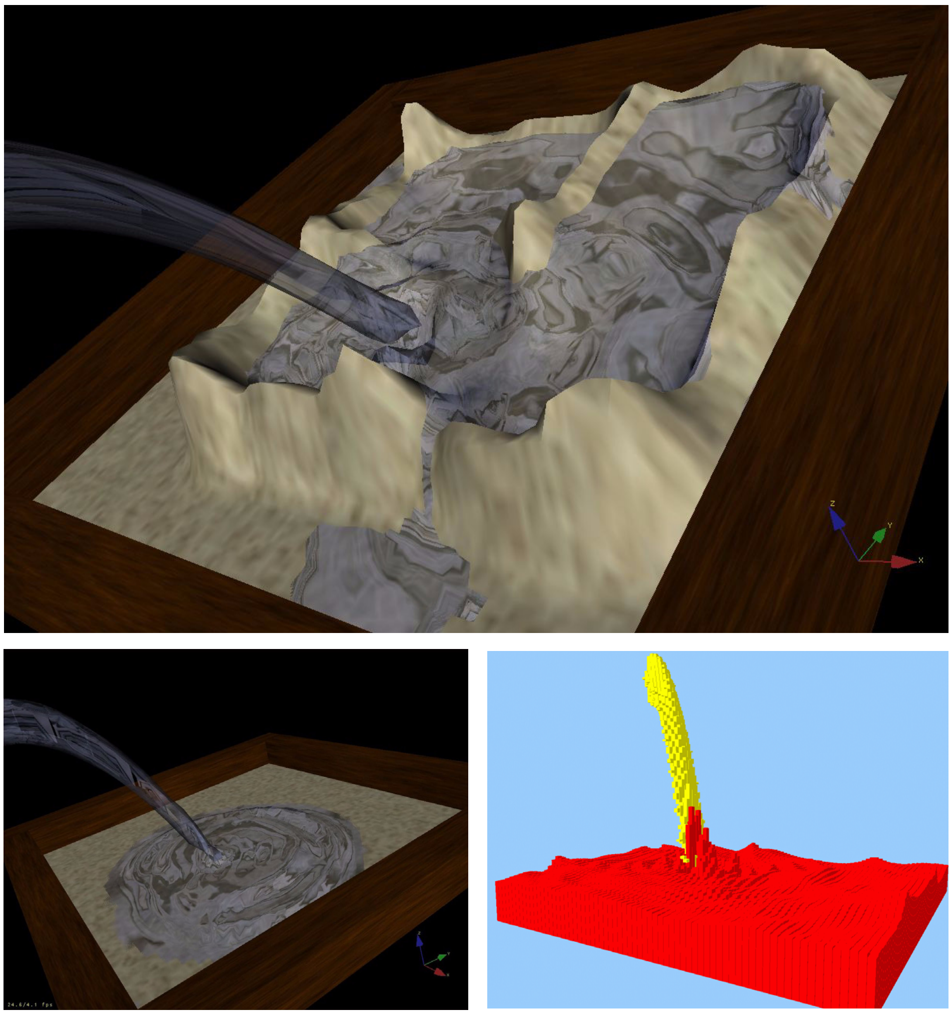
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics lab

Real-Time Fluid Simulation Using Height Fields



Semester Thesis

Bálint Miklós

Summer 2004
Advisor: Dr. Matthias Müller

Prof. Dr. Markus Gross
Computer Graphics Lab
ETH Zürich

Abstract

Realistic water simulation is a challenging domain for the computer graphics community. Water-like phenomena can be described quite accurately using the Navier-Stokes equations and, thus, convincing simulations can be generated. But the computational complexity of this approach does not allow interactive frame rates for realistic scenes on today's hardware.

To achieve real-time performance one has to simplify the generality of a Navier-Stokes based simulation in order to lower the computational time. A fast and fairly simple 2.5 dimensional method can be used for simulate water surfaces. This approach describes the water surface as a height-field, which constitutes one of the most important limitations of this model.

In this thesis an extension of the height field model is developed. The layered water model is used to overcome to a certain extent the height-field limitation, by simulating two interacting water layers. The ultimate goal of this extension is the simulation of wave breaking.

Chapter one of my thesis is an introduction to the water simulation problem and gives an overview of the water simulation methods used in the computer graphics community. In the next chapter the wave breaking phenomena is presented shortly. Upcoming chapters discuss in detail the physical model of the layered water simulation. At the end of the thesis results are presented and conclusions and further work is discussed. In Appendix A a couple of implementation details are presented.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics lab

Semester Thesis for Mr. Bálint Miklós

Real-Time Fluid Simulation Using Height Fields

Introduction

A water surface with non-breaking waves can be represented efficiently by a 2.5D height field. There exists a simple algorithm to animate height fields which is based on the 2D wave equation. However, the 2.5D representation has its limits. With simple columns of water it is not possible to animate breaking waves or splashes.

Assignment

The task of this thesis is to extend the 2.5D approach in order to make possible the simulation of new physical phenomena.

Core

- "Arbitrary underground. The algorithm should be able to cope with arbitrary terrains as undergrounds.
- "Water-air columns. The replacement of single water columns with columns containing intervals of water and air should be investigated.
- "Interaction with rigid bodies. The existing code for the interaction of rigid bodies with the surface should be extended such that bodies can sink in the fluid.

Extensions

- "Breaking waves. Friction with the underground and the use of water-air columns and horizontal velocities should be investigated for the simulation of breaking waves.
- "Splashes. Another extension is he simulation of the inclusion of air and splashes when rigid bodies plunge.

Remarks

A thesis in written form and an oral presentation will conclude this project. The advisor of the thesis is Dr. Matthias Müller, Institute for Scientific Computing, ETH Zürich.

Project start: Tuesday, April 6th 2004
Project end: Tuesday, July 13th 2004

Table of Contents

1 Introduction

Water simulation is a challenging task for the computer graphics community. Given the complexity of fluid (liquids and gases) physics it is difficult to develop a realistic and fast model simulating water dynamics. Although it is not an easy field, a lot of work has been done recently in water simulation.

After the motivation, a short overview of the main water simulation methods is presented, capturing advantages and disadvantages of particular models. Later in the chapter our layered water idea is introduced, together with the possibilities and limitations of this approach.

1.1 Motivation

Jeffrey Katzenberg (DreamWorks SKG) the producer of "Shrek" stated that the hardest shot in the movie was the pouring of milk into a glass [9]. In the "Shrek" movie there were water, mud, beer and milk effects.

Animation movies are one main application domain not just of water, but fluid simulations in general. The challenges of this problem are to find the most accurate and realistic model to simulate fluid behavior. Another task is to provide the animator with detailed control over the simulation in order to be able to achieve the desired animation. From a computational point of view there are practically no restrictions, the most important thing is to generate the desired animation. A screenshot (Fig. 1.1) from the “Shrek 2” illustrates the realism of these simulations.

Figure 1.1: Water wave animation in Shrek 2.

Some other applications like computer games, medical simulations or virtual environments, interactive frame rates, and real-time interaction with the user have to be achieved with limited computational resources. Of course having these limitations, one has to give up full accuracy and photorealism that can be achieved in offline simulations. The challenge for these applications is to find a fast model which sufficiently approximates the fluid dynamics.

The different methods presented in the next section have to find a trade-off between simulation accuracy, realism and computing requirements.

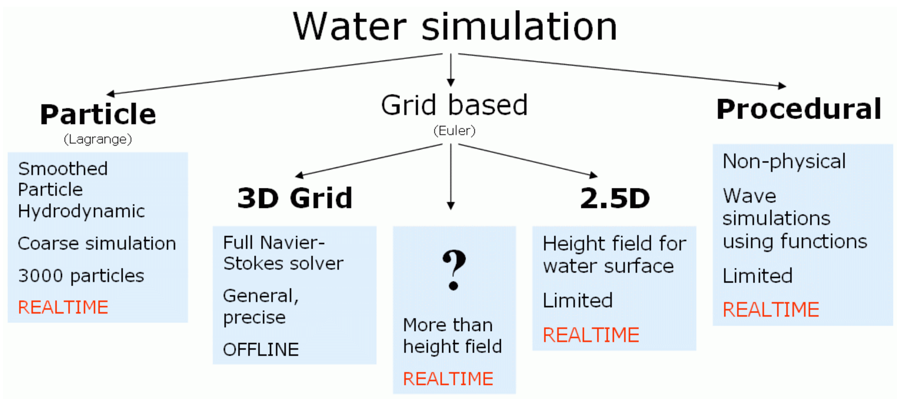


Figure 1.2: Overview of the main water simulation methods, and our new approach.

1.2 Related Work

There are three main directions in water simulation (Fig. 1.2): methods based on spacial grids, those using particles to simulate water volume, and the non-physical procedural methods.

The particle based methods model the volume of the water using a set of particles (Lagrangian approach). This method implicitly insures volume conservation, only pressure and viscosity have to be computed. This method has been used for simulating fluids, water and melting objects. The big advantage is that simulations can be computed at interactive rate, thus enabling real-time applications. For example the Smoothed Particle Hydrodynamics based method (Fig. 1.3) can produce simulations with interactive rates using models consisting of up to 5000 particles. [16]

Figure 1.3: Particle based water simulation using SPH system. Interactive frame rates.

Procedural approaches do not compute solutions of physical models, but use parametric functions and mathematical descriptions to approximate the desired phenomena. There were several works to simulate water waves, ocean surface simulations using this approach. [7, 10, 19, 24]

Grid methods divide the simulation space into small cells, and computations are done on this finite number of cells. The fluids are described by velocity, density and pressure fields. (Euler method) For integration, conservation of mass and momentum equations are used. Very often the solutions of the different forms of the Navier-Stokes equations are computed [14, 3, 6]. Because of the complexity of these equations the simulation usually can not be computed in real-time. On the other hand, the simulations are calculated on the whole simulation domain, thus enabling simulation of a wide variety of phenomenons. The results are very convincing and the simulations look very precise and realistic as for example in Carlson's [3] water-rigid body simulation (Fig. 1.4).

