Visualization of Large-Scale Fluid

Thomas Krog

LYNGBY 2003
MASTERS THESIS
IMM-2003-73

IMM

Printed by IMM, DTU

Abstract

In this project a new approximation of the Shallow Water equations is used to simulate large-scale fluids in real-time. The simulation has been tested against a Smoothed Particle Hydrodynamics model with reasonable results. However, artifacts occur when it is tested with bumps on the ground thus improvements are needed before the approximation can be used as a general purpose solver of large-scale fluids.

The visualization of the fluid has significant influence on the way a user experience the fluid. By extending the visualization with flying foam the average user believes that it is large scale fluid in contrast to a visualization without foam. The foam has been modelled by transparent billboards and lines thus it requires a lot of blending to get a smooth transition between the opaque fluid surface and the transparent foam. Therefore an effective blending system has been implemented at the boundary of the fluid to save the pixelshader from a lot of work.

Keywords: Visualization, Real-Time, Fluid Simulation, Shallow Water Equations, Height maps, Foam Visualization

Preface

A number of people have been helpful in the development of this project. Thanks to my two supervisors Andreas Bærentzen and Niels Jørgen Christensen. They have given me fairly free hands in the project, however, they have been ready to offer advice when it was needed. I also want to thank Per Slotsbo for many good discussions and brainstormings leading to valuable solutions in the project. Furthermore, Marinus Rørbæk, Anders Hartung Christensen and my parents have provided important corrections to the report. Finally, I want to thank the people who have evaluated the realism of the fluid visualization.

Contents

Preface **2**

1 Introduction **9**
1.1 Objective 11
1.2 The Readers Prerequisites 11
1.3 The Structure of the Report 11

I Fluid Dynamics **12**

2 Fluid Dynamics Models **13**
2.1 Grid Representation 13
2.2 Smoothed Particle Hydrodynamics 14
2.3 Navier-Stokes Equations 15
2.3.1 Conservation of Volume 15
2.3.2 Conservation of Momentum 15
2.3.3 Applications 17
2.4 Height Map 17
2.4.1 Volume Interpretation 17
2.4.2 Limitations 18
2.5 Navier-Stokes Equations used on Height Maps 19
2.5.1 The Boundary Algorithm 19
2.5.2 The Height Map Algorithm 21
2.5.3 Discussion 21
2.6 Volume Conservation is easier in Height Maps 23
2.7 Shallow Water Equations 23
2.7.1 A Practical Explanation of the SWE 24
2.7.2 Applications 27

2.7.3 The Conservative Equations . . . 27
2.8 Conclusion . . . 28

3 Non Conservative SWE **29**
3.1 The Velocity Representation . . . 29
3.2 Velocity Advection Problems . . . 31
3.2.1 The Term is Too Local . . . 31
3.2.2 The Depths should have Influence . . . 32
3.2.3 The Term is too Global . . . 33
3.2.4 Local Term with Weighted Depths . . . 33
3.3 A Model without VEAT . . . 36

4 Conservative Shallow Water **38**
4.1 The Imperative VEAT . . . 38
4.1.1 Discharge Conservation . . . 42
4.2 Stability . . . 42
4.2.1 Discussion . . . 42

5 The Boundary of the Fluid **43**
5.1 Fluid Floating Down a Slope . . . 43
5.1.1 The Fluid will Never Leave a Cell . . . 43
5.1.2 The Boundary Velocity Grows towards Infinity . . . 45
5.2 Determining the Cells Near the Boundary . . . 45
5.3 Invisible Fluid . . . 46
5.4 Removing Fluid . . . 46
5.5 Modification of the Volume Advection . . . 47

6 Results **49**
6.1 The Breaking Dam Experiment . . . 49
6.2 A Bump Experiment . . . 49
6.3 Conclusion . . . 51

II Visualization **54**

7 Tessellating the Fluid Surface **55**
7.1 Closing the Gap at the Boundary . . . 55
7.2 Smoothing the Boundary in Time . . . 57
7.2.1 A Boundary based on the Fluid Depth . . . 57

7.2.2 A Boundary based on the Future . . . 58
7.3 Smoothing the Boundary in Space . . . 59
7.3.1 Obtaining a Horizontal Fluid Surface . . . 59
7.3.2 Smoothing in the xy-plane . . . 60
7.4 Blurring the Boundary . . . 60
7.4.1 Internal Boundary Blurring . . . 60
7.4.2 External Boundary Blurring . . . 61

8 Flying Particles **63**
8.1 Motivation . . . 63
8.1.1 Controlling the Viscosity . . . 63
8.2 Survey of Models . . . 64
8.2.1 Particles with mass . . . 64
8.2.2 Massless Particles . . . 64
8.3 Particle Spawning . . . 65
8.3.1 Waterfalls . . . 65
8.3.2 Accumulation . . . 68
8.4 Moving the Particles . . . 69
8.4.1 Performance . . . 70
8.4.2 Random Generators . . . 71
8.4.3 Bouncing on the Ground . . . 71
8.4.4 Setting up the Network . . . 73
8.5 Visualization . . . 73
8.5.1 Primitives . . . 73
8.5.2 Discussion of the Primitives . . . 74
8.5.3 Avoiding Intersections with the Ground . . . 75
8.5.4 Blurring the Boundary . . . 77

9 Foam on the Fluid Surface **79**
9.1 Survey of Methods . . . 79
9.1.1 Flow Visualization . . . 79
9.1.2 Foam Visualization . . . 80
9.2 The Basic Idea . . . 80
9.3 Even Distribution . . . 82
9.3.1 Removing Surface Particles . . . 82
9.3.2 Adding Surface Particles . . . 83
9.3.3 Fading In and Out . . . 84
9.4 Calculation of the Foam . . . 84

9.5 Visualization of the Foam . . . 85
9.6 Smoothing the Texture Boundary . . . 85
9.7 Determining the Size of the Textures . . . 86

III Results **88**

10 Implementation **89**
10.1 Performance . . . 89
10.1.1 Removing the Branches . . . 89
10.1.2 Potential Bottlenecks . . . 90
10.2 Render to Texture . . . 91
10.3 Grid Searching . . . 91

11 Comparison **93**
11.1 Comparison with other Models . . . 93
11.1.1 Comparison with Kass . . . 93
11.1.2 Chens Simulation . . . 94
11.1.3 Laytons Simulation . . . 96
11.2 Comparison with real Water . . . 96
11.3 Animation . . . 96
11.4 Discussion . . . 99

12 Conclusion **100**

A Notation **101**
A.1 Symbols . . . 101
A.2 Abbreviations . . . 101
A.3 Terms . . . 102
A.4 Variables and Constants . . . 102

B Accessories **103**

List of Figures

2.1 The Relation Between the Cells and the Velocities in 2D . . . 14
2.2 An example of the height map in 1D . . . 18
2.3 Volume Interpretation . . . 19
2.4 An example of a 2D height map . . . 20
2.5 An example of a Waterfall . . . 22
2.6 Collision of fluids in a pipeline . . . 24
2.7 An explanation of the Advection . . . 25

3.1 Edge- versus center velocity . . . 30
3.2 Large Depth Difference . . . 32
3.3 Global Velocity Advection Problem in a Breaking Dam . . . 34
3.4 Local Velocity Advection Problem in a Breaking Dam . . . 35
3.5 Local Velocity Advection Problem (screenshot) . . . 35
3.6 A Waterfall in a Stationary State . . . 36

4.1 Imperative VEAT Solutions . . . 40
4.2 VEAT Solution for Negative Discharge . . . 41

5.1 Fluid Floating Down a Slope . . . 44
5.2 Boundary Determination . . . 45
5.3 A Lake Removed by the Boundary Behaviour . . . 46

6.1 The breaking dam experiment with imperative VEAT. . . . 50
6.2 A Bump experiment on a height map with imperative VEAT . 52

7.1 Internal Triangles . . . 56
7.2 Six special cases. . . . 56
7.3 Tessellating three special cases. . . . 56
7.4 Complete Tessellation . . . 57
7.5 No Time Smoothing . . . 58

7.6 Time Smoothing 58
7.7 Obtaining a horizontal fluid surface at the boundary. 59
7.8 Smoothing in the xy-plane 60
7.9 Unwanted Transparency. 61
7.10 External Boundary. 61
7.11 Three simple cases for the external boundary. 62
7.12 External boundary on a concave fluid area. 62

8.1 The Top of Various Waterfalls 66
8.2 The Velocity of Foam a Waterfalls 68
8.3 Initial foam velocity when the fluid accumulates 69
8.4 Simplifying the Flying Particle Network 70
8.5 Foam Bouncing 72
8.6 Foam Bouncing Schematic 72
8.7 Flying Particle Network for Waterfalls 73
8.8 Foam Intersection Screenshots 76
8.9 Illustration of a Foam Intersection 77
8.10 Bluring the Boundary 78

9.1 Flow Visualization by Wijk 80
9.2 Texture Copy 81
9.3 A Waterfall with Surface Particles 82
9.4 External Surface Textures 86
9.5 Two Layers of Fluid 87

10.1 Grid Searching 92

11.1 Comparison with Kass 94
11.2 Chens Simulation 95
11.3 Laytons Simulation 96
11.4 Waterfall from a movie called River Wild. 97
11.5 Imitation of the waterfall from figure 11.4. 97
11.6 Animation 98

A.1 A Breaking Wave 102

Chapter 1

Introduction

Simulation of 3D fluid is a challenging topic, and it is still an active research area. Significant progress has been made during the last decade where a new fluid representation was invented by Monaghan [20] based on smoothed particle hydrodynamics. Furthermore, improvements have been made for the traditional grid representation where Foster [9] has developed the most outstanding improvement.

Such true 3D simulations offer nice realism, since the 3D representation of the fluid makes it possible to set up realistic models of the fluid dynamics. The drawback of such models is that they are computationally expensive and hence interactive simulations in true 3D are only at a preliminary stage, e.g. IO-Interactive expects to have an interactive simulation ready at the release of Playstation 3 in 2006. Even when the interactive 3D simulations become available, the amount of details in the fluid are limited, and hence there are a number of applications where remarkably more details are needed.

In this project methods are examined to increase the amount of details in simulation of large-scale fluid in landscapes. The true 3D simulations tend to waste information along the vertical direction in landscapes, thus a tremendous amount of information can be saved by assuming that the velocity at the bottom of the fluid is equal to the velocity at the surface of the fluid. The methods using this assumption are called *single layered velocity methods* (abbreviated SLV methods). The saved information allows a more detailed representation of the riverside and the shore which makes the methods convenient for various phenomena such as rivers, lakes, seas and floodings.

The SLV methods have primarily been used to simulate climatic and tidal

phenomena without attention to interactive restrictions. Examples of such methods are found in [1] and [24] both of which use a set of equations called the Shallow Water Equations. As regards real-time methods the proportion is more limited. Kass [15], Layton [2] and Chen [4] are examples of real-time methods based on SLV. Layton use the Shallow Water Equations to simulate fluid with a stationary shore. Kass simulates a dynamic shore however a simplified version of the Shallow Water Equations is used. Chen [4] has proposed an alternative method to the Shallow Water Equations which is suitable for small-scale vortices. However, it does not provide a physically correct simulation of the volume. Therefore it is not well suited for breaking dams which involve large depth variations. Furthermore it has problems with waterfalls and rivers as discussed in section 2.5.3. Consequently the main focus will be kept on the Shallow Water Equations in this project.

