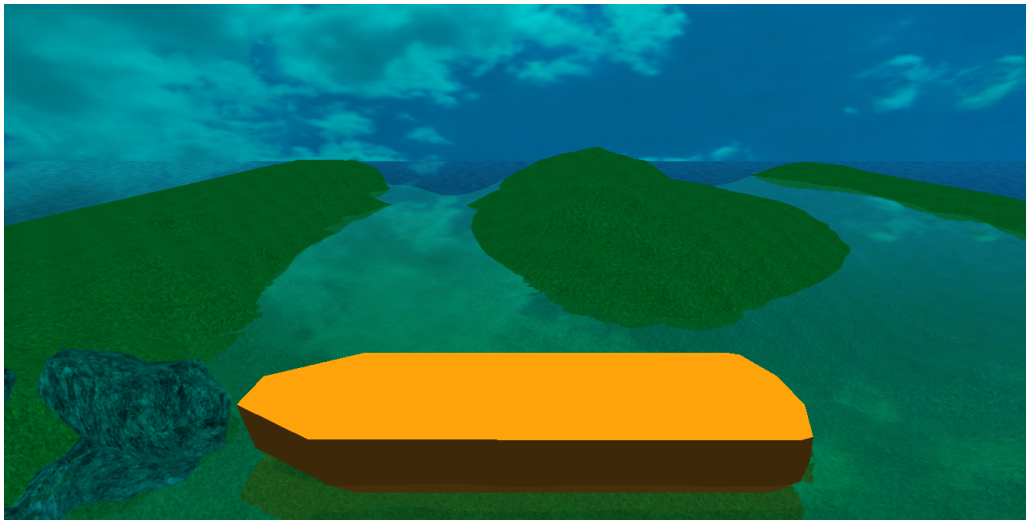


Figure 3.10: With Fresnel

3.7 Rendering

Water surface has been rendered using vertex buffer objects (VBO's) using reflection map and refraction using fresnel equations. Figure 3.9 and 3.10 show the difference between using fresnel equations for rendering.

3.8 Result

The real time water simulation including SWE algorithm, FFT based wave generation, boundary conditions and object interaction takes 3-4 milliseconds on the given hardware. The whole water simulation with user interacting with the boat runs at around 60 fps. The simulation looks plausibly good and the user interaction with the rigid body is also smooth. The aim of the project to simulate real time water simulation algorithms has been achieved successfully with user finding no distinction between the two as distraction. The reader is directed to the implementation and the binary file to see the exact implementation and result of the work.

Chapter 4

Conclusion and Future Work

The results achieved in this project look plauisbly good although there is always room for improvement. There are many algorithms and different approaches to implement different simulations and effects. This work makes no comparison of the techniques used but aims at making a real time application with water effects. This simulation will be used by GIANTS software for their further project works.

For the future work rendering and other lighting effects can make the application look better. Other effects such as foam, splashes etc. can be added for more realistic effects.

Bibliography

[1] L. Yaeger, C. Upson, and R. Myers, "Combining physical and visual simulation—creation of the planet jupiter for the film 2010," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 85–93, 1986.

[2] M. Mueller-Fischer and R. Bridson, "Fluid simulation," SIGGRAPH Course Notes, 2007.

[3] N. Foster and D. Metaxas, "Realistic animation of liquids," in *Graphical Models and Image Processing*, vol. 58, no. 5, pp. 471–483, 1996.

[4] J. Stam, "Stable fluids," in *Proc. of ACM SIGGRAPH*, pp. 121–128, 1999.

[5] N. Foster and R. Fedkiw, "Practical animation of liquids," in *Proc. of ACM SIGGRAPH*, pp. 23–30, 2001.

[6] D. Enright, S. Marschner, and R. Fedkiw, "Animation and rendering of complex water surfaces," in *ACM Trans. SIGGRAPH*, vol. 21, no. 3, pp. 736–744, 2002.

[7] J. Kim, I. Ihm, and D. Cha, "View-dependent adaptive animation of liquids," in *ETRI Journal*, vol. 28, pp. 697–708, Dec. 2006.

[8] M. Kass and G. Miller, "Rapid, stable fluid dynamics for computer graphics.," in *ACM Trans. Graph*, vol. 24, no. 4, pp. 49–55, 1990.

[9] A. T. Layton and M. V. D. Panne, "A numerically efficient and stable algorithm for animating water waves," in *The Visual Computer*, pp. 41–53, 2002.

[10] P. Hess, "Extended boundary conditions for shallow water simulations," Master's thesis, ETH Zurich, 2007.

[11] J. Tessendorf, "Simulating ocean water," SIGGRAPH course notes, 1999.

[12] M. Mueller, J. Stam, D. James, and N. Thurey, "Real time physics class notes," SIGGRAPH Course Notes, 2008.

[13] J.-P. Berenger, "A prefectly matched layer for the absorption of electromagnetic waves," in *J. Comput. Phys.*, vol. 114, no. 2, pp. 185–200, 1994.

[14] R. L. Higdon, "Radiation boundary conditions for dispersive waves," in *SIAM J. Numer. Anal.*, vol. 31, no. 1, pp. 64–100, 1994.

[15] A. G. Mastin, P. A. Watterberg, and J. F. Mareda, "Fourier synthesis of ocean scenes.," in *IEEE Comput. Graph. Appl.*, vol. 7, no. 3, pp. 16–23, 1987.

[16] E. Catto, "Exact buoyancy for polyhedra," in *Game Programming Gems 6* (M. Dickheiser, ed.), pp. 175–187, Charles River Media, 2006.