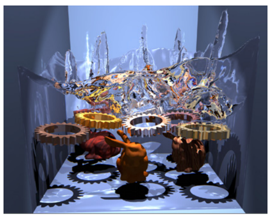


Figure 1.4: Carlson's Navier-Stokes method simulates rigid body - water interaction.

A variation of the grid approach [11, 20] simplifies the model in order to allow fast computations. Using a 2 dimensional discretization of columns instead of a full 3d "cube grid" a height field surface of the liquid can be simulated very efficiently. The water surface is described by the height of the columns. The results can be computed without problem at interactive rates, but the variety of phenomena that can be simulated is quite limited. The biggest limitation of this approach is the assumption that the water surface can be described by a function (Fig. 1.5).

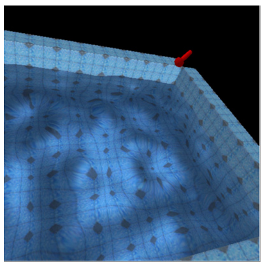


Figure 1.5: Schneider's height field simulation. GPU accelerated method with refraction rendering.

In this work the extension possibilities of this height field model is studied. The goal is to develop enhancements that can still be computed at interactive frame rates.

1.3 More than Height Field

The height field approach produces quite convincing water surface simulation, including very nice wave propagations. But this simple model is not able to approximate the very complex non-linear phenomena of wave breaking. Apart from these nonlinear effects the water surface of an ocean can be easily described by a height field. Therefore it seems logical to try to extend the height field model to be able to simulate breaking waves as well.

The first impediment to simulate such phenomena is the incapability of the model to handle more than one vertical value for a position in the horizontal plane. The other problem is to find a fast physical model, that can animate the evolution of a wave, during this breaking process.

Our extension ideas are the following:

- Use a two-layered bottom and upper water model instead of the simple height field
- When the height field model can not realistically describe the water wave, move water from the bottom to the upper layer

- Animate the upper layer based on the wave speed information from the bottom layer
- As soon as the wave breaking process is finished, i.e. the upper layer touches the bottom layer, merge again the two layers into the height field model

These extensions would allow real-time simulations like ocean shores with breaking waves or pouring of water into a glass. Complex phenomena like colliding waves are beyond the capabilities of this model firstly because of the restriction of two water layers. Another problem would be the speed of the physical model needed for animating these very complex dynamics.

2 Breaking Waves

2.1 Wave Breaking Phenomenon

Wave breaking is one of the most commonly observed features of water waves. This phenomenon is nonlinear, and very difficult to describe anatically, some consider it the most difficult wave phenomenon to describe mathematically. At the present, very few properties of breaking waves can be predicted with reasonable accuracy. The famous painting (Fig. 2.1) of Hokusai illustrates the complexity and the different scales of the wave breaking process.

Figure 2.1: “The Great Wave” by Hokusai, Japan’s best known plastic artist.

But this is only one “type” of wave breaking, the plunging mode. Waves may break in a number of different ways [8]. More generally, the essence of most nonlinear wave phenomena is due to a depence of the wave speed on the wave amplitude. Steep waves on mild slopes tend to break by spilling water gently from their crests, and there is little reflection of the incident wave energy. Long low waves on steep slopes tend not to break at all. Instead they surge up and down the slope with most of the wave energy being reflected.

Battjes [1] introduced the “surf similarity parameter” or Iribarren number, N_I, to describe different wave behaviours:

$$N_I = \frac{\tan\beta}{\sqrt{H/E_o}}, \tag{2.1}$$

where:

$$E_o = gT^2/(2\pi) \tag{2.2}$$

and β is the slope angle, the wave height H is measured at the toe of the slope and T is the wave period. The N_I values are associated with different wave actions as it is shown on Fig. 2.2.

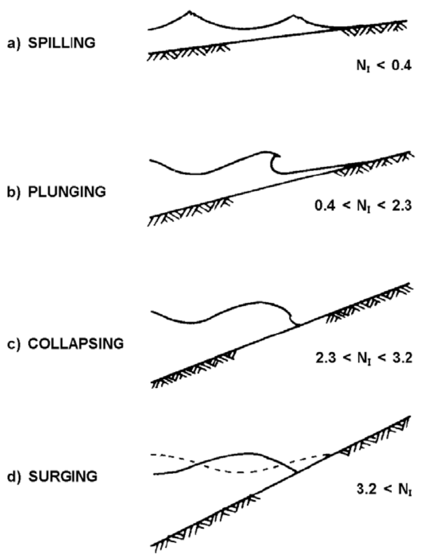


Figure 2.2: Different types of waves and their surf similarity number values

In case of deep water only the wavelength is important, and breaking occurs when the wave steepness, *H/E*, is approximately 0.14. According to Miche's expression [13] in shallow water depths, d_b the maximum wave height, H_b can be approximated by

$$H_b = 0.88 \cdot d_b \tag{2.3}$$

2.2 Plunging Breaking Waves

An often observed and experienced form of wave breaking is the plunging wave breaking. This process can be explained heuristically as follows. The closer the water is to the ground, the more it is deccelerated by the friction with the underground. More exaclty the speed of the wave is $\sqrt{gd}$, where d is the undisturbed water depth (Fig. 2.3). Therefore, since the top of the wave travels the fastest, the wave will steepen toward the front and eventually topple over. This means, that in case of shallow water the wave always breaks if we wait long enough. Of course, in case the wave is climbing a slope the overturning and the breaking will occur sooner.

One straightforward way to simulate plunging waves is to use a 3D Eulerian approach, and using the Navier-Stokes equations to generate the simulation. These equations allow a precise and realistic animation. The recently presented method (Fig. 2.4) of Mihalef et al. [14] offers a user friendly interface to the animator, as well. But these animations need 10-12 min/frame computational time on a Pentium 3, 2.6 GHz machine with 2 Gb od RAM for a resolution of $128 \times 128 \times 64$.

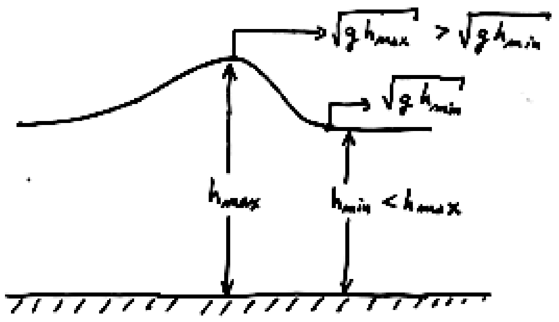


Figure 2.3: Wave speed in shallow water

Peachey [19] developed a model which is based on the so called Airy model using shape- and phase functions to describe the shape of the waves approaching the shore. Fournier's [7] and Tessendorf [22] method uses functions based on the trochoid functions to generate wave refraction. The procedural model of Jeschke et al. [10] allows a fast, interactive rate wave breaking model, but requires a lot of work from the animator, because everything has to be modeled by hand, since no information is transmitted over the time.

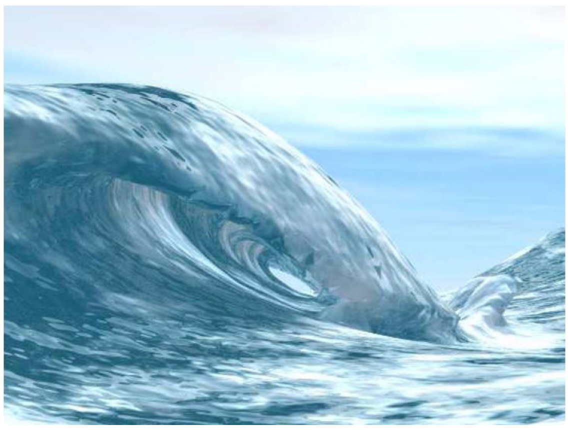


Figure 2.4: Wave simulated by Mihalef et al. 10-12 minutes computation per frame.

2.3 Layered water for plunging waves

The layered water modeling method should be placed between these two main directions, the full 3D physical simulation, and the fast, more empirical animations.

The idea is to use two models for the simulation of the wave. The bottom layer is a height field simulation of water surface, which has to capture all possible features of a wave until the moment the wave can not be described by the height field anymore. This means, that the wave shape should change in accordance to the underground, it should steepen as approaching the shore. Once the height field simulation is approaching the "shock" situation, water should be taken from the bottom layer into the upper layer. From this moment there are two different layers of water simulated. Because of gravity, after a while the water from the upper layer will touch again the bottom layer (or the underground, if there is no more water). At this meeting point the water from the upper layer is taken to the bottom layer to finish wave breaking process. Of course special attention has to be accorded to the state conservation of the water which is taken from one layer to the other one.

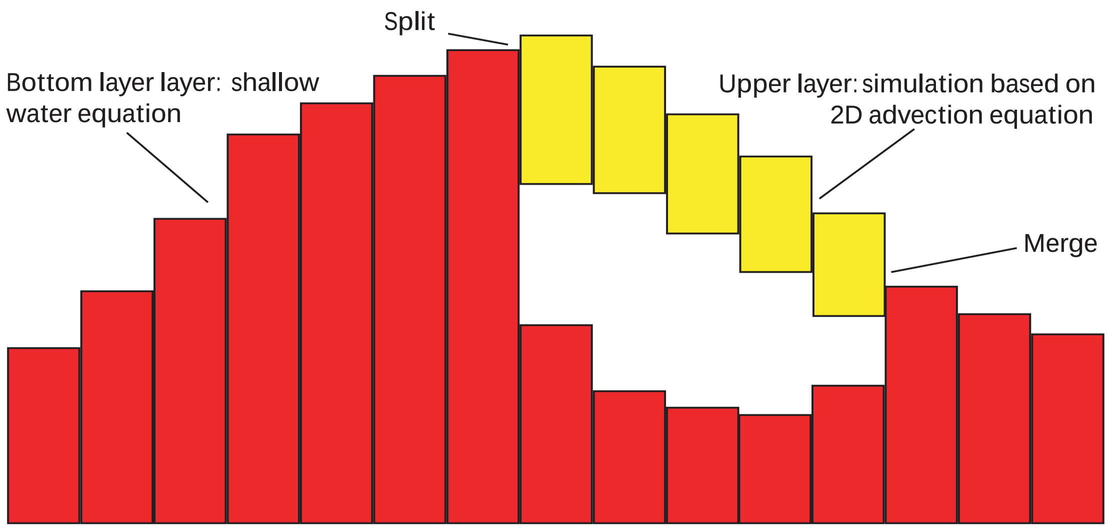

Split
Bottom layer layer: shallow water equation
Upper layer: simulation based on 2D advection equation
Merge

Figure 2.5: Layered water wave breaking idea. The red color is the bottom layer, the yellow marks the upper layer.