The visualization also plays an important role in this project. The volume of the fluid can easily be visualized using polygons and some standard lighting from OpenGL [18]. However, a simple animation of the volume does not provide much information about the flow direction of the fluid. Since the flow information comes for free from the fluid simulation, it seems natural to extend the visualization so that this information is visible to the viewer. The flow is visualized by usage of textures which follow the direction of the flow. This is a classic problem and various solutions exist. Wijk [29] has developed a detailed visualization of the flow, however the source images of the water tend to become noisy. Other solutions like Jobard [3] have been developed which avoid the noise, however, they are more computationally expensive. In this project the performance is important whereas the details of the flow are less important. Consequently a new method is developed which satisfies these demands.

Furthermore, the clean volume of the fluid tends to be insufficient at waterfalls, thus an additional particle system is used to handle such situations. In [16] a particle system has been briefly described for oceans, whereas [21] provides a system of a small-scale.

At the bottom of the waterfalls it is expected that the foam from the waterfall will start floating in the fluid surface. This is handled effectively by extending the texture of the fluid surface.

1.1 Objective

In this project the objective is to develop an interactive visualization of large-scale fluid in landscapes. This is a comprehensive task requiring a simulation of the fluid dynamics before it is possible to do the visualization. In order to limit the size of the project the numerical solver of the simulation has been downgraded to keep the focus on the physical equations which are the foundation of the simulation. The simulation is based on SLV methods, thus it is not possible to handle breaking waves, however the methods are used to model rivers, lakes, seas and floodings.

For the visualization the primary task is to visualize the velocity of the fluid by using suitable textures. Furthermore, the scale of the fluid plays an important role, thus it is examined how effects like flying foam affect the scale of the fluid.

In this project no lighting methods are provided since it is a relatively well solved problem, see [16]. Furthermore, issues about transparent fluid are omitted.

1.2 The Readers Prerequisites

In order to limit the size of the report it is assumed that the reader has got a couple of introductory university courses in mathematics and computer graphics. Appendix A contains a summary of the notation used in the report.

1.3 The Structure of the Report

The simulation of the fluid surface tends to be a rather isolated problem from the rest of this project. Actually, most of the articles about fluid simulation in this bibliography do not develop visualization methods. Furthermore, the interface between the simulation and the visualization is quite clear. The simulation should determine the movement of the fluid surface and the velocities of the fluid and hence provide it for the visualization method. Therefore it seems natural to solve the simulation as an isolated part in this report. The last part deals with the visualization method which has free access to the position of the fluid surface and the velocities of the fluid.

Part I
Fluid Dynamics

Chapter 2

Fluid Dynamics Models

This chapter provides background knowledge of relevant fluid dynamics models from the literature. In order to simulate a fluid, a data structure is needed to represent the fluid. The next two sections describe the two most widespread representations of fluids.

2.1 Grid Representation

The grid representation is the classical way of representing fluid. It is the most spread representation of fluids and [8] and [25] are examples of projects using this representation.

Figure 2.1 shows a 2D grid where the velocities are defined *between* the cells. For instance $\mathbf{u}_{i+1/2,j}$ is the velocity from cell $c_{i,j}$ to $c_{i+1,j}$. This is called *edge velocities*. Alternatively, the velocities could be defined as *center velocities* which are velocities at the center of the cells.

Naturally, the 2D grid can be extended to a 3D grid by adding an extra index to the cells.

The term *neighbour cells* is defined such that each cell $c_{i,j}$ has four neighbour cells namely $c_{i+1,j}$, $c_{i-1,j}$, $c_{i,j+1}$ and $c_{i,j-1}$.

Note that the grid does not represent the volume of the fluid, which should be stored in an extra data structure. For example in [9] they extend the grid with marker particles which are massless particles following the velocity of the grid. In this way the particles represent the volume of the fluid.

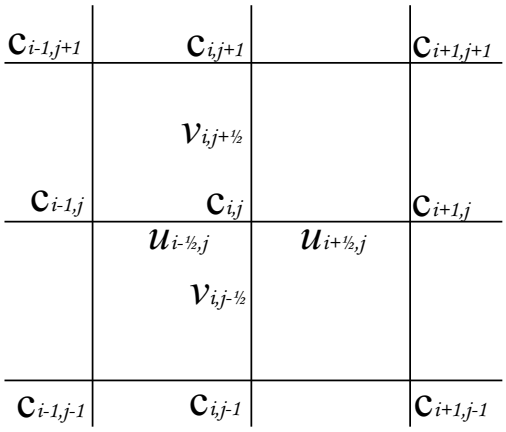

Figure 2.1: The relation between the cells and the velocities in a 2D grid with edge velocities.

2.2 Smoothed Particle Hydrodynamics

The Smoothed Particle Hydrodynamics (abbreviated SPH), is a relatively new representation of fluids, which was first presented by Monaghan in 1994 (see the article in [20])[1]. The fluid is represented as particles and each of these contains a velocity. This means that the velocity of the fluid can be determined at any location as a weighted sum of the nearest particles.

In this method it is considerably easier to add boundaries than it is for the grids. The boundaries can be represented as a continuous function $f(\mathbf{x})$ which returns the external force at location $\mathbf{x}$.

One of the drawbacks in this method is that the simulation is slower than a grid simulation. There are several reasons for this. First of all the particles are flowing freely between each other, thus it is more time-consuming to find neighbour particles. Furthermore, it is necessary to calculate the distance between each pair of neighbour particles which involves a square root.

Another drawback is that the running time is $\mathcal{O}(n^3)$ in 3D where n^{-1} is the distance between two neighbour particles. In other words if the resolution is doubled along each axis, the running time will be multiplied by eight.

This means that it is difficult to obtain real-time simulations in 3D at present time.

[1]The SPH was originally invented by Lucy [17] to simulate gases.

2.3 Navier-Stokes Equations

The motion of a fluid can be described by the Navier-Stokes Equations, which were developed by Navier and Stokes in the 1840s. In order to set up the equations the velocity of the fluid is described as a vector field $\mathbf{w}$. Now the equations state the evolution of $\mathbf{w}$ in time, when there are two restrictions namely conservations of volume and momentum. Each of the two restrictions result in an equation which is described briefly in the next two sections. A detailed derivation of the equations exists in [23].

2.3.1 Conservation of Volume

The conservation of volume requires that the amount of fluid flowing into an arbitrary volume of the fluid must be the same as the amount of fluid floating out of the volume. Formally this requirement can be written as

$$\nabla \cdot \mathbf{w} = 0. \tag{2.1}$$

2.3.2 Conservation of Momentum

The conservation of momentum is fulfilled iff. (if and only if) each small mass $\mathrm{d}m$ of the fluid obeys Newton's second law

$$\mathrm{d}\mathbf{F}_{net} = \mathrm{d}m\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t},$$

where $\mathrm{d}\mathbf{F}_{net}$ is the net force affecting the fluid. The force $\mathrm{d}\mathbf{F}_{net}$ is composed by several contributions which are described in the following:

Pressure When there is a difference in pressure between two locations, it will affect the fluid with a force pointing towards the location with low pressure. The force becomes

$$\mathrm{d}\mathbf{F}_p \;=\; -\frac{\mathrm{d}m}{\rho}\,\nabla p,$$

where p is a scalar field of the pressure while ρ is the density.

Drag When there is a difference in velocity between two locations, there will be a force which will smoothen the velocity difference. The force is determined by

$$\mathrm{d}\mathbf{F}_d \;=\; \frac{\mathrm{d}m}{Re}\nabla^2\mathbf{w}$$

where Re is a constant called the *Reynolds number* [4]. The constant is inversely proportional to the viscosity of the fluid. The force $\mathrm{d}\mathbf{F}_d$ is the only friction force affecting the fluid, thus a high value of Re gives a turbulent flow while a low value of Re gives a laminar flow.

External External force $\mathrm{d}\mathbf{F}_e$ which affect the fluid. The only external force in this project is the gravity $\mathbf{g}$.

This completes the list of forces, and hence the net force becomes

$$\mathrm{d}\mathbf{F}_{net} = \mathrm{d}\mathbf{F}_p + \mathrm{d}\mathbf{F}_d + \mathrm{d}\mathbf{F}_e.$$

By inserting this equation into Newton's second law we get

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = \frac{1}{Re}\nabla^2\mathbf{w} - \frac{1}{\rho}\,\nabla p + \mathbf{g} \tag{2.2}$$

The equations (2.1) and (2.2) together form the Navier-Stokes Equations for a particle in a fluid.

However, in this project we use grids to simulate the fluids. Consequently, the velocities are at fixed locations, thus we need an additional term to convert from a particle velocity $\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t}$ to a grid velocity $\frac{\partial\mathbf{w}}{\partial t}$ as described in [26]. The convertion from particle velocity to grid velocity can be purely mathematically derived as

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = \frac{\partial\mathbf{w}}{\partial t} + (\mathbf{w}\cdot\nabla)\mathbf{w}.$$

This term is called the *velocity advection term* (abbreviated VEAT) since it transfers the velocity from one grid cell to another. The term will be explained more thoroughly in section 2.7.1. Note that this is the term which makes Navier Stokes equations nonlinear. The term can be inserted into (2.2) and we get

$$\frac{\partial\mathbf{w}}{\partial t} = \frac{1}{Re}\nabla^2\mathbf{w} - \frac{1}{\rho}\,\nabla p + \mathbf{g} - (\mathbf{w}\cdot\nabla)\mathbf{w} \tag{2.3}$$

which together with (2.1) form the Navier-Stokes equations for fluid in a grid.

2.3.3 Applications

There are a large number of simulation models based on the Navier-Stokes equations. The models described in [9] and [8] are examples of simulations which produce 3D dynamics of fluid with high realism.

In both articles the Navier-Stokes equations are approximated using a finite-difference solution. The method converts the partial derivatives to difference operators as described in [14] e.g. $\frac{\partial u}{\partial x}$ at cell $c_{i,j}$ can be approximated as

$$\frac{u_{i+1/2,j} - u_{i-1/2,j}}{\Delta x}.$$

The drawback of such models is that they have got a running time of $\mathcal{O}(n^3)$ where n is the number of subintervals along one axis. Thus similarly to SPH simulations it is difficult to obtain real-time simulations.

2.4 Height Map

In order to avoid the $\mathcal{O}(n^3)$ running time the SLV methods have been introduced to use a horizontal 2D grid to simulate a 3D volume of fluid with Navier Stokes equations. Each cell $c_{i,j}$ in the grid is extended with a ground height $b_{i,j}$ and the height of the fluid surface $h_{i,j}$. In this way the depth of the fluid can be determined as $d_{i,j} = h_{i,j} - b_{i,j}$. This extended 2D grid is called a *height map* since it represents a volume of the fluid. Figure 2.2 shows an example of a 1D height map which represents the shape of a ground covered by some fluid[2].

A *boundary cell* is defined as a cell $c_{i,j}$ for which $0 < d_{i,j}$ while at least one of its neighbours does not contain any fluid.

The notation $\tilde{d}_{i+1/2}$ indicates that the value does not exist in the height thus the value is to be approximated e.g. as $\frac{d_i + d_{i+1}}{2}$. The notation may be used on any value in the height map.

2.4.1 Volume Interpretation

Sometimes it is convenient to think of the cells as 3D boxes. Figure 2.3 shows how a 1D height map can be interpreted to represent a volume of fluid in

[2] b_i, h_i and d_i contain only one index since it is a 1D grid on figure 2.2.

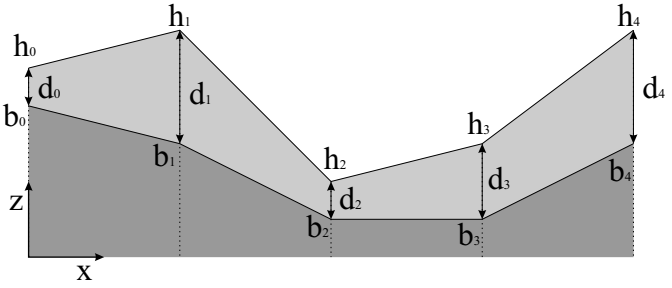

Figure 2.2: An example of the height map in 1D. The dark shaded shape is the ground whereas the light shaded shape is the fluid. b_i and h_i are the z-coordinates of the vertices on the surface of the ground and the fluid respectively. d_i is the depth of the fluid.