In the next chapters the layered water approach (Fig. 2.5), a fast, physically based model capable of the following features is presented:

- Height field simulation for shallow water waves, including steepening, deccelerating of waves, interaction with the underground.
- Animation method for animating water in the upper layer. The initial state and gravity will determine the water "trajectory"
- Merging method of upper and bottom layers including transfering upper layer's velocity

In order to extend the model to animate wave breaking, a criterion has to be found saying how much water has to be transfered from the bottom layer to the upper layer and when this transfer should occur.

3 Bottom Layer

The bottom layer simulation uses existing results from the literature. There has been a couple of papers discussing and presenting height field based water simulation. The governing mathematical equations in these methods are based on the two dimensional wave equation.

3.1 Wave Equation

The wave equation is the partial differential equation that describes the wave propagation with speed *c*. For clarity I will discuss the one dimensional version of the equation, but everything presented can be easily extended in two dimensions. Let us denote with function $h(x, t)$, the height of the wave in point *x* at time *t*. The one dimensional wave equation has the following form:

$$\frac{\partial^2 h}{\partial t^2} = c^2 \cdot \frac{\partial^2 h}{\partial x^2} \qquad (3.1)$$

Let's consider $h(x, t) = A \cdot f(x + c \cdot t) + B \cdot f(x - c \cdot t)$, with arbitrary reals *A* and *B*. We notice that $\frac{\partial^2 h}{\partial x^2} = A \cdot \frac{\partial^2 f}{\partial x^2} + B \cdot \frac{\partial^2 f}{\partial x^2}$, and $\frac{\partial^2 h}{\partial t^2} = c^2 \cdot \left(A \cdot \frac{\partial^2 f}{\partial t^2} + B \cdot \frac{\partial^2 f}{\partial t^2}\right)$. This means, that any function of this form is an analytical solution of the wave equation. As the form of function *f* and the figure shows, this equation animates the travel of a wave with a constant speed. Practically it simulates a water wave propagation for an infinite depth (Fig. 3.1), and it has been used by the computer graphics community for real-time fluid surface animation.

This equation is a good starting point, but we all know that shallow water waves change their shape and speed in function of the water depth, the underground shape.

3.2 Shallow Water Equation

Starting from the Navier-Stokes equations of fluid flow we can deduce a vastly simplified set of equations that has widely used for shallow water. The simplification arises from three approximations. Firstly, we assume that the water surface is a height field. The second approximation is that the vertical velocity of water particles can be ignored. The third assumptions is

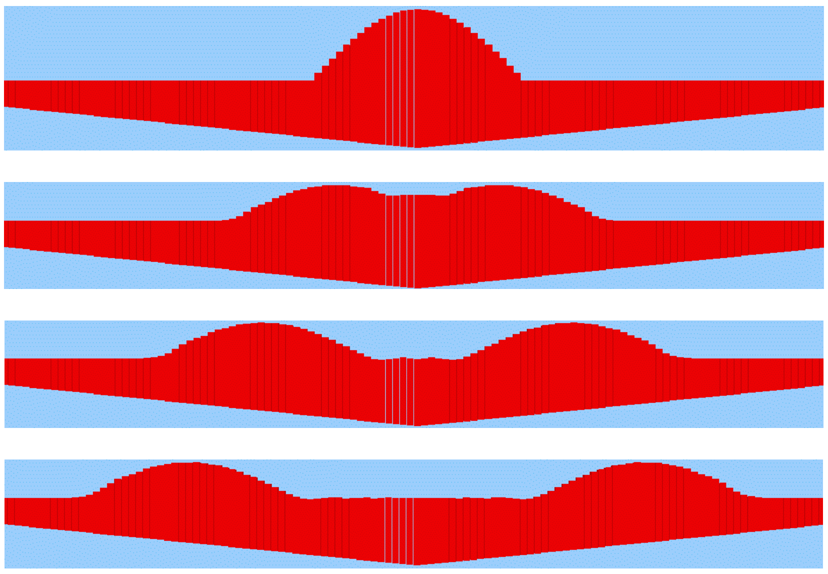


Figure 3.1: Animation using the wave equation. The top picture is the initial state, the frames below show the travel of the unchanged wave shape

that the horizontal component of the speed of the water in a vertical column is approximately constant. These assumptions limit the model accuracy, since turbulent flows or unusually high friction on the bottom can not be simulated.

Let us extend our notations by *b(x)* for the underground height, *d(x,t)=h(x,t)-b(x)* for the depth of the water, and *u(x,t)* for the horizontal velocity of a vertical column. Using the above assumptions the following are the shallow water equations [5]:

$$\frac{\partial u}{\partial t} + u \cdot \frac{\partial u}{\partial x} + g \cdot \frac{\partial h}{\partial x} = 0 \tag{3.2}$$

$$\frac{\partial d}{\partial t} + \frac{\partial}{\partial x}(ud) = 0 \tag{3.3}$$

Equation 3.2 expresses Newton’s law *F=ma*, and Equation 3.3 ensures volume conservation. Even with these assumptions the equations are non-linear. Assuming the fluid velocity is small and the depth is slowly varying, we can get following expressions after ignoring the second term of Equation 3.2 and linearizing around a constant value of *h*:

$$\frac{\partial u}{\partial t} + g \cdot \frac{\partial h}{\partial x} = 0 \tag{3.4}$$

$$\frac{\partial h}{\partial t} + d \cdot \frac{\partial u}{\partial x} = 0 \tag{3.5}$$

Now, if we differentiate the Equation 3.4 with respect to *x*, and Equation 3.5 with respect to *t* we end up with:

$$\frac{\partial^2 h}{\partial t^2} = gd \cdot \frac{\partial^2 h}{\partial x^2} \tag{3.6}$$

Equation 3.6 is called the shallow water equation [shallow water 2]. If we compare it with the wave equation (Equation 3.1) we notice that our new equation is actually the wave equation with speed $c = \sqrt{gd}$. Here c is not a constant anymore but a function of x and t, since it depends on the depth of the water. Moreover, we notice that the speed function is exactly the speed of the water particles of the plunging wave.

This equation would allow us to generate simulations with refracting waves (Fig. 3.2), that change their shapes on slopes because now column speeds are not going to be constant for the whole simulation, but will be always locally dependent on the water depth.

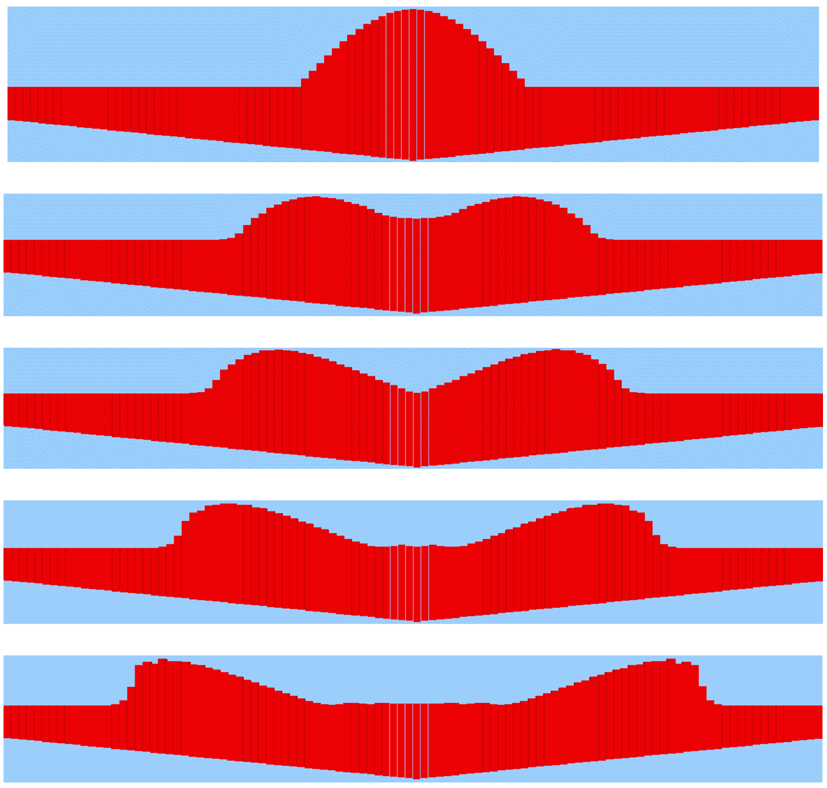


Figure 3.2: The shallow water wave. The wave steepens because of the slope.

Once we have a numerical solution for this equation we can simulate shallow water height field that interacts with the underground. Kass and Miller, who have first introduced this equation for the computer graphics community [11] used a finite difference discretization and a first order implicit solution. In the layered water model we use another finite difference solution, which will be presented in the next section.

3.3 The Pipe Model

O'Brien and Hodgins introduced a pipe model [18] which uses the same assumptions as the shallow water equation, and models the water surface as a height field. Their discretization is exactly the same, using columns of water. The animation of the surface is done by moving water between neighboring columns. To do that the model uses axis aligned pipes between adjacent columns (Fig. 3.3 and Fig. 3.4).