3D. Based on this interpretation the volume of cell c_i becomes:

$$d_i \Delta x \Delta y.$$

2.4.2 Limitations

Unfortunately, the compactness of the height map does not come for free. Here, is a couple of limitations which should be kept in mind:

- The distance between neighbour vertices on the surface of the ground is dependent on the slope of the ground. This can be realized by considering the distance between the vertices at $(1, b_1)$ and $(2, b_2)$ on figure 2.2. Consequently, the slope $\frac{|b_1 - b_2|}{\Delta x}$ grows when the distance between the vertices grows and hence the resolution decreases. Less formally the resolution is too low at the areas with large slope whereas the resolution is too high at the areas where the ground is horizontal. In many applications the waterfalls play an important role, thus it is a considerable problem that they have got the lowest resolution due to the large slope. The problem could be solved by a finite volume method as described in [5], however in this project we will keep the simple height map.

- It is not possible to simulate caves or bridges since each cell in the height map only consists of a single height $b_{i,j}$ for the ground.

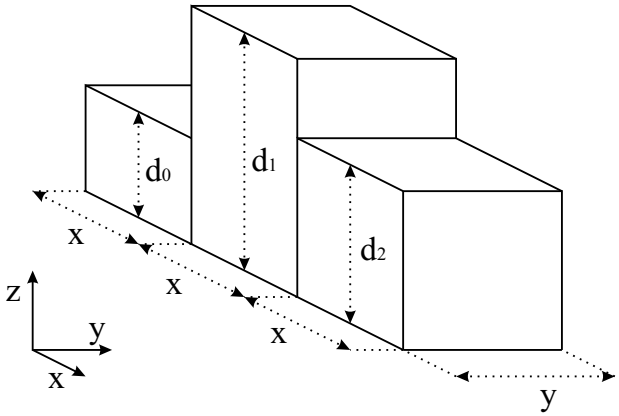

Figure 2.3: The figure shows a volume interpretation of a 1D height map. For the sake of simplicity it is assumed that $b_0 = b_1 = b_2$.

- No breaking waves appear since the data structure only allows a single layer of fluid. The data structure could be extended to support multible layers of fluid, but it would complicate the calculations considerably. An alternative solution might be to launch particles.

2.5 Navier-Stokes Equations used on Height Maps

Chen [4] uses a 2D height map to simulate a volume of fluid in $\mathcal{O}(n^2)$. Here is a brief algorithm based on Chen's method which is composed of a boundary algorithm and an algorithm to determine a height map[3]. The purpose of the boundary algorithm is to ensure volume conservation while the height map algorithm approximates details such as ripples.:

2.5.1 The Boundary Algorithm

The idea behind this algorithm is to determine the movement of the boundary only by examination of the cells at the boundary:

- **Initialization**

[3]In this description some of the data structures have been left out since it is only the realism of the model which is of interest.

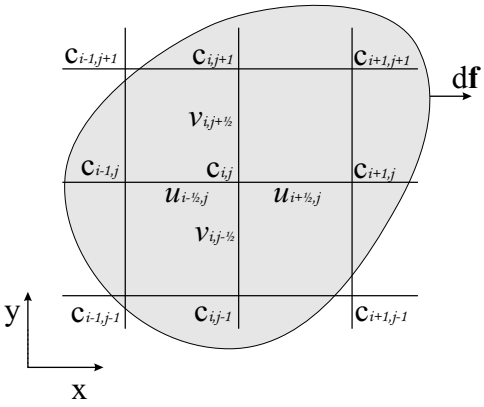

Figure 2.4: An example of the height map in 2D. The shaded shape is the area covered by fluid. The normalized vector $d\mathbf{f}$ is orthogonal to the boundary of the fluid.

First the model should be properly initialized by marking the cells which are covered by the fluid. Figure 2.4 shows an example of a height map where $c_{i-1,j-1}$, $c_{i,j-1}$, $c_{i+1,j}$, $c_{i+1,j+1}$, $c_{i,j+1}$ and $c_{i-1,j}$ are the boundary cells. The cell $c_{i,j}$ is not a boundary cell since all its neighbours are covered by fluid. Furthermore, a non negative flow rate should be selected for each of the boundary cells. The fluid will contain a source with a flow rate corresponding to the sum of all the boundary flow rates. The sum of the boundary flow rates will remain constant during the simulation. $h_{boundary}$ is defined as the surface height of the lowest boundary cell.

- **For Each Time Step**
  Examine every boundary cell $c_{i,j}$ in the following way:

  if $c_{i,j}$ has neighbour cells outside the fluid and below $h_{boundary}$ **then**
  $c_{i,j}$ should expand the boundary to these neighbours. The flow rate of $c_{i,j}$ is divided equally to the new boundary cells.

- **Increase the boundary height**

  if no expansion of the boundary has occurred **then**
  $h_{boundary}$ should be increased correspondingly to the sum of all the boundary flow rates.

2.5.2 The Height Map Algorithm

For each time step do the following:

1. **Boundary Conditions.** The velocities **u** and **v** which intersect the boundary of the fluid should be assigned a scaled value of the flow rate at the particular boundary cell.

2. **Solve Navier-Stokes equations.** Use a finite-difference solution of the Navier-Stokes equations to determine the velocities **u** and **v** on the height map. For each cell the pressure is determined as

$$p = \epsilon^{-1}\left(\frac{\partial \mathbf{u}}{\partial x} + \frac{\partial \mathbf{v}}{\partial y}\right), \tag{2.4}$$

   where $0 < \epsilon$ is a very small number introduced. For $\epsilon \to 0$ equation (2.4) satisfies equation (2.1).

3. **Calculate the depths.** For each cell the depth

$$d = h_{boundary} + a_1 p, \tag{2.5}$$

   is calculated from the pressure and a chosen constant a_1 which determines the height of the waves.

2.5.3 Discussion

There are several advantages with Chen's method since it solves Navier-Stokes equations in real-time. This leads to a relatively high realism in the horizontal plane with vortices and the ability to simulate fluid with various viscosities.

However, the method also contains a number of problems. Most of the problems are rather easy to solve, however, in this discussion one serious problem has been discovered which is not discussed in Chen's work. The problem, occurs when Chen's method is used to simulate a waterfall as shown on figure 2.5. When the fluid passes the dot-and-dash line, the waterfall starts. This means that the right most boundary cell will allways have exactly one neighbour cell which is below $h_{boundary}$, and hence the boundary will expand to the right for each time step. Consequently there is no expansion along the y-axis and therefore the width of the waterfall along the y-axis

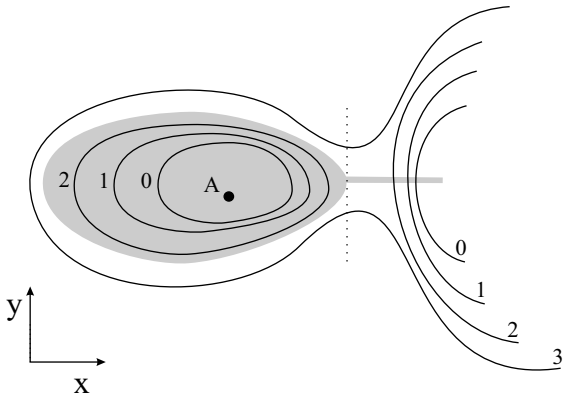

Figure 2.5: The figure shows an example of a landscape visualized by iso-curves. The numbers indicate the height of the corresponding curve (the z-coordinate of the curve). The bullet A is a source, and the shaded shape indicates the area covered by fluid. When the fluid passes the dot-and-dash line, a waterfall starts.

becomes zero independent of the flow rate[4]. This is generally a problem for such algorithms which only examine the boundary cells.

Furthermore, it is a problem that the boundary expands for every time step during the waterfall. This implies that the velocity of the boundary is inversely proportional to Δt. In other words for $\Delta t \to 0$ the boundary velocity will go towards infinity. Unlike the waterfall width problem this problem has an easy solution, namely to add an extra variable to every boundary cell. This variable should keep track of the amount of time until the cell should expand.

The final problem in this discussion is that the depth d in the height map may become negative since there is no lower bound of the pressure. The problem may be solved by substituting 2.5 with

$$d = f\left(h_{boundary} + a_1 p\right),$$

where f is a suitable monotonous increasing function which is non negative.

In the next section we will examine the Shallow Water equations which do not have any of the problems in Chen's method. However, it is at the cost

[4]The same problem will occur in rivers where the slope never become zero.

of an advantage in Chen's method, namely the ability to set the viscosity of the fluid.

2.6 Volume Conservation is easier in Height Maps

Volume conservation is one of the fundamental properties of fluids described in section 2.3.1. It turns out that the property is quite difficult to obtain in a general fluid simulation which should handle a situation as shown on figure 2.6. The figure shows two fluids which are moving towards each other with constant velocity, i.e. there are no external forces like gravity. When the two fluids collide at the dot-and-dash line, the velocity of the entire fluid must change immediately in order to obtain incompressible fluid [5]. For example the fluid will shrink if the velocity of the rear of fluid A is constant during the collision.

In a general fluid simulation like sph. or a grid the incompressibility can either be obtained by a small time step or by specific algorithms solving a large set of equations. Both solutions are computationally expensive since you have to transfer information from the front to the rear of the fluid in very short time. This is not suitable in a real time simulation.

Luckily the situation shown on figure 2.6 does not occur in a height map. That is, in a height map obstacle W_2 does not exist and hence the fluid can move upwards. This means that the velocity of the rear of fluid A can be constant during the collision, while the fluid is incompressible.

2.7 Shallow Water Equations

The Navier-Stokes equations are quite computationally expensive to solve, thus by assuming that the water is shallow, the equations can be simplified. Formally, *shallow water* means that the wave length L is much larger than the depth d of the fluid. In [26] the fluid is called shallow if $20d < L$. Two important properties can be derived from the shallow water assumption, namely that the phase velocity (the velocity of a front) becomes $\sqrt{gh}$, and

[5] Real fluid is slightly compressible, but it is beyond the scope of this analysis since the difference is not visible.

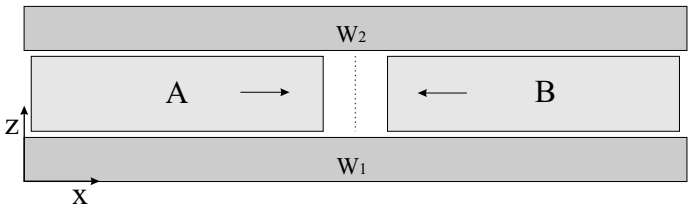

Figure 2.6: Two fluids A and B are moving toward each other in a 2D pipeline consisting of the obstacles W_1 and W_2. After some time the fluids collide at the dot-and-dash line.

the pressure of the fluid is hydrostatic. These properties lead to

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + g\frac{\partial h}{\partial x} = 0 \tag{2.6}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + g\frac{\partial h}{\partial y} = 0 \tag{2.7}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\Big(u(h-b)\Big) + \frac{\partial}{\partial y}\Big(v(h-b)\Big) = 0 \tag{2.8}$$

which are the *Shallow Water equations* (abbreviated SWE) in 2D. The equations use the notation described in section 2.4, and g is the gravity, while t is the time.

Equation (2.6) and (2.7) are derived from equation (2.3) and the equations are still nonlinear due to the velocity advection term

$$(\mathbf{w}\cdot\nabla)\mathbf{w} = \left(\begin{bmatrix} u \\ v \end{bmatrix}\cdot\nabla\right)\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \end{bmatrix}$$

which is written as the general left-hand side in (2.3) while it is written as the specific 2D right-hand side in (2.6) and (2.7).

2.7.1 A Practical Explanation of the SWE

The SWE might be a bit hard to understand, thus in this section we will provide a more practical approach to the equations in 1D:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + g\frac{\partial h}{\partial x} = 0 \tag{2.9}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\Big(u(h-b)\Big) = 0 \tag{2.10}$$

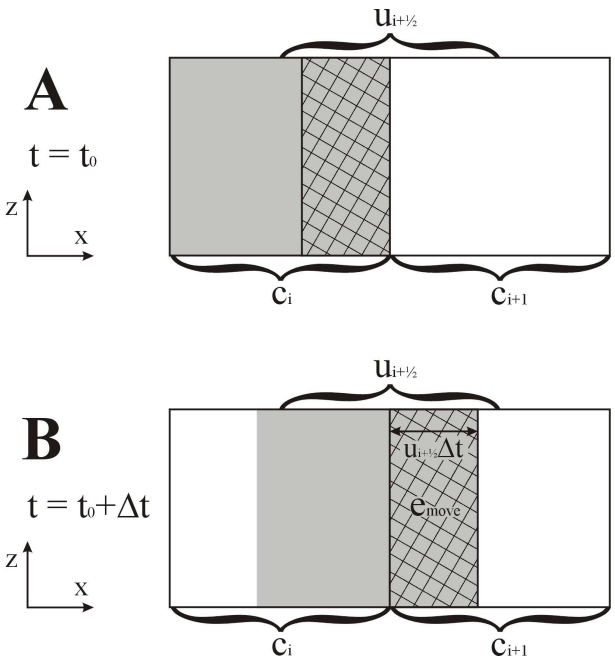

Figure 2.7: The figure shows how the hatched volume is moved from c_i to c_{i+1} during a time step. e_{move} denotes the volume which is obtained by extrusion of the hatched area along the y-axis by distance Δy.

We will explain the equations using the volume interpretation from section 2.4.1. Figure 2.7A shows two cells from the xz-plane of figure 2.3 at time $t = t_0$. The fluid which is inside cell c_i at time $t = 0$ is colored gray. All velocities are positive facing towards right, thus after some time Δt, some of the gray fluid has moved into cell c_{i+1}. The state after the movement has been shown on figure 2.7B at time $t = t_0 + \Delta t$. The hatched area represents the volume of fluid, which has moved from c_i to c_{i+1} during the time step. This volume becomes

$$e_{move} = u_{i+1/2}\tilde{d}_{i+1/2}\Delta t\Delta y, \tag{2.11}$$

since $u_{i+1/2}\Delta t$ is the distance covered by the fluid during the time step. Similarly the volume transferred from c_{i-1} to c_i is $u_{i-1/2}d_{i-1/2}\Delta t\Delta y$. The change in volume at c_i becomes the difference between the two volume transfers just determined:

$$\left(u_{i-1/2}\tilde{d}_{i-1/2} - u_{i+1/2}\tilde{d}_{i+1/2}\right)\Delta t\Delta y.$$

This corresponds to an approximation of the volume advection term

$$\frac{\partial h}{\partial t} = -\frac{\partial}{\partial x}\big(ud\big)$$

from equation (2.10).

Now we will explain the VEAT which occurs whenever there is a velocity difference $\frac{\partial u}{\partial x}$ between cell c_i and c_{i+1}. Consider a situation where

$$0 < \tilde{u}_i < \tilde{u}_{i+1}.$$

This means that e_{move} will have a lower velocity than the rest of the volume at cell c_{i+1}. Therefore $\tilde{u}_{i+1}$ should lose velocity which is proportional to $e_{move}\frac{\partial u}{\partial x}$. From equation (2.11) we see that e_{move} is proportional to $u_{i+1/2}$, thus the velocity loss becomes

$$u\frac{\partial u}{\partial x}$$

which is the VEAT.

Finally the gravity term states that a height difference $\frac{\partial h}{\partial x}$ affects the fluid by a force from the high area towards the low area.

2.7.2 Applications

The SWE have a wide number of applications ranging from prediction of flooding [13] to real-time simulations like Kass [15] and Layton [2]. Kass simulated the equations without the velocity advection term, thus the equations become linear. Layton implements the full SWE, however the boundary conditions are simplified so that the height of the ground under the fluid is constant[6]. In other words the boundary cells remain static during the simulation. Consequently, none of them have solved the nonlinear shallow water equations with a free boundary.

One of the problems with the velocity advection term is that it tends to make errors when $\left|\frac{\partial(h-b)}{\partial x}\right|$ is large since it transfers velocities without weighting the depths. This means that a small depth can affect a large depth in the same manner as if the depths were equal. The problem is handled in 3.2.2.

The next section describes a modification of the SWE so that the velocity advection term is weighting the depths.

2.7.3 The Conservative Equations

The shallow water equations exist in an alternative form called the conservative form since it conserves the momentum. In this form no velocities u and v are stored in the height map. Instead the velocities have been substituted with *discarges* U and V defined as:

$$U = ud, \qquad V = vd. \tag{2.12}$$

Most of the articles test their solvers against analytic solutions (examples are [24] and [10]). However, the analytic solutions have only been derived for very simple initial conditions like the breaking dam experiment. Consequently, no articles have been found where the solvers are tested on complex topology. For example a situation where the fluid passes a bed with high slope.

In order to handle discontinuities such as the front of the breaking dam, the Riemann solver developed in [27] is typically used. The book [28] contains a thorough describtion of SWE and Riemann solvers.

[6]The boundary restriction is not explicitly stated in the report but all the tests have got this restriction.

2.8 Conclusion

In this chapter we have discussed various methods to simulate fluid with extended focus on the height map methods with running time $\mathcal{O}(n^2)$. The height map methods we have found fall into three groups:

- A 2D solution of Navier-Stokes Equations where the pressures are used to determine the depth of the fluid. We have found some problems with waterfalls which were not discussed in Chen's report.
- A couple of implementations of the non conservative shallow water equations have been developed. These implementations run in real time however the boundary problems have not been solved for the complete SWE.
- A solution to the conservative shallow water equations. This method is primarily used to simulate large-scale clima and environmental evolution thus no articles has been found for real-time simulation.

The problems with Chen's method makes it unqualified as general purpose solver in landscapes. It might be possible to use Chen's method to increase the amount of details on the surface of the fluid simulated by SWE. However, this task is rather comprehensive thus in the next chapters we will only examine SWE.

Chapter 3

Non Conservative SWE

First we will examine the possibilities in the non conservative version of SWE described in section 2.7. The height map from section 2.4 is used as representation of the fluid. In order to simplify the examination we will consider the 1D version of the SWE:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + g\frac{\partial h}{\partial x} = 0 \tag{3.1}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\Big(ud\Big) = 0 \tag{3.2}$$

Before we can solve these equations, we need to decide whether to use center velocities or edge velocities from section 2.1.

3.1 The Velocity Representation

The velocity representation settles whether equation (3.1) is evaluated at an edge or at the center of a cell. For instance if it is center velocities we want to determine $\left\langle \frac{\partial u}{\partial t} \right\rangle_i$ while it is $\left\langle \frac{\partial u}{\partial t} \right\rangle_{i+1/2}$ for edge velocities where $i \in Z$. Equation (3.2) is always evaluated at the center of the cells, since that is where the height are.

Whether to choose center velocities or edge velocities is primarily a matter of simplification in connection with the finite difference operators. For example the gravity term $g\frac{\partial h}{\partial x}$ from equation (3.1) becomes

$$\left\langle g\frac{\partial h}{\partial x} \right\rangle_{i+1/2} = g\frac{h_{i+1} - h_i}{\Delta x} \tag{3.3}$$

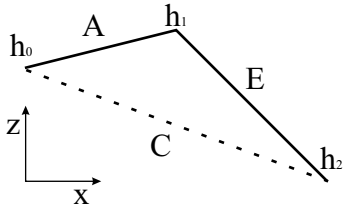

Figure 3.1: The figure shows that the gravity on the edge velocity $v_{1+1/2}$ is determined directly by slope E. However, the center velocity v_1 is determined by slope C which is the average of slope A and E.

using the finite difference operator on edge velocities. However, for center velocities the term becomes

$$\left\langle g\frac{\partial h}{\partial x}\right\rangle_i = g\frac{h_{i+1}-h_{i-1}}{2\Delta x} \tag{3.4}$$

which involves the average of the two nearest edges to cell $c_{i,j}$ along the x-axis. Figure 3.1 shows a 1D height map, where the edge velocities operates directly on the slopes of the edges, whereas the center velocities operates on averages of the slopes.

Another way to compare the two solutions, is to consider the distance from the position at which the approximation takes place to the position of the values in the height map which are used in the approximation. This distance is defined as the *approximation distance*. For example in equation (3.3) the approximation takes place at index $i+1/2$, thus the distance to the position of the values used in the height map becomes $\Delta x/2$. However, for the center velocity the distance becomes Δx. In the differential equations this distance goes towards zero, thus equation (3.3) is prefered since it has a smaller distance than equation (3.4).

For the volume advection term (abbreviated voat.) $\frac{\partial}{\partial x}\big(ud\big)$ with center velocity the approximation becomes:

$$\left\langle \frac{\partial}{\partial x}\big(ud\big)\right\rangle_i = \frac{u_{i+1}d_{i+1}-u_{i-1}d_{i-1}}{2\Delta x}$$

thus the approximation distance is Δx. However, for the edge velocity the approximation becomes:

$$\left\langle \frac{\partial}{\partial x}\big(ud\big)\right\rangle_i = \frac{u_{i+1/2}(d_i+d_{i+1})-u_{i-1/2}(d_{i-1}+d_i)}{2\Delta x}$$

thus the approximation distance is $\Delta x/2$ for the velocity while it is still Δx for the depths.

For the velocity advection term (abbreviated VEAT) along the x-axis there is no numerical difference since the center velocity gives:

$$\left\langle u\frac{\partial u}{\partial x}\right\rangle_i = u_i\frac{u_{i+1}-u_{i-1}}{2\Delta x}$$

whereas the edge velocity gives:

$$\left\langle u\frac{\partial u}{\partial x}\right\rangle_{i+1/2} = u_{i+1/2}\frac{u_{i+3/2}-u_{i-1/2}}{2\Delta x}. \tag{3.5}$$

We conclude that the gravity term is best approximated with the edge velocities. For the two advection terms the difference between the two velocity approximations is small. Therefore the edge velocity approximation is chosen. We can now insert the approximated terms into equation (3.1) and equation (3.2) to get:

$$\left\langle \frac{\partial u}{\partial t}\right\rangle_{i+1/2} = -u_{i+1/2}\frac{u_{i+3/2}-u_{i-1/2}}{2\Delta x} - g\frac{h_{i+1}-h_i}{\Delta x} \tag{3.6}$$

$$\left\langle \frac{\partial h}{\partial t}\right\rangle_i = -\frac{u_{i+1/2}(d_i+d_{i+1})-u_{i-1/2}(d_{i-1}+d_i)}{2\Delta x} \tag{3.7}$$

These equations can be applied in an ODE solver like Eulers-method. However, there are certain problems with these equations which have already been pointed out briefly in section 2.7.2.

3.2 Velocity Advection Problems

The VEAT causes problems which are to be examined. For simplicity we consider equation (3.6) without the gravity.