Based on the laws of the hydrodynamics the pressure at column *(i,j)* of the grid is

$$P_{ij} = h_{ij}\rho g + p_0, \qquad (3.7)$$

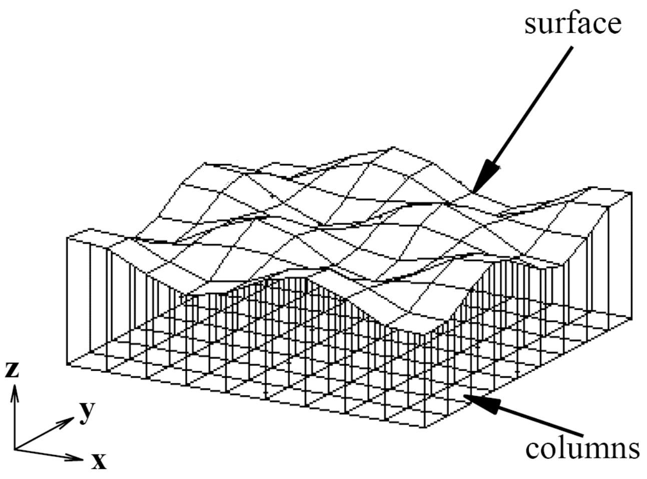


Figure 3.3: Height field model to represent the surface.

where ρ is the density of the water. The acceleration of the water in the pipe is determined by the pressure difference of the two columns *(i,j)* and *(k,l)* which the pipe connects:

$$a_{ij \to kl} = \frac{A(\Delta P_{ij \to kl})}{m}, \tag{3.8}$$

where *A* notes the cross area of the pipe. Using *L* for the length of the pipe and substituting Equation 3.7 we get the following expression for the acceleration:

$$a_{ij \to kl} = \frac{g \cdot (h_{ij} - h_{kl})}{l} \tag{3.9}$$

The water flow between the two adjacent columns based on an Euler timestep will be:

$$q'_{ij \to kl} = q_{ij \to kl} + \Delta t \cdot A \cdot a_{ij \to kl} \tag{3.10}$$

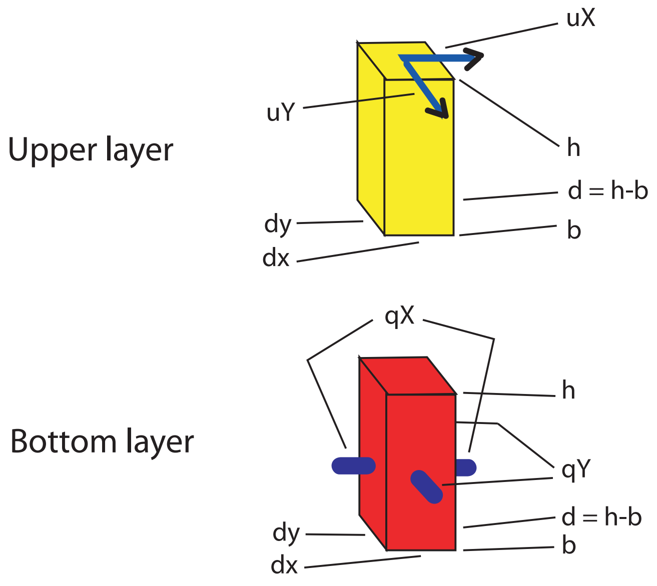


Figure 3.4: Bottom and upper layer columns

Finally the water column height change in a timestep can be computed with the following formula:

$$h'_{ij} = h_{ij} + \sum_{kl \in \eta_{ij}} q_{ij \to kl} / (\Delta x \cdot \Delta y), \tag{3.11}$$

where Δx and Δy are the distance between the mesh grid points, thus their product gives the base area of a column. The integration algorithm could use the Equation 3.9 and Equation 3.10 to calculate the new flows, and Equation 3.11 for updating the column heights.

If we use the

$$\Delta h_{ij} = \frac{\sum_{kl \in \eta_{ij}} (h_{kl} - h_{ij})}{\Delta x \cdot \Delta y} \tag{3.12}$$

approximation for the Laplacian of h, and integrate the two dimensional wave equation using the simplest explicit method we will end up with a very similar expression for h:

$$u'_{ij} = u_{ij} + \Delta t \cdot c^2 \cdot \frac{\sum_{kl \in \eta_{ij}} (h_{kl} - h_{ij})}{\Delta x \cdot \Delta y} \tag{3.13}$$

$$h'_{ij} = h_{ij} + \Delta t \cdot u_{ij} \tag{3.14}$$

If we compare these expressions with the explicit solution of the wave equation we will find that, actually, the results are the same in case we substitute $c^2 = \frac{g \cdot A}{L}$.

Hence by modifying the measures of the pipes we can modify the speed of water flow, respectively the wave velocity. If we choose A and L constant values for the whole simulation domain, we will actually solve a two dimensional wave equation by animating with the pipe model. In case we choose $\frac{g \cdot A}{L} = g \cdot d \Leftrightarrow \frac{A}{L} = \bar{d}$, the pipe model will solve actually the shallow water equation. In this notation $\bar{d}$ represents the algebraic mean of the depths of the two neighboring columns. Intuitively this would mean, that the pipe sizes dynamically change in time based on the depth of the water on the certain position.

One could ask, why to go all around the pipe abstraction and flow values to calculate get a result, which we could have calculated with exactly the same accuracy but using less memory. It is true, that for the flow values we need more memory space, but we will have some directional flow/velocity information, as well. This information is not needed in this context, but we will see later, how in the layered water model will need these flow values.

3.4 Negative Values

Until now we did not consider the precision and stability of our discretized numerical solution. Just as any other explicit differencing scheme, our solution is subject to the Courant-Friedrichs-Lewy stability criterion:

$$\Delta t < \frac{\Delta x}{u} \tag{3.15}$$

In case we are dealing with theoretically infinite depths, there is no danger of getting invalid results. But once we are solving the shallow water equation, it can very easily happen, that after a timestep some columns will contain negative amount of water. This strange situation can happen because of the discretization in time. The water exchange between columns is determined by the situation at the beginning of the timestep. If the timestep is relatively large, then it can happen that the acceleration calculated at the beginning of the timestep could move too much water out of the column, which leads to negative water values. This problem happens in case of explicit or even implicit solutions of the shallow water equation.

The solution used by Kass and Miller [11] checked the total volume of water in every connected component of water before and after the timestep. If the volumes are different, distribute uniformly the difference over the columns in the connected region.

O'Brien's solution [18] is a faster method. He proposes to correct negative value columns by moving water from the neighbor columns. This solution does not provide such a guarantee of stability like the one form Kass and Miller, but it is much faster to compute than the first method. Our experience shows that this method is satisfactory in practice. So, all we need to do is to go through all the columns, and process the columns of negative amount of water. The correction function finds all the neighboring columns that take water from the column, and collects back as much water which will bring the column to the zero level. The water amount removed from the neighbor columns is proportionally distributed based on the flow values. The more water a column took, the more role has to have in correcting the negative value.

In the first step of the correction we sum up all the flows from the pipes which took water from the negative column. Like this, we can calculate the "contribution" of a column, i.e. the ratio between its flow and the sum taking flow. Now we distribute the negative water volume among the taking columns directly proportional with their contributions.

3.5 Diffusion term

Concerning the performance of this numerical solution, we have experienced quite strong oscillations in case when some big changes or discontinuities appear in our height field. These high frequency disturbances are not dumped, and once appeared will never disappear from the simulation.

On the other hand, even the theoretically perfect solution would not really look realistic because of the lack of damping in the system. Once a wave started, it would infinitely travel back and forth on our simulation domain.

To overcome these two problems we have introduced a diffusion term of the height field in the simulation. This term imitates very nicely the natural behavior of the water surface, including damping and eliminating the annoying high frequency oscillations (Fig. 3.5). Again, I will present the one dimensional version, and later in the numerical solution I give the results for the two-dimensional version.

The diffusion equation has the following form:

$$\frac{\partial h}{\partial t} = D \cdot \frac{\partial^2 h}{\partial x^2}, \tag{3.16}$$

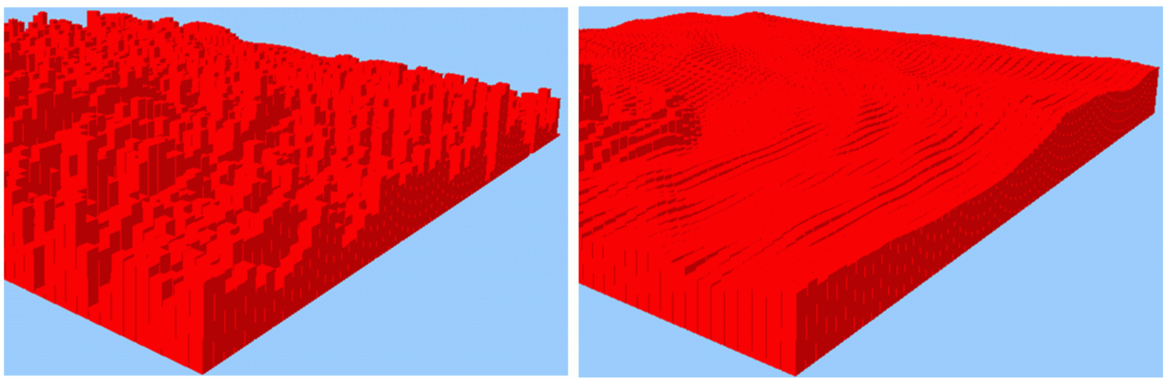


Figure 3.5: Results without diffusion and with diffusion term.

where D is called the diffusion coefficient. Combining this equation with the shallow water equation (Equation 3.6) we get:

$$\frac{\partial^2 h}{\partial x^2} = \frac{1}{D} \cdot \frac{\partial h}{\partial t} + \frac{1}{gd} \cdot \frac{\partial^2 h}{\partial t^2} \tag{3.17}$$

The two dimensional version will have the Laplacian on the left hand side of the equation. It is straightforward to extend the pipe model to incorporate this diffusion term as well, since the diffusion contains only first order derivative with respect to the time. The only modification is that to height value calculated with the pipe model one has to add the discrete version of $D \cdot \left(\frac{\partial^2 h}{\partial y^2} + \frac{\partial^2 h}{\partial y^2}\right)$. This means, that diffusion term can be calculated in one step, but a temporary copy of the height field has to be used in order to be able to calculate the finite difference approximation of $\Delta h = \frac{\partial^2 h}{\partial y^2} + \frac{\partial^2 h}{\partial y^2}$ (Equation 3.12).