3.2.1 The Term is Too Local

The first problem occurs when there is an edge $u_{i+1/2}$ with zero velocity. Since $u_{i+1/2}$ is one of the factors in the VEAT, the right-hand side in equation (3.6) gives zero. Consequently, $u_{i+1/2}$ will continue to be zero forever. Clearly this

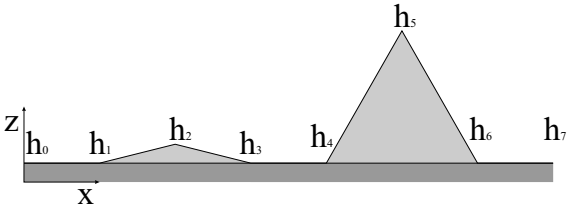

Figure 3.2: A small drop of fluid at h_2 floats to the right while the large drop of fluid at h_5 floats to the left. The two drops has equal absolute velocity thus $u_{1+1/2} = u_{2+1/2} = -u_{4+1/2} = -u_{5+1/2}$.

is not realistic since $u_{i+1/2}$ should change if fluid is approaching the edge. The problem is that the $u_{i+1/2}$ is too local thus the problem can be solved by substituting this factor with the average of the two neighbour velocities. After this modification equation (3.6) becomes:

$$\left\langle \frac{\partial u}{\partial t} \right\rangle_{i+1/2} = -\frac{u_{i+3/2}^2 - u_{i-1/2}^2}{4\Delta x} - g\frac{h_{i+1} - h_i}{\Delta x} \tag{3.8}$$

3.2.2 The Depths should have Influence

The next problem occurs when there are large variations in the depths. For example figure 3.2 shows a situation where two drops of fluids collide between h_3 and h_4. The two drops have the same absolute velocity, thus we would expect the fluid to move to the left after the collision since the right most drop has the largest volume. However, this does not occur because the VEAT does not contain any depths. In the concrete example $u_{3+1/2}$ continues to be zero forever since all the velocities continue to be symmetric around $u_{3+1/2}$.

The most obvious solution would be to weight the velocities involved in the VEAT. Then the term becomes

$$\left\langle u\frac{\partial u}{\partial x} \right\rangle_{i+1/2} = \frac{d_{i+3/2}u_{i+3/2}^2 - d_{i-1/2}u_{i-1/2}^2}{2\Delta x(d_{i+3/2} + d_{i-1/2})}, \tag{3.9}$$

where the depths are the average of the two nearest depths. After this modification the fluid on figure 3.2 will move to the left after the collision.

3.2.3 The Term is too Global

While the solution proposed in section 3.2.1 solves a problem, it creates a new problem. The problem is perhaps best described by an example like the breaking dam experiment shown at figure 3.3. In this example equation (3.1) is considered with the gravity term and the updated VEAT from equation (3.9).

We will now examine the course of $u_{3+1/2}$ which is determined by

$$\left\langle \frac{\partial u}{\partial t} \right\rangle_{3+1/2} = -\left\langle u\frac{\partial u}{\partial x} \right\rangle_{3+1/2} - \left\langle g\frac{\partial h}{\partial x} \right\rangle_{3+1/2}. \tag{3.10}$$

When the breaking dam starts at figure 3.3A, the only velocity that will become different from zero in the first time step is $u_{4+1/2}$. Quickly the velocity $u_{4+1/2}$ will increase due to the gravity. The rest of the velocities are relatively low compared with $u_{4+1/2}$ thus we have that

$$\left\langle u\frac{\partial u}{\partial x} \right\rangle_{3+1/2} = \frac{d_{4+1/2}u_{4+1/2}^2 - d_{2+1/2}u_{2+1/2}^2}{2\Delta x(d_{4+1/2} + d_{2+1/2})} \; > \; 0 \tag{3.11}$$

and since this term occurs with negative sign in (3.10) we have that $u_{3+1/2}$ will become negative since the gravity term is relatively small in (3.10). Therefore we get the situation shown at figure 3.3B. Clearly this behaviour is different from a real breaking dam, and it is also different from the analytic solution of SWE. The problem is that equation (3.11) is too global. The equation approximates the value $u_{3+1/2}$ by the average of the two neighbours and due to the large value of $u_{4+1/2}$ the approximation becomes positive even though the value of $u_{3+1/2}$ is negative.

3.2.4 Local Term with Weighted Depths

In this section we will go back to the original local VEAT from equation (3.5) and extend it with the weighted depths described in section 3.2.2. Consequently, the extended VEAT becomes[1]:

$$\left\langle u\frac{\partial u}{\partial x} \right\rangle_{i+1/2} = u_{i+1/2}\frac{\tilde{d}_{i+3/2}u_{i+3/2} - \tilde{d}_{i-1/2}u_{i-1/2}}{2\Delta x(\tilde{d}_{i+3/2} + \tilde{d}_{i-1/2})}, \tag{3.12}$$

[1]The tilde in equation (3.12) indicates that the corresponding value does not exist in the height map, thus it should be approximated from the neighbours.

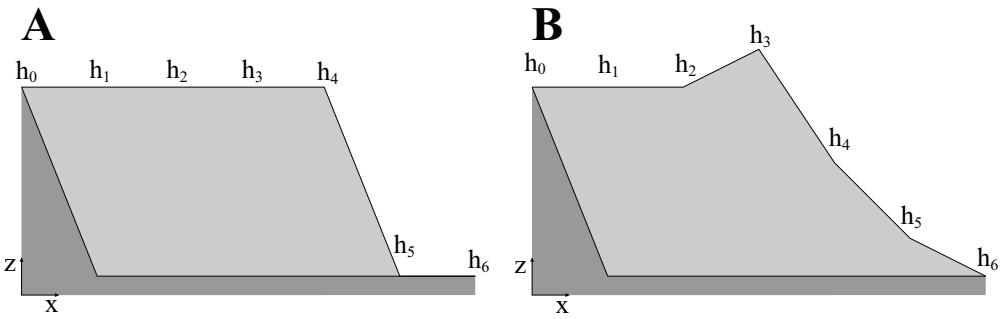

Figure 3.3: The figure shows a breaking dam experiment for the global VEAT. Figure A shows the initial state of the experiment while figure B shows the fluid after some time.

Since it is a local VEAT it has the problem described in section 3.2.1.

Anyway, to complete our investigation of the various methods we will test the solution on the breaking dam problem. The result of the simulation is shown on figure 3.5 while figure 3.4 shows a schematic version. We observe that $u_{4+1/2}$ becomes so high that $h_4 < h_5$, and it starts a short wave on figure 3.4B. Further tests on various height map resolutions show that this wave has a wavelength which is $2\Delta x$. Naturally the wavelengths should be independent on Δx, thus we should strive to avoid these kinds of waves. Figure 3.5 shows how the wave remains unchanged on the front of the fluid.

So why do these artificial short waves occur? Well here is a hypothesis:

The short waves are created by the high value of $u_{4+1/2}$ in the start of the simulation. The high value of $u_{4+1/2}$ should imply that the VEAT should transfer it to the right. However, this does not happen due to the small difference between $u_{3+1/2}$ and $u_{5+1/2}$.

In the next chapter we will propose a new model of the VEAT term which will transfer the velocity away from $u_{4+1/2}$ even if $u_{3+1/2}$ and $u_{5+1/2}$ were equal. However we will first finish the discussion of the non conservative SWE by considering a model without the VEAT.

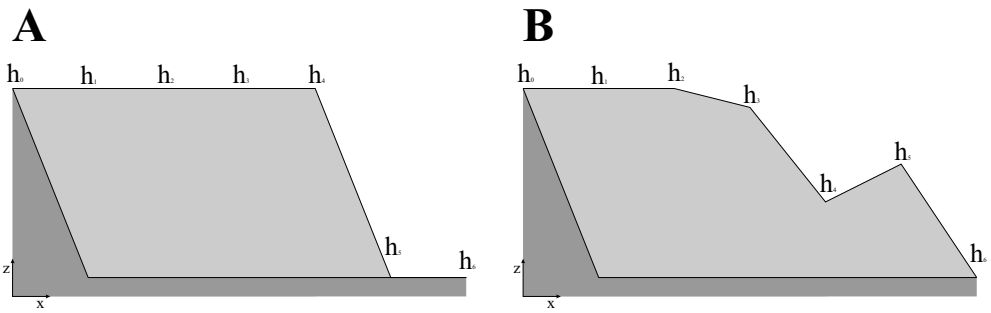

Figure 3.4: The figure shows a breaking dam experiment for the local VEAT Figure A shows the initial state of the experiment while figure B shows the fluid after some time.

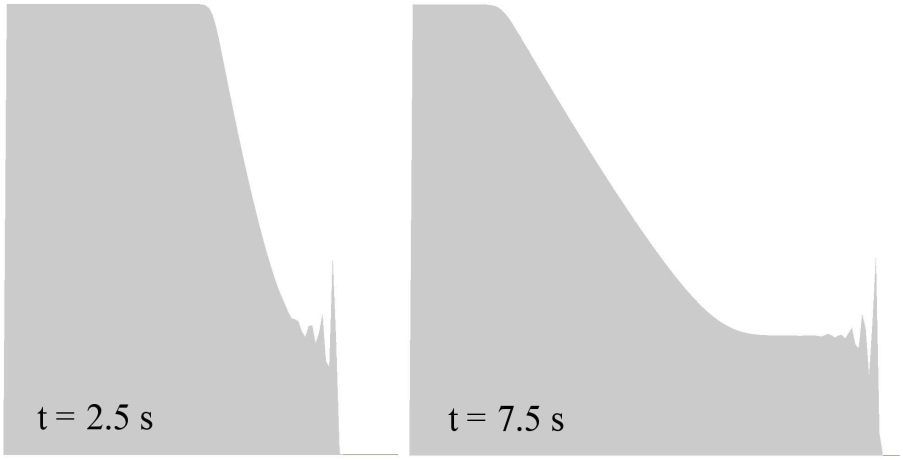

Figure 3.5: The figure shows two states from the breaking dam experiment described schematically on figure 3.4. The left and right images show the state at time $t = 2.5s$ and $t = 7.5s$ respectively.

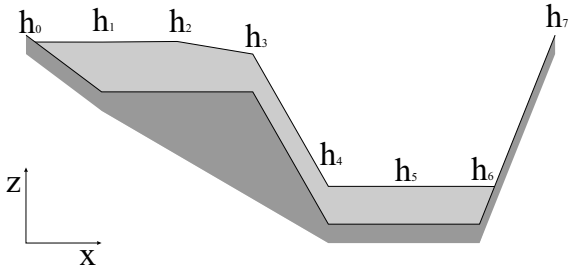

Figure 3.6: The figure shows a waterfall at $u_{3+1/2}$ which has reached a stationary state. There is a source at h_1 and a mouth at h_6. The source and the mouth has the same flow rate which remains constant. After an amount of time the fluid will reach a stationary state as shown on the figure.

3.3 A Model without VEAT

Due to the VEAT problems we will now consider a model without the VEAT as it has been done by Kass [15]. The missing VEAT implies a couple of artifacts which are to be described.

The worst artifact we have found occurs when the model is used to simulate a waterfall. Figure 3.6 shows an example of a waterfall which has reached a stationary state due to the source and the mouth. However, this does not happen when the VEAT is missing since $u_{3+1/2}$ increases forever due to the gravity term. In other words the velocity $u_{3+1/2}$ goes towards infinity since the VEAT does not transfer the slow velocity at $u_{2+1/2}$ into $u_{3+1/2}$.

In order to avoid this problem we introduce an artificial resistance term

$$a_r u^3$$

so that we obtain the stationary state at figure 3.6. The high power has been chosen so that the term does not remove the waves with low velocity, thus the term is not meant to be a realistic term.