3.6 Algorithm

The final algorithm for solving an oscillation free, volume conserving, shallow water equation based water flow on an arbitrary ground will look the following way. It uses a $w \times r$ regular grid, and needs to transfer the following information from one timestep to the other one.

- h matrix for the height of the water ($w \times r$),
- qX matrix pipe flows in x direction ($(w-1) \times r$)
- qY matrix pipe flows in y direction ($w \times (r-1)$)

In a timestep a temporary local value of the height field will be needed, as well. The following operations are needed for every simulation steps:

1. Update every qX and qY value by calculating acceleration based on adjacent column heights.
2. Save old copy of height field
3. Update every column height with the diffusion term calculated from the temporary height values and with the water exchange based on pipe flows.

4. Correct every negative value column by correcting it with water from neighboring columns

Remarks:

- A column containing no water has the height of the ground (h=b)
- In order to save predictable negative value corrections we can filter out acceleration of water flow out of an empty column. This typically happens on a slope with no water.

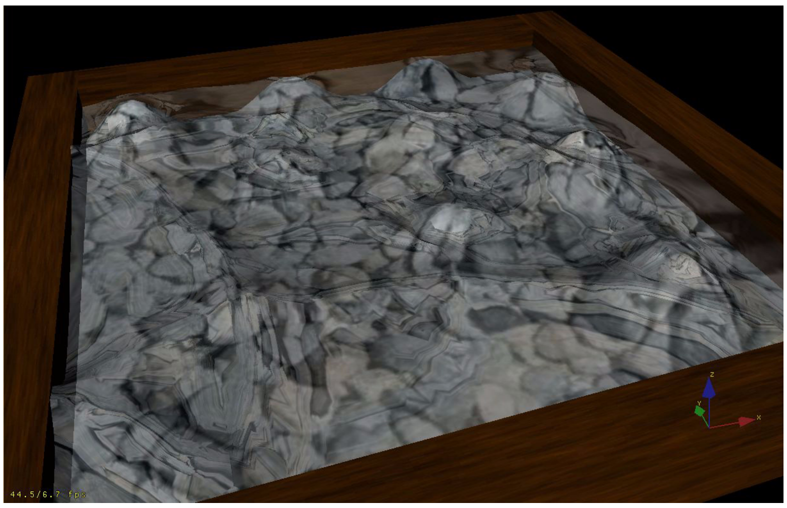


Figure 3.6: Height field water surface. Semi transparent interpolated surface with environment texture mapping

This shallow water pipe model results in a realistic simulation of a height-field water surface which interacts with the underground. (Fig. 3.6)

4 Upper Layer

Our first try to implement the upper layer water dynamics tried to use a similar model as in the bottom layer. But this approach can not capture the main driving forces of the water “moving in the air” since in this case not the pressure but gravity and the initial velocity are the only driving factors of the movement.

Actually what happens is the following: an amount of water is transported in the direction and with the speed which the upper layer was initialized. In addition, the water is subject to the effect of the gravitational force, as well. This means, that actual water particles are transported horizontally with a constant speed and accelerated vertically with *g*. Note that in our setting the water particles do not correspond to the column, since the columns are fixed spacially. This means that in order to stick to our column settings we have to **transport some quantities like water amount, velocity and position through the grid**.

4.1 Advection Equation

We will use another partial differential equation for our model, this time the linear advection equation. The one dimensional version of this parabolic equation has the following form:

$$\frac{\partial s}{\partial t} + \frac{\partial}{\partial x} us = 0\,, \tag{4.1}$$

where *s(x,t)* is the scalar to be transported a *u* is the velocity field which transports *s*.

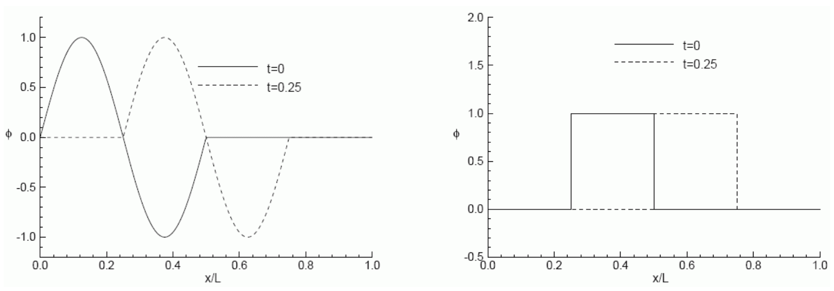


Figure 4.1: One dimensional advection

The analytical solution of this equation is very similar to the one of the wave equation:

$$s(x, 0) = f(x) \tag{4.2}$$

$$s(x, t) = f(x - vt) \tag{4.3}$$

What this form suggest is, that to get the solution at any time t we just have to go ut along the space dimension starting from the current x value and look up the value on the initial shape, *f(x)*. The initial shape will travel based on the value of u (Fig. 4.1). Actually a scalar value is transported through the grid.

In the two dimensional version the velocity field becomes a vector that determines the transport along both, the x and y axis.

4.2 Numerical Schemes

We will use a finite volume discretization to solve Equation 4.1. For that we will use the notations presented in Fig. 4.2. P is the cell of interest. With capital letters are noted the cells,

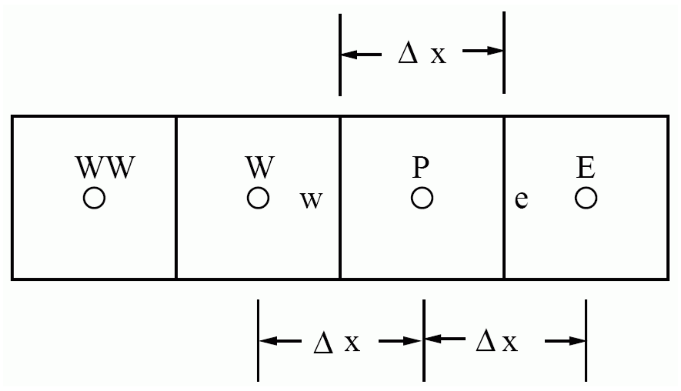


Figure 4.2: One dimensional finite volume discretization of the 1D advection equation.

and small letters denote cell faces. As you can see, the cellsize is Δx.

Considering s changing only linearly over a cell and by integrating Equation 4.1 over the cell P we get:

$$\left(\frac{\partial s}{\partial t}\right)_P \Delta x + (u_e s_e - u_w s_w) = 0 \tag{4.4}$$

Since in our setup, the linear advection equation, u is known, the problem for determining the cell flux reduces to the determination of the s_e and s_w face values. For simplicity, in this discussion we will assume that the velocity u is constant over the domain. By choosing different approximations we can derive several integration schemes.

4.2.1 Central Difference Scheme

The first straightforward try would be the arithmetic mean of the neighboring cells. This scheme is known as the central difference scheme:

$$\frac{s'_P - s_P}{\Delta t} + v\frac{(s_E - s_W)}{2\Delta t} = 0 \tag{4.5}$$

This scheme is second-order accurate in space and first order accurate in time, but the von-Neumann stability analysis shows, that this scheme is unconditionally unstable. The instability of the scheme is the result of the combination of the space and time discretization. Using implicit integration instead of the explicit, we get an unconditionally stable method, but we do not have the physical plausibility of the solution.

4.2.2 First Order Upwind Scheme

As a next try we could use face values influenced only be cells from the transport direction. This means, if we assume we have a positive u, the face s_e will be influenced only by cell P, and not by E anymore. By simply assigning $s_e = s_P$ and $s_w = s_W$ we obtain the first order upwind scheme:

$$\frac{s'_P - s_P}{\Delta t} + u\frac{s_P - s_W}{\Delta x} = 0 \tag{4.6}$$

The stability analysis of this scheme reveals that the scheme is stable, as long as the Courant-Friedrich-Levy criterium (Equation 3.12) is satisfied.

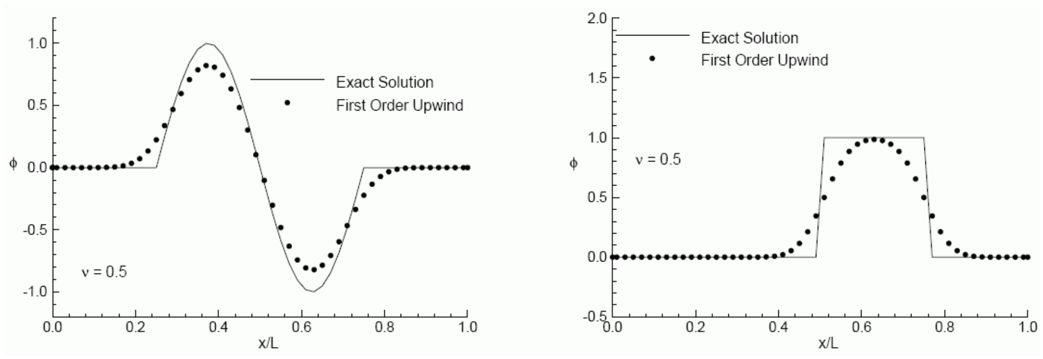


Figure 4.3: The performance of the first order upwind scheme

This scheme generates the right amount of translation, but only after 25 five steps there are noticeable differences between the initial and the advected shape. (Fig. 4.3). Discontinuities and slopes are considerably smoothed. A simple error analysis using Taylor series expansions shows that the most significant error terms are very similar to the diffusion term already discussed. This effect called artificial, false or numerical diffusion can be seen on our plots, as well.

Unfortunately, because of this artifact the first order upwind scheme will be unusable for us. To overcome the artificial diffusion the Lax-Wendroff scheme adds an extra diffusion term to the equation. This method is more expensive to calculate but it is second order accurate in both space and time. Its performance it is superior to the first order upwind scheme, but in discontinuities will introduce spurious “wiggles”.

4.2.3 Second order upwind schemes

There is another way to overcome big artificial diffusion of the first order upwind scheme. Remember, that in our discussion of the upwind scheme we wanted to approximate the face values from the direction where the transport comes from. But all we did was simply to use the nearest cells value. Higher order upwind schemes use higher order approximations for face values.