Furthermore, there are still a couple of small waves with wavelength $2\Delta x$ at the front of a breaking dam. These waves are so small that we can remove them by introducing an artificial pressure term so that the modified SWE become:

$$p = gh + a_p \frac{\partial h}{\partial t} \tag{3.13}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial p}{\partial x} + a_r u^3 = 0 \tag{3.14}$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\Big(ud\Big) = 0 \tag{3.15}$$

For zero gravity equation (3.13) states that the pressure is high at the areas where the fluid accumulates.

We will not go into a long discussion about this model since it is based on artificial terms.

Chapter 4

Conservative Shallow Water

In this chapter we will come up with a new model of the VEAT, in a height map based on the conservative form of the SWE described in section 2.7.3. Consequently, the velocities in the height map have been substituted with discharges U and V defined by equation (2.12) which is repeated below:

$$U = ud, \quad V = vd \tag{4.1}$$

The conservative form of SWE is obtained by inserting the discarge definition into the SWE from equation (3.1) and (3.2):

$$\frac{\partial U}{\partial t} + u\frac{\partial U}{\partial x} + gd\frac{\partial h}{\partial x} = 0 \tag{4.2}$$

$$\frac{\partial h}{\partial t} + \frac{\partial U}{\partial x} = 0 \tag{4.3}$$

These equations could be solved in a similar way as it has been done in section 3.1. However, this will lead to problems similar to the ones described in 3.2. In order to solve the problems we will develop a new VEAT.

4.1 The Imperative VEAT

We will now describe a new way to determine the VEAT based on the principle from section 2.7.1, where the volume transferred from c_i to c_{i+1} was determined in equation (2.11) for velocities[1]. This equation can be converted

[1]The method is called imperative since for each pair of neighbour cells c_i and c_{i+1} it applies changes to the discharges in an imperative manner corresponding to the discharge transfered in the volume from c_i to c_{i+1}.

Algorithm 1

1. Determine q_{total} from the discharges in the height map.
2. Use equation (4.6) to determine q_{move}.
3. Apply q_{move} to the discharges in the height map.

into discharges and the transferred volume becomes

$$e_{move} = U_{i+1/2}\Delta t \Delta y, \tag{4.4}$$

when $0 < U_{i+1/2}$. The transfer of e_{move} transfers discharge from c_i to c_{i+1} thus it is the purpose of the VEAT to determine how this transfer affects the discharges in the height map. The discharge contained in e_{move} can be determined by assuming that the discharge is distributed even inside c_i:

$$\frac{q_{move}}{e_{move}} = \frac{q_{total}}{e_{total}}, \tag{4.5}$$

where q_{move} is the discharge in e_{move}, while q_{total} and e_{total} are the discharge and the volume respectively of the entire cell c_i. The volume e_{total} has been determined in section 2.4.1 thus by inserting equation (4.4) into (4.5) we get

$$q_{move} = q_{total}\frac{U_{i+1/2}\Delta t}{d_i \Delta x}. \tag{4.6}$$

We can now describe the structure of the imperative VEAT algorithm which is shown in algorithm 1.

Step 2 is straightforward, while there are various solutions to step 1 and 3. We have found four solutions to step 3 as shown on figure 4.1. Each arrow indicate that a discharge q_{move} is transferred from the rear to the front of the arrow. In step 1 q_{total} is determined from the rear of the arrow. This is necessary to avoid situations where we remove discharge from areas without discharge.

Solution A and B look promising since they are symmetric around the transfer from c_i to c_{i+1}. However, both of them have got the problem that they are too global, thus they generate short waves with wavelength $2\Delta x$.

Solution C has problems when the gravity is zero, since the discharge cannot move into areas with zero discharge.

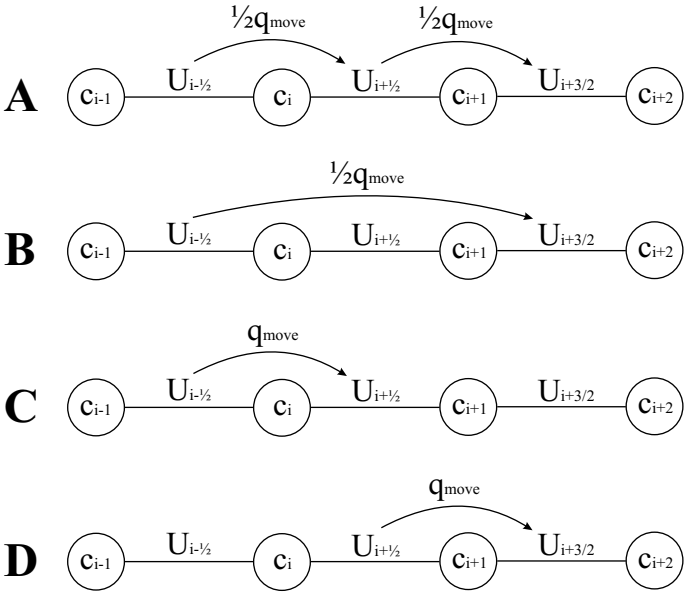

Figure 4.1: The figure shows four solutions to the VEAT. The arrows show how the discharge is transferred. For example q_{move} is removed from $U_{i-1/2}$ and added to $U_{i+1/2}$ in solution C.

Algorithm 2 *This is a specific imperative algorithm to determine the VEAT For simplicity it will only consider the situation where* $0 < U_{i+1/2}$. *Note that we change the values of* $\Delta U_{i+1/2}$ *during the algorithm, thus it works in an imperative way in contrast to the traditional VEAT.*

1. For each discharge $U_{i+1/2}$ set $\Delta U_{i+1/2} \leftarrow 0$
2. For each discharge $U_{i+1/2}$ do the following steps:
   (a) $q_{total} \leftarrow U^n_{i+1/2}$
   (b) Use equation (4.6) to determine q_{move}.
   (c) $\Delta U_{i+1/2} \leftarrow \Delta U_{i+1/2} - q_{move}$
   $\Delta U_{i+3/2} \leftarrow \Delta U_{i+3/2} + q_{move}$
3. Now $\Delta U_{i+1/2}$ holds the change caused by the VEAT on $U_{i+1/2}$ thus we can add the term to the SWE.

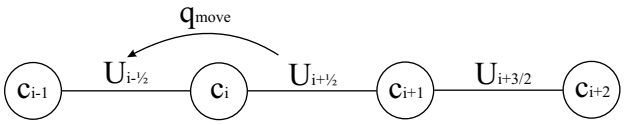

Figure 4.2: The figure shows how the VEAT is handled when $U_{i+1/2} < 0$.

Instead we will choose solution D and make the assumption that the discharges can be displaced so that $U_{i+1/2}$ is the discharge at cell c_i while $U_{i+3/2}$ is the discharge at cell c_{i+1}. Furthermore, we will add an extra variable $\Delta U_{i+1/2}$ for each discharge $U_{i+1/2}$ in the height map [2]. This is enough to describe a specific algorithm to calculate the VEAT as shown in algorithm 2.

Until now we have only considered the situation where $0 < U_{i+1/2}$, thus we need to consider the situation where $U_{i+1/2} < 0$ which should be handled as shown on figure 4.2. When q_{move} is determined remember that e_{total} should refer to the volume of cell c_{i+1} since it is the source cell.

[2]Note that the new variable $\vec{U}_{i+1/2}$ is just a temporary in each time step so we do not need to extend the representation of the fluid.

4.1.1 Discharge Conservation

An important property of the imperative VEAT is that the sum of all discharges remain constant during the VEAT. This is easy to prove since each time we modify ΔU we add q_{move} to one variable ΔU_i and subtracts q_{move} from another variable ΔU_j. Consequently, the sum of all the ΔU's remains zero.

4.2 Stability

A stability analysis is beyond the scope of this report. Instead we will briefly describe the problems with the Euler-method used in this project.

The main problem with the Euler-solver is the restrictions on the size of the time step. For instance if the time step is too large the gravity will overreact such that

$$\frac{\Delta u_{i+1/2}}{u_{i+1/2}} < -2, \quad \text{where} \quad \Delta u_{i+1/2} = \Delta t \left\langle \frac{\partial u}{\partial t} \right\rangle_{i+1/2}$$

is satisfied the method will diverge.

4.2.1 Discussion

The Euler-method has primarily been chosen such that we could keep the focus on the SWE and the visualization of the fluid.

However the Euler-method still has its applications in large-scale landscapes where Δx is so large that $u\Delta t < \Delta x$.

The simulation could be improved by substituting the Euler-method with an implicit method as described in [15] and [2]. In this way larger time steps could be used. However the visualization will not remain smooth in time, thus it will be necessary to interpolate between the frames determined by the numerical method. This is beyond the scope of this project since the interpolation method is quite comprehensive to implement at the boundary of the fluid where horizontal interpolation is necessary.

Chapter 5

The Boundary of the Fluid

The general physical model assumes that the fluid fills each grid cell homogeneously. However this is not the case for the cells which have empty cells as neighbours thus we will develop a new model for the boundary.

Formally we will define the *boundary of the fluid* as the set of cells c_i which have a neighbour cell c_j such that

$$0 < d_i \bigwedge d_j = 0$$

is satisfied.

The following sections describe the various boundary problems and some of their possible solutions.

5.1 Fluid Floating Down a Slope

Most of the boundary problems can be illustrated by the situation shown at figure 5.1. The figure shows a drop of fluid floating down a slope. If one of the general fluid simulations from chapter 3 and 4 is used to simulate the situation two problems will occur:

5.1.1 The Fluid will Never Leave a Cell

The first problem occurs due to the assumption that the fluid fills the cells homogeneously. Therefore it is not possible to transfer all the fluid in one cell to another cell since it implies that the velocity of the fluid grows towards infinity for $\Delta t \to 0$. Consequently the fluid will never leave a cell.

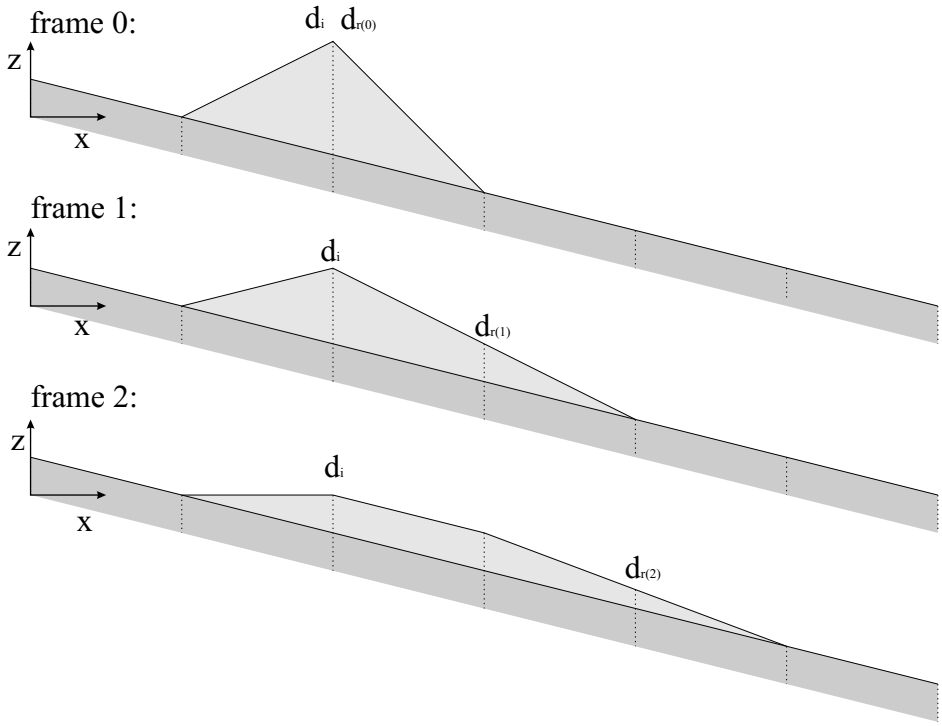

Figure 5.1: The figure shows 3 successive frames of a drop of fluid floating down a slope. The fluid is simulated by the general physical model without any considerations of the boundary. Cell i is the leftmost cell while cell $r(n)$ is the rightmost cell at frame n.