Let us consider the Taylor series expansion of the point P:

$$s(x) = s_P + (x - x_P)\frac{\partial s}{\partial x} + \frac{(x - x_P)^2}{2!}\frac{\partial^2 s}{\partial x^2} + O(\Delta x)^3 \tag{4.7}$$

If we evaluate the first two terms of the Equation 4.7 in the point $x_e = x_P + (\Delta x)/2$ we obtain:

$$s_e = s_P + \frac{\Delta x}{2}\frac{\partial s}{\partial x} \tag{4.8}$$

Using this expression we will have a second order error on the face value estimation. Based on the approximation of the partial derivative of u, there are two schemes known in the literature. The Fromm [23] scheme uses $\frac{\partial s}{\partial x} = \frac{s_E - s_W}{2\Delta x}$, to end up with the final formula:

$$s_e = s_P + \frac{s_E - s_W}{4} \tag{4.9}$$

The Beam-Warming [2] scheme approximates the derivatives using the E and W cell values, and has the following form:

$$s_e = s_P + \frac{s_P - s_W}{2} \tag{4.10}$$

4.2.4 Third Order Upwind Scheme

We can go further and use the first three terms of the Equation 4.7 to insure more accuracy of our scheme. The third order upwind scheme uses the following approximations for the partial derivatives:

$$\frac{\partial s}{\partial x} = \frac{s_E - s_W}{2\Delta x} \tag{4.11}$$

$$\frac{\partial^2 s}{\partial x^2} = \frac{s_E + s_W - 2s_P}{(\Delta x)^2} \tag{4.12}$$

The final expression for the face value is:

$$s_e = \frac{s_E + s_P}{2} + \frac{s_E + s_W - 2s_P}{4} \tag{4.13}$$

This scheme is known as the Quadratic Upwind Interpolation for Convective Kinetics (QUICK) scheme [12]. The two presented second order and the QUICK scheme can be combined into a single expression [15]:

$$s_e = s_P + \frac{1-k}{4}(s_P - s_W) + \frac{1+k}{4}(s_E - s_P) \tag{4.14}$$

For *k=-1* we get the Beam-Warming, for *k=0* the Fromm, and for *k=0.5* the QUICK scheme formula. Similarly, the s_w can be approximated, only the indices have to be “shifted” with one cell:

$$s_w = s_W + \frac{1-k}{4}(s_W - s_{WW}) + \frac{1+k}{4}(s_P - s_W) \tag{4.15}$$

Now substituting Equation 4.14 and Equation 4.15 in Equation 4.4 we will get the integration scheme for the one dimensional advection. In two dimensional case we just simply advect once the scalar along the x axis, and then along the y axis.

The QUICK method eliminate a big part of the artificial diffusion [15, 25] (Fig. 4.4), is pretty simple to implement, and since it is based on the upwind technique it is conditionally stable. But we have to be aware of the fact that using this scheme we get similar small errors at discontinuities like the Lax-Wendroff scheme has.

Now we have a relatively accurate and fast advection scheme on our column grid. The next section will present which measures and how have to be advected in order to simulate moving water in the air using a grid.

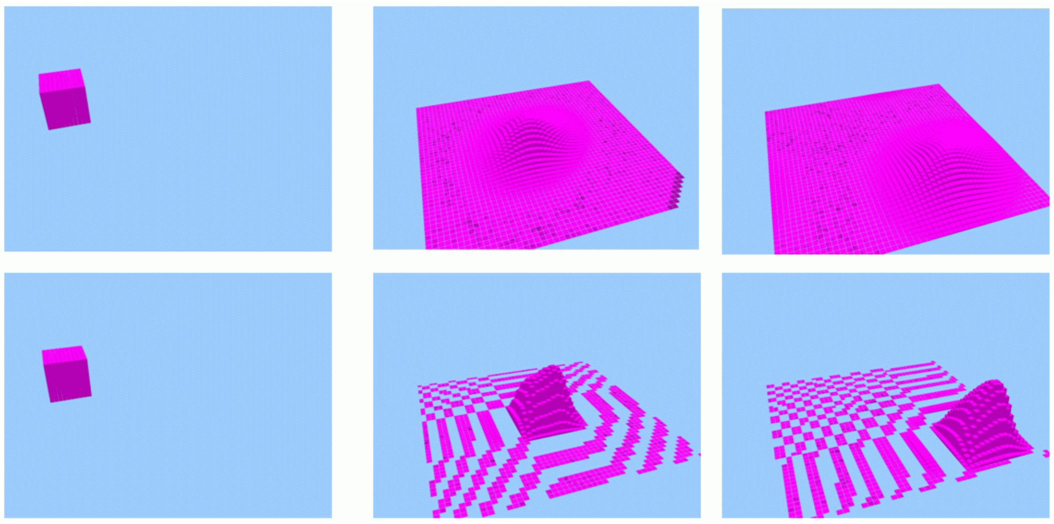


Figure 4.4: The top row shows a first order advection of a cube. The lower frames are the result of the same animation with the QUICK integration.

4.3 Simulation Algorithm

The first step in developing the model for simulating the water in the air is to consider only the horizontal movement. We want to see only a certain amount of water traveling in a certain direction. i.e. a simple advection of the amount of the water. (see Fig. 4.4)

If the cube in Fig. 4.4 changed its vertical position during the animation we would simulate a water cube that has been thrown, actually it is exactly what we need. But we can not simply accelerate the columns themselves because not the spatially fixed columns change their positions, but the water traveling in the columns. This means we have to keep track of vertical velocities of the water, their accelerations. This means, that the vertical velocity has to be advected with the same velocity field, as well. Basically, we need to advect the whole state of the water through the grid of columns, in order to know how much water, and where exactly has to be at a certain time. This includes the water's vertical position, the vertical velocity and the water amount. In contrast to the bottom layer where the absolute height of the column was calculated, here the amount of the water is advected.

Until now we have only considered a fixed velocity field. This meant that the water was transported in one direction in the whole simulation. In order to allow changing velocities, we could include next to the vertical speed the two horizontal speed components to the state of the upper layer water. Then two more advection cycles are needed for the x and y axis speeds. What actually will happen, is that at the end of every simulation cycle the velocity field itself it is advected, and in the next step the new advected velocity will be used for the advections.

The final algorithm for one timestep looks as follows:

1. Advect vertical speed and update it based on gravity - uZ
2. Advect bottom and update it based on vertical speed - b
3. Advect water amount - d
4. Advect x direction speed - uX
5. Advect y direction speed - uY

We have seen, that the advection operation is applied on five different measures in one simulation cycle. Since all of these six measures are important for the result, the accuracy and stability of the advection is very important. On the other hand the speed of the advection algorithm has a significant effect on the overall speed of the system, therefore it is important to use a fast numerical scheme. As a conclusion, the numerical advection scheme is the crucial part of the upper layer simulation, which can have a great impact on the accuracy and on the speed of the model.

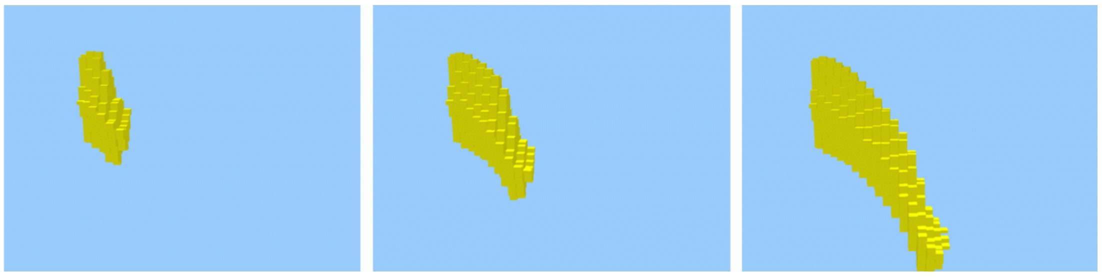


Figure 4.5: The result of upper layer movement. Flowing water from a source

5 Layer Merging

So far we described how to simulate both bottom and upper layer. Now, we propose a way to connect them as soon as the upper layer touches the bottom layer

We have different state variables for the two layers (Figure 3.3, “Height field model to represent the surface.,” on page 12):

- the bottom layer stores the qX and qY flow values
- the upper layer stores uX and uY velocity values

First, we need two conversion methods between these values, and second we have to decide how we calculate the new state values when we take water from one layer to the other one.

5.1 Bottom to Upper Layer Conversion

What is the speed of a column which has been taken from a bottom layer? We know the flows of the four connecting pipes. In order to really capture the movement of the water we calculate the average of the two same directional pipes of the column:

$$\overline{qX} = \frac{qX_{left} + qX_{right}}{2} \tag{5.1}$$

$$\overline{qY} = \frac{qY_{up} + qY_{down}}{2} \tag{5.2}$$

Now these average flows characterize the flow of the cell. Note that, in our abstraction we use one pipe connecting the two columns and these pipes have always the same direction. In case a backflow is happening in the pipe we will assign a negative flow value.

The next question would be how we transform a flow into a velocity value. The flow tells us how much water volume is transferred in a unity of time through of a unity of surface. If we want to calculate how much water is taken into the next column in a unity of time, we divide the volume by the area of the interface between the two columns. This way we compute how fast the water flows through a unity of area. The expression will be:

$$uX = \frac{\overline{qX}}{\Delta x \cdot d} \tag{5.3}$$

$$uY = \frac{\overline{qY}}{\Delta y \cdot d} \tag{5.4}$$

Having these two expressions one can calculate the velocity of the upper layer which corresponds to the flow in the bottom layer.

5.2 Upper to Bottom Layer Conversion

The other problem is how to compute the flow between the columns if we know the velocity and the amount of water. Knowing the velocities we have only one value for a direction, and we want to calculate two. That means for the horizontal direction we know *uX* and we want to calculate qX_{left} and qY_{right}.

This is exactly the amount of water volume transported in a unity of time. The volume can be calculated by multiplying the length with the area. Therefore we will get the following expressions:

$$qX = uX \cdot \Delta x \cdot d \tag{5.5}$$

$$qY = uY \cdot \Delta y \cdot d \tag{5.6}$$

In order to make the column travel nicely we have to set both pipes in one direction to have the same flow. That would yield the movement of a shape formed by the columns. This means that:

$$qX_{left} = qX_{right} = qX \tag{5.7}$$

$$qY_{up} = qY_{down} = qY \tag{5.8}$$

5.3 Merging

Now that we now the flow of a certain upper layer column, we just have to see how to combine the existing bottom layer water column with the new one.

The amount of water is straightforward, we just simply have to add the amount of water from the upper layer to the water in the bottom layer. But what do we do with the flows? We have two sets of flows for one single column. In the beginning of the chapter where the height field model was developed, we assumed that the horizontal velocity of water particles does not change a lot along a vertical column. Now we have exactly a contradictory situation, since it is probable that the two flows do not have the same value. In this sense we are at the limits of the model.

Of course we would like to get an “average” value of the two flows. However a small amount of water can not influence the behavior of a great column significantly, and this is why a depth-proportional interpolation seems a heuristically good solution for our problem.

Therefore we have the following expression to calculate the new flow of the merged column:

$$qX = \frac{d_{bottom} \cdot qX_{bottom} + d_{upper} \cdot qX_{upper}}{d_{bottom} + d_{upper}} \tag{5.9}$$

$$qY = \frac{d_{bottom} \cdot qY_{bottom} + d_{upper} \cdot qY_{upper}}{d_{bottom} + d_{upper}} \tag{5.10}$$

If the upper flows are substituted with the expressions from the section below then we have no more unknown values to calculate the new flow, and we have a model to merge the two layers.

As a remark, one should point out, that only this merged column's flow is changed. What this means is, that once this water gets in a new column there the flow will be only governed by the pressure based model. This means there is not a real and theoretically completely correct state conversion. The height field model is perfect for simulate changes of height of a water surface, but originally was not developed for simulation of flows in certain directions.

This effect can be observed as well, when as boundary condition a nonzero pipe is enforced at the cornet of the simulation domain. This is actually a way to say that here this water flowing in to our simulation. Now, no matter which pipe, the x, or the y direction is non-zero, the simulation will look exactly the same, since this value only is important for this only one column, and the other pipe's flows will be determined on the pressure difference. The simulation will basically transport unbiased this amount of water in all directions the same way, although theoretically we injected it from a certain direction.

Despite this "deficiency" of the model, in practice we have experienced realistical water behavior in our simulations (Fig. 5.1). The reason can be, that the horizontal speed transport does not have a very essential role in these simulations.

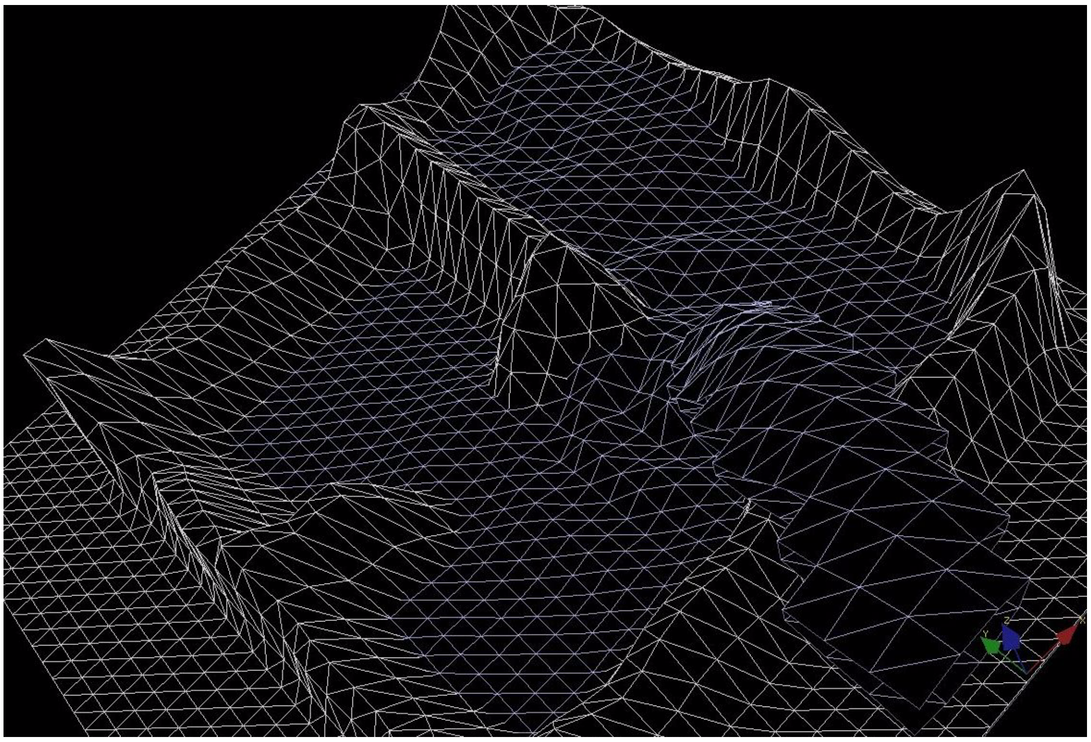


Figure 5.1: The direction of the upper water correctly influences the bottom layer. The right part of the area has is filled much faster as the left one. One can notice the difference at the middle wall.

Once a splitting method is found which imitates the behavior of the wave realistically enough we have all the mechanisms to generate breaking waves. Due to time limitations of my work I could not develop this part of the model, but I think that using the informations presented in Chapter 2, there might be a way to do it.

6

Results and Conclusions

This chapter summarizes the results of this work, emphasizing the advantages and the weaknesses of the method. On the other hand, I try to give ideas for further developments and enhancements.

6.1 Simulations

The most important question would be: What can we simulate with the method? Having two layers gives us the limitation on the phenomena we can simulate.

We can simulate movements of water on arbitrary underground including refracted water waves on the underground. Water is slowed down as it approaches the shore, thus waves will be really aligned to the shore shape.

Using the second layer as well, we can generate simulations of pouring of the water. We can freely choose the shape the quantity and the position of the water flowing in, just like the direction of it. Although the upwind implementation works only for one sign of a velocity, I think, there is no real problem to extend the implementation so that one can handle in one algorithm all the directions. Of course all the parameters of the animation can be changed during the simulation, thus we can interact with the system. For example the position of the water source can be moved around, or even the direction of the in flowing water can be changed. One can even change all these parameters during the simulation. In this case again we have arbitrary underground shape.

6.2 Results

Since the goal of my thesis was to develop the right physical model, my implementation's primary goal was to prove the theoretical results. Therefore, no special accelerating methods, like GPU vertex shaders, were applied to really use to the maximum the computational resources. Even like this our implementations are quite fast.

The implementation using Novodex SDK [17] uses a very simple straightforward rendering of the water columns using boxes. This was very useful for the debugging and development process. A second implementation in the Open Inventor Coin3d [4] framework was a fast solution for doing nicer rendering with minimal programming overhead. I am sure, that once the model is better crystallized, one can make a much more preferment implementation like these.

The Novodex SDK implementation (Fig. 6.2) could run without problem with gridsize of 70x70, allowing real time user interaction on a 1.7 GHz Pentium 4 with 256 Mb RAM. The Coin3D implementation (Fig. 6.1 and Fig. 6.3) it is slightly slower. On the same computer the gridsize of 40x40 simulation run with a framerate of 40 frames/second. On a Pentium 3, with 196 Mb RAM the same simulation runs with 26 frames/second.

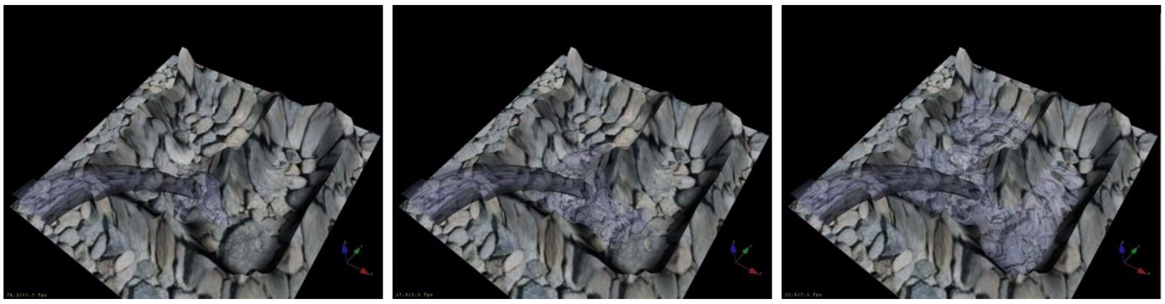


Figure 6.1: Coin3d implementation of pouring water simulation. Schreenshots.

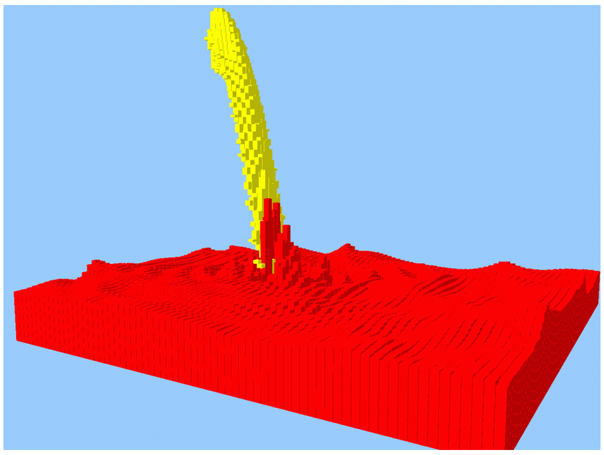


Figure 6.2: Novodex SDK implementation of pouring water. The two colors show the different layers of the simulation.