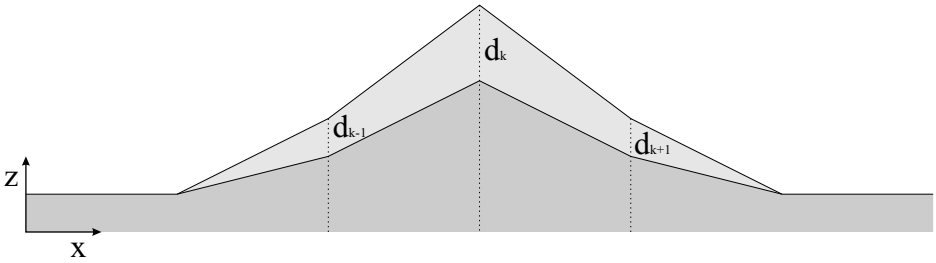

Figure 5.2: An example where the fluid should disappear from cell d_k which is not a boundary cell.

5.1.2 The Boundary Velocity Grows towards Infinity

Consider the rightmost cell $d_{r(n)}$ of the fluid on figure 5.1. Due to the gravity the velocity $v_{r(n)+1/2}$ is positive and therefore an amount of fluid is transfered from $d_{r(n)}$ to $d_{r(n)+1}$. This means that for each time step the fluid enters a new cell. Consequently the velocity of the boundary of the fluid grows towards infinity for $\Delta t \to 0$.

5.2 Determining the Cells Near the Boundary

In order to solve the boundary problems we will modify the general fluid simulation such that an alternative behavior is used for the boundary behaviour cells (abbreviated bb. cells) while the general fluid simulation keeps running unchanged for the rest of the cells.

In order to apply the boundary behaviour it is necessary to find a suitable definition of the bb. cells. The bb. cells might be defined such that they are equal to the boundary cells. The problem with this solution is that cells which are about to become boundary cells do not necessarily get the boundary behaviour. For instance consider the situation shown at figure 5.2. Here the fluid should disappear from cell k due the gravity. This does not happen since k does not have any neighbour without fluid.

Instead the bb. cells are defined as the cells i for which the depth is below a given threshold $d_i < \alpha$. In this way the depth of the fluid in cell k on figure 5.2 will get below α after some time and the boundary behaviour will ensure that the fluid disappears from the cell.

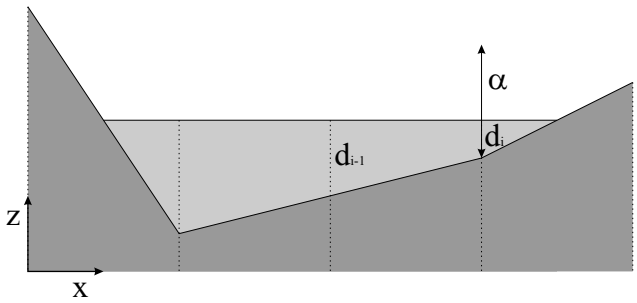

Figure 5.3: The figure shows a lake where the slope of the ground is smaller than α. This means that the depth of the rightmost cell i is smaller than α and hence the lake is removed after some time. Note that the value of α is relatively large on the figure to avoid that the lines huddle together. In the real simulation $\alpha = \frac{\Delta x}{10}$.

5.3 Invisible Fluid

The boundary problems could be solved by making the fluid invisible as soon as the depth gets below the threshold α. However this solution introduces a new problem namely that the volume conservation is broken. For example all the fluid on figure 5.1 will disappear after some time, and if there is a valley at the bottom of the slope the fluid will accumulate in a lake and hence it will reappear.

5.4 Removing Fluid

It is possible to avoid that the fluid reappears by removing fluid with a depth below α. But again the volume conservation is seriously broken since the simulation will remove all lakes for which the slope of the ground is below α no matter how large the lake is. For example consider the lake shown on figure 5.3. Since the slope of the ground to the right is below α, there will always be a cell i for which $d_i < \alpha$. When the fluid is removed from i the fluid in the lake will float into cell i and it will be emptied again. This will continue until $d_{i-1} < \alpha$, which implies that both cell i and c_{i-1} is emptied. The process continues until the entire lake is empty.

Algorithm 3 (Boundary Behaviour) *This algorithm describes how a depth is transfered from cell c_i to c_j, when c_i is a bb. cell. Let f be the (positive) depth which is transfered from c_i to c_j without the boundary algorithm. We will now describe how the bb. changes q such that we avoid the problems described in section 5.1.*

1. In order to avoid that the velocity increases at cells without fluid we set the velocity to zero:
   $u_{(i+j)/2} \leftarrow 0$
2. **if** $d_j = 0$ **then**
   $q \leftarrow 0$
   else
   $q \leftarrow \max(\min(d_i, \frac{h_i - h_j}{2}), 0)$

5.5 Modification of the Volume Advection

In order to ensure the volume conservation, we have chosen to search for a proper modification of the volume advection term. The modification has been shown in algorithm 3. The statement $q \leftarrow 0$ in step 2 ensures that the fluid will not float into an empty cell, such that we avoid infinite velocity at the front of the fluid on figure 5.1. The other part of the branch ensure that the rear of the fluid is horizontal. This is important in lakes which contain a mouth.

There is still one problem with the proposed algorithm. The problem occurs if the drop on figure 5.1 has a volume which is smaller than $\alpha \Delta x \Delta y$. In this case the fluid will not float down the slope since the first part of the branch in step 2 will be executed each time. Clearly we cannot remove the fluid since it will lead to the problems described in section 5.4.

Instead we will extend each cell c_i with a variable r_i which denotes the time elapsed since it was an empty cell. When r_i gets greater than a constant a_{dropt} we will allow the small drops to enter a new cell. This extension is implemented in algorithm 4.

Algorithm 4 (Boundary Behaviour with Small Drops)
This algorithm is an extension to algorithm 3 such that it will handle small drops of fluid.

1. $u_{(i+j)/2} \leftarrow 0$
2. **if** $d_j = 0 \ \wedge \ r_i < a_{\text{dropt}}$ **then**
   $q \leftarrow 0$

   else
   $q \leftarrow \max(\min(d_i, \frac{h_i - h_j}{2}), 0)$

Chapter 6

Results

6.1 The Breaking Dam Experiment

Figure 6.1 shows a breaking dam experiment, which compares a 1D height map with a SPH simulation from [22]. The 1D height map consists of 300 cells and uses the imperative VEAT from section 4.1. The SPH simulation uses 11250 particles thus the height map runs 10^3 to 10^4 times faster than the SPH simulation[1].

Generally the shape of the two fluids match reasonably well when we take into consideration that SPH has running time $\mathcal{O}(n^2)$ while the height map has running time $\mathcal{O}(n)$. The main problem with the height map is that the acceleration is too low thus the height map looks as if it models fluid with high resistance. Section 8.1.1 discuss some of the practical consequences of this problem.

6.2 A Bump Experiment

We shall now extend the breaking dam experiment from section 6.1 with a small bump. Figure 6.2 shows the extended experiment. The fluid hits the bump at $t = 15\ s$ and at $t = 17\ s$ we observe how the SPH particles fly in the air. Naturally the basic height map does not have this ability, however in chapter 8 the height map will be extended with flying particles.

[1]None of the two implementations has been optimized for speed thus it is only a rough estimate.

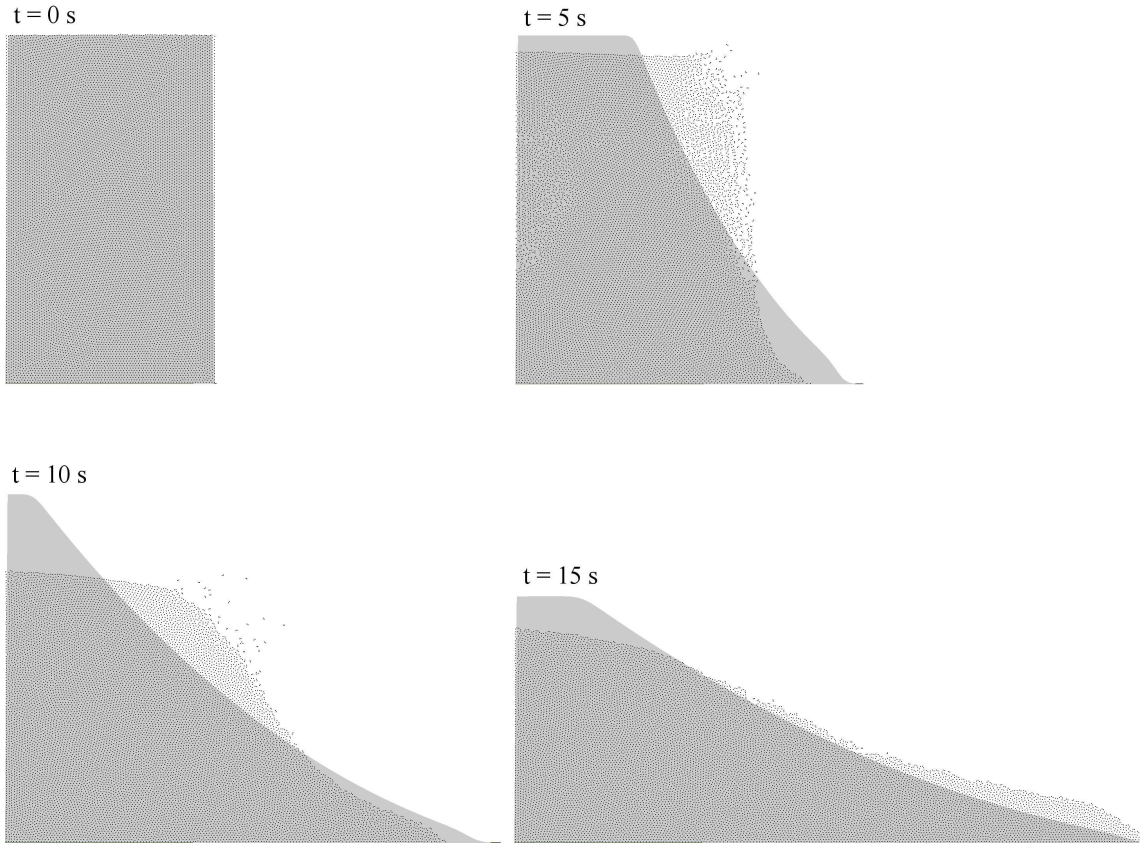

Figure 6.1: The figure shows four frames from a comparison between the SPH simulation from [22] and the 1D height map using the imperative VEAT from section 4.1. The tiny black dots are the sph particles while the light shaded shape is the height map.

At 20 $s < t < 35$ s the surface of the height map is not as smooth as the SPH The reason is that the fluid in the height map passes the bump almost unaffected. We observe that the fluid changes direction with almost 90 degrees without loss of velocity. This is not the case in the SPH where the velocity of the fluid to the left of the bump is zero at the bottom. In this way the fluid at the bottom actually works as a smoothing extension to the bump. This means that the SPH particles at the top has a smooth curve as if they were passing a smooth bump. It might be possible to approximate this behaviour in the height map by extending each cell with an extra depth which keeps track of the amount of still fluid at the bottom.

At 75 s the fluid becomes horizontal however the fluid keeps floating from left to right.