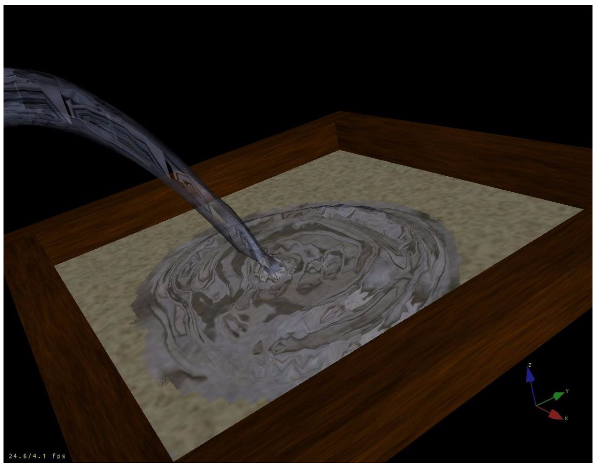


Figure 6.3: Water spreads out as it hits the bottom surface. 40x40 simulation grid

6.3 Conclusions

The model presented here allows fast, interactive rate simulations than can do more than a simple height field. If a splitting algorithm was found, the model would be ready to simulate real time plunging breaking waves. This algorithm could find applications in areas like computer games.

The model itself seems to force over the upper layer the Eulerian method, although the more natural solution might particle based. Using a grid we are hoping still on a reasonable running time, and try to keep a homogeneous model. On the other hand, the number of particles to generate a breaking wave (not the pouring simulation) could grow very fast, and in that case the presented method would probably be faster.

As a future work, the missing chain should be found, the splitting of the columns. Once having that the first wave breaking simulations could be tried. I would like to emphasize again, that the advection scheme of the upper layer is extremely important, it is possible that more work on that schemes would pay off for the overall performance of the model.

As a conclusion, this work proves that there are possibilities in real-time grid-based liquid model to simulate more than a simple height field. The results are convincing even in case of relatively coarse discretization resolutions.

A

Rendering and Implementation

A.1 Rendering a Surface

The bottom layer it is quite straightforward to render as a surface. We have the convenience that usually a bottom surface has to be rendered as well. In this sense, we can always render a complete surface of the simulation domain as water surface using the heigh values. Where there is no water the height value is anyway smaller or equal to the bottom value, thus if right ordered the bottom surface will cover the "unreal" water surface. If we use the same triangle discretization for the bottom and upper layer a higher level graphics library like Coin3d [4] can generate quite realistic surfaces even at the boundary of the water-bottom surface.

The upper layer is not that trivial to render. We have to wrap the upper layer columns with triangles. We did it in three steps. First, the lower values, actually the column positions triangles are generated, but only there where we have water. Next the upper layer's top triangles are similarly generated, but not just the column position, but the position plus the amount of water describes the surface. In the last step we have to wrap the columns from aside as well. For this we have to detect those columns which have neighbors that should not be rendered (they do not contain water), and on those faces we need to generate vertically positioned triangles.

But what vertices do we use for the triangles. In order to simplyfy the decisions in the upper layer I have chosen to interpolate the vertices from the four neighboring column positions. More exaclty only those neighbouring columns influence the vertex position which contain water. Like this every column has four adjacent vertices, and whenever we need to render it, we will generate two triangles using these vertices. As an enhancement we can pull up lower upper vertices defined by less then four columns, respectively push down top upper vertices. This we can do with a simple interpolation between the original top a lower vertices, to update their values. This will yield a smoother surface on the side of the water in the upper layer.

In our implementation the layers are completely independently, this means, that the merging point is not really a single surface, one can see discontinouties. It could be another enhancement to find a fast way to generate a guaranteed continous surface there, as well. But a very complicated rendering is not really possible because of the real-time constraint, but as the picture shows even like this one can achieve nice renderings.

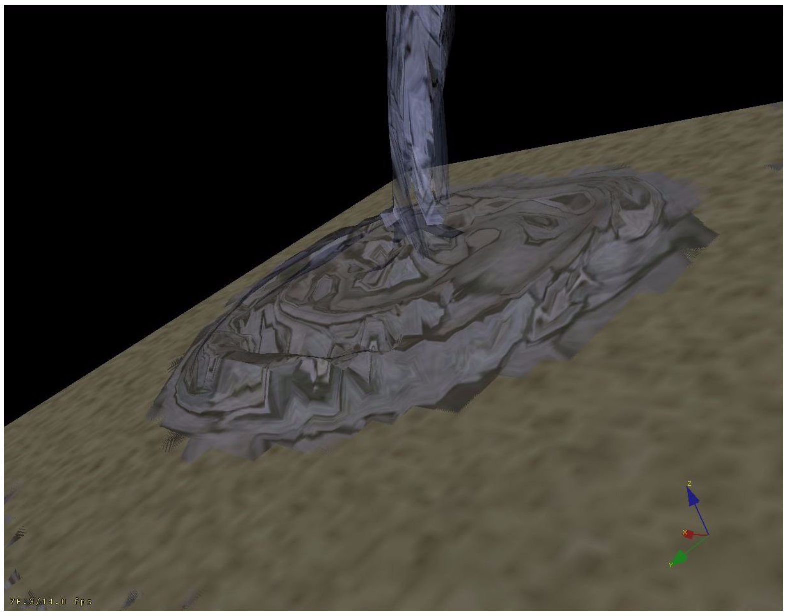


Abbildung A.1: This frame is rendered using only a 40x40 grid for the whole square simulation domain

A.2 Implementation

In the Novodex SDK [17] a LayeredWater class has been created that implements the Animatable interface.

In Coin3d [4] environment a specialized SoShapeKit class animated the water surface. Another SoShapeKit class represented the ground of the simulation. This was read from a 257x257 raw file, and resampled to the actual resolution we needed. This way it is very easy to generate grounds for the simulations, since the Terragen [21] program can export the surface heights in this format. The surfaces are generated as SoIndexedTriangleStripSet, which enables correct normal calculations for the surface. Using the indexed version we can minimize the memory needs, and as experience even with high resolutions we did not encounter memory problems. Lastly, if we build up the correct scene graph, with texture nodes we will get real-time renderings like on the figure above. To insure more realism the water surface in environment mapped, a light blue diffuse color is used and 0.5 transparency factor assigned.

B

References

[1] J.A. Battjes. “Surf similarity”. In *Proceedings of 14th Coastal Engineering Conference, Copenhagen, Denmark, American Society of Civil Engineers*, New York, pages 466-480, 1974

[2] R. M. Beam and R. F. Warming. “An Implicit Factored Scheme for the Compressible Navier-Stokes Equations,” In *AIAA Journal, Vol. 16, No. 4, 1978*, pages 393-402, 1987

[3] M. Carlson, P. J. Mucha and Greg Turk. “Rigid fluid: animating the interplay between rigid bodies and fluid”. In *ACM Transactions on Graphics Volume 23, Issue 3 (Proceedings of the 2004 SIGGRAPH Conference)*, pages: 377 - 384, 2004

[4] Coin3d 2.3.0. www.coin3d.org

[5] G. Crapper. “Introduction to Water Waves.” John Wiley & Sons, New York, 1984

[6] D. P. Enright, S. R. Marschner, and R. P. Fedkiw. “Animation and rendering of complex water surfaces.” In *SIGGRAPH 2002 Conference Proceedings*, pages 736–744, 2002.

[7] A. Fournier and W. T. Reeves. “A simple model of ocean waves.” In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, pages 75–84, 1986.

[8] T. S. Hedges. “Wave Breaking and Reflection”. http://www.liv.ac.uk/~ec22/topics/Wave%20Breaking%20and%20Reflection.pdf

[9] M. A. Hiltzik and A Pham. “Synthetic actors guild.” *Los Angeles Times*. May 8, 2001

[10] S. Jeschke, H Birkholz and Heidrun Schmann. “A Procedural Model for Interactive Animation of Breaking Ocean Waves”. In *The 11-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2003.*

[11] M. Kass and G. Miller. “Rapid, stable fluid dynamics for computer graphics” In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, pages 49–57, 1990

[12] B.P. Leonard. “A stable and accurate convective modelling procedure based on quadratic upstream interpolation.” In *Computer Methods in Applied Mechanics and Engineering*, 19:59–98, 1979.

[13] R. Miche. “Mouvement ondulatoires de la mer en profondeur constante ou decroissante”, In *Annales des Ponts et Chaussees*, pages 130, 1944

[14] V. Mihalef, D. Metaxas and M. Sussman. “Animation and Control of Breaking Waves”. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* 2004.

[15] J. Y. Murthy. “Numerical Methods in Heat, Mass, and Momentum Transfer. Draft Notes”, http://widget.ecn.purdue.edu/~jmurthy/me608/main.pdf, 2002

[16] M. Müller, D. Charypar, M. Gross. “Particle-Based Fluid Simulation for Interactive Applications”. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation (SCA) 2003*, pages 154-159.

[17] Novodex SDK. www.novodex.ch

[18] J. O’Brien and J. Hodgins. “Dynamic simulation of splashing fluids.” In *Computer Animation 95*, pages 198–205, 1995

[19] D. R. Peachey. “Modeling waves and surf.” In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, pages 65–74, 1986

[20] J. Schneider and R. Westermann. “Towards real-time visual simulation of water surfaces.” In *Proceedings of the Vision, Modeling, and Visualization Conference 2001 (VMV-01)*, pages 211–218, 2001.

[21] Terragen - Photorealistic Scenery Rendering Software. http://www.planetside.co.uk/terragen/

[22] J. Tessendorf. “Simulating Ocean Water.” In *Simulating Nature: Realistic and Interactive techniques*. SIGGRAPH Course Notes 47, 2001.

[23] R.A. Trompert and U. Hansen. “The application of a finite volume multigrid method to three dimensional flow problems in a highly viscous fluid with a variable viscosity, Geophys.” In *Astrophys. Fluid Dyn. 83 (1996)*, 261-291, 1996

[24] P.Y. Ts’o and B. A. Barsky. “Modeling and rendering waves: wave tracing using beta-splines and reflective and refractive texture mapping.” In *ACM Transactions on Graphics 6, 3 (1987)*, pages 191–214, 1987

[25] Y. Wang and K. Hutter. “Comparisons of numerical methods with respect to convectively dominated problems.” In *International Journal for Numerical Methods in Fluids, Int. J. Numer. Meth. Fluids 2001*; 37:721–745, 1997