At 75 $s < t < 180$ s the velocity of the fluid decreases and hence the VEAT gets smaller influence. This implies that the gravity will get a more local influence and therefore the fluid starts accumulating to the left of the bump. The opposite effect occurs at the right of the bump where the gravity causes the fluid to increase velocity.

We observe the large slope of the fluid surface at the right side of the bump. This is an artifact since we would expect the fluid above the bump to flow to the right like a breaking dam. Therefore we cannot use the height map as a general purpose fluid simulation.

It should be notified that we have not read any other articles about SWE which contain test results on a rough bed. The only tests we have seen for height maps with rough bed are developed by Kass [15] but the method does not include the VEAT. Chen also shows a test on a rough bed but the test includes a source so we do not get the situation with low velocity at the left side of the bump and high velocity at the right side of the bump. Therefore the VEAT does not cause problems.

6.3 Conclusion

We conclude that the VEAT is fairly problematic to approximate by the finite difference method from section 3.2. Instead we developed an imperative VEAT with discharge conservation, which handles the breaking dam experiment quite well since the deviation to the SPH simulation is small. However the height map has fundamental problems as soon as a small bump is added to the ground. Consequently we have not been able to develop a general

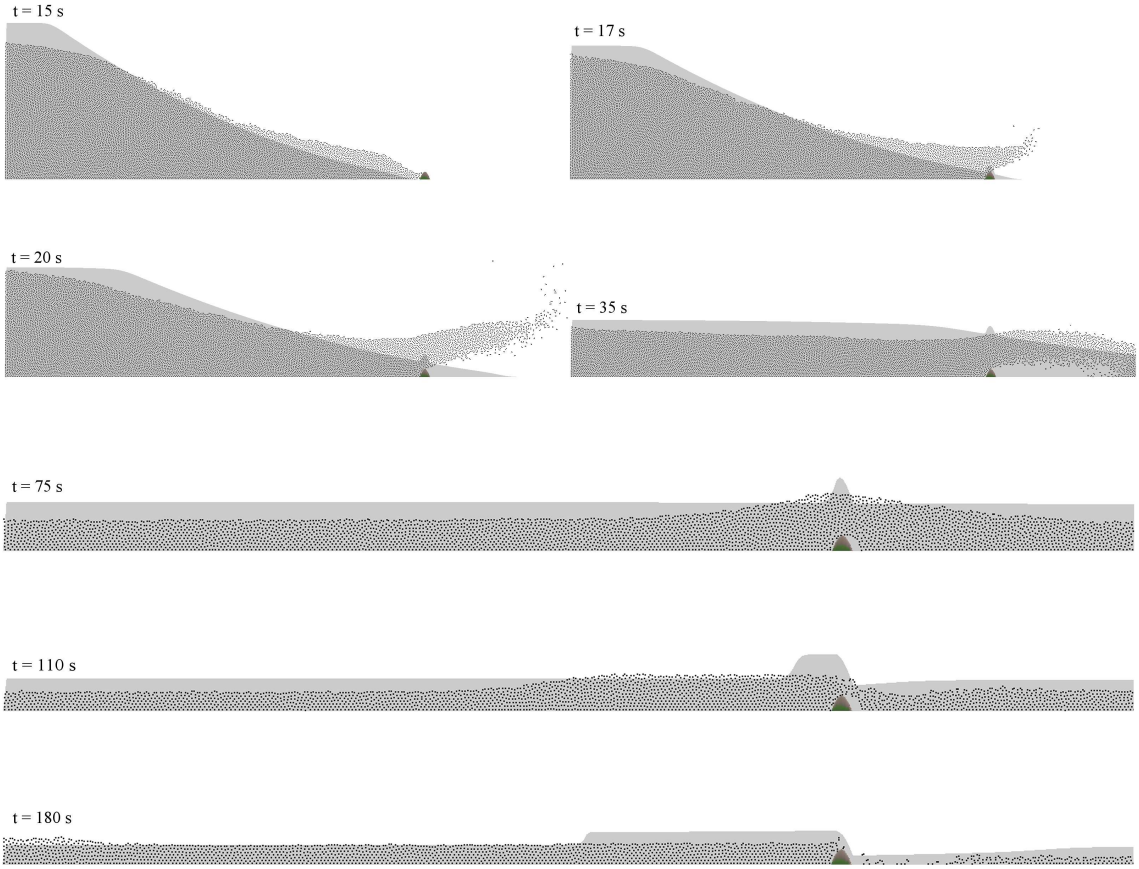

Figure 6.2: The figure shows seven frames from a breaking dam experiment which is an extension to the experiment from figure 6.1. Therefore the first frame at $t = 15\ s$ corresponds to the last frame from figure 6.1. The experiment has been extended with a small bump which has just been reached by the fluid at $t = 15\ s$. At the three last frames we have zoomed such that everything has been scaled by a factor 2 along each axis. The tiny black dots are the sph particles while the light shaded shape is the height map.

purpose height map with a VEAT.

It should be notified that we have not read any other articles about SWE which contain test results on a rough bed.

Instead we use a non conservative method without the VEAT as described in section 3.3 which corresponds to the model from Kass extended with an artificial resistance and a pressure term.

Finally we have developed a boundary algorithm in chapter 5 which conserves the volume of the fluid.

Part II

Visualization

We will now develop a visualization of the fluid simulation from the previous part. In the first chapter we approximate the *Geometry of the Fluid Surface* by a set of triangles so that we can render the surface with OpenGL. In the next chapter we extend the visualization with *Flying Particles* to model foam. The last chapter describes how to apply a texture to the fluid surface so that we get *Foam on the Fluid Surface*. The texture is also used to visualize the velocity of the fluid since the surface foam should float with the velocity of the fluid.

Chapter 7

Tessellating the Fluid Surface

In this chapter we will determine how to transform the fluid surface from the height map into triangles, so that we can render the surface by OpenGL. Every cell $c_{i,j}$ which is inside the fluid has a well-defined point $[i\Delta x, j\Delta y, h_{i,j}]^T$ in the fluid surface. Therefore it is easy to determine the triangles which are inside the fluid. Figure 7.1A shows how the surface from a 2D height map is divided into triangles. However we need to determine the triangles at the boundary or else there will be a gap between the fluid surface and the surface of the ground. The gap is shown more clearly at figure 7.1B which represents a 1D height map.

Four cells are defined to be a *square* iff. each of the cells has exactly two of the other cells as neighbours. The four cells marked with crosses on figure 7.1A form a square.

7.1 Closing the Gap at the Boundary

We will now describe how to close the gap between the fluid surface and the ground at the squares which are at the boundary of the fluid. Each cell can either be inside the fluid or outside the fluid thus there are 16 different squares to handle. The 16 cases can be formed by rotations of the 6 squares shown on figure 7.2. The bullets indicates cells which are inside the fluid whereas the circles indicate cells which are outside the fluid.

The situation on figure 7.2E is handled on figure 7.1A and at figure 7.2F all cells are outside the fluid thus no triangles should be drawed. At figure 7.2D there is no connection between the cells which are inside the fluid thus

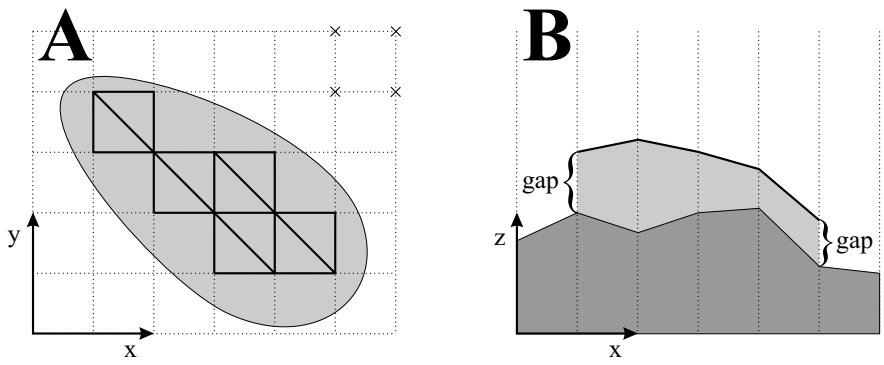

Figure 7.1: Internal triangles. Figure A shows a 2D height map whereas figure B shows a 1D height map

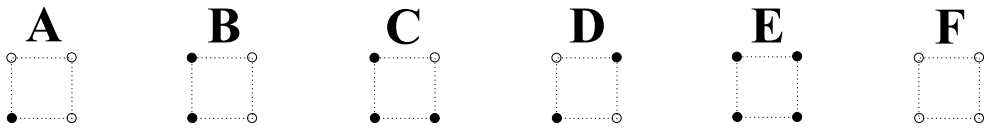

Figure 7.2: Six special cases.

the situation is handled by using the situation on figure 7.2A twice.

Consequently there are three special cases which must be solved. Figure 7.3 shows how the three situations are tessellated. Each asterisk is placed at the midpoint between the two nearest cells. The three special cases makes it possible to tessellate any fluid boundary, for instance figure 7.4 shows an example where the gaps have been closed.

Even though the gaps have been closed the boundary is still fairly rough. The boundary of the fluid is the area where users typically spot the dis-

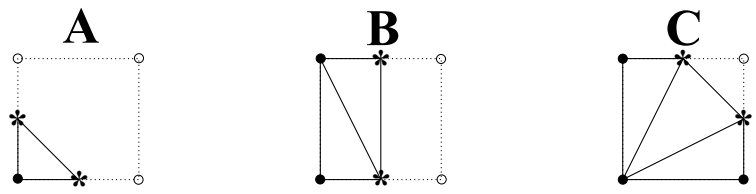

Figure 7.3: Tessellating three special cases.

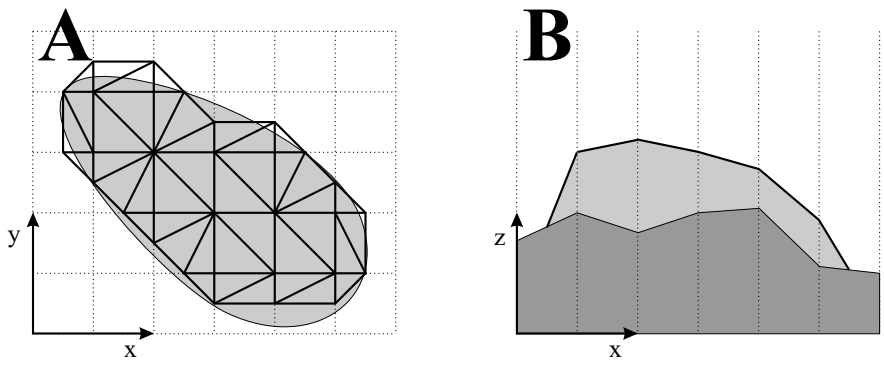

Figure 7.4: Complete tessellation. Figure A shows a 2D height map whereas figure B shows a 1D height map

cretization of the fluid. Since the number of boundary vertices is $O(n)$ while the total number of vertices is $O(n^2)$ it is typically more effective to smooth the boundary rather than increasing the resolution of the height map. In the next sections we will smooth the boundary in time and space, by carefully placing the asterisks on figure 7.3.

Note that the smoothing algorithm can be described without considering the three special cases from figure 7.3. We simply write an algorithm to place the boundary/asterisk between a cell which is inside the fluid and a neighbour cell which is outside the fluid.

7.2 Smoothing the Boundary in Time

The boundary of the fluid does not have a smooth movement as shown on figure 7.5. From $t = 0$ to $t = 2\Delta t$ the boundary remains stationary a discontinuity happens from $t = 2\Delta t$ to $t = 3\Delta t$. Instead we want to obtain a smooth movement as shown on figure 7.6.

7.2.1 A Boundary based on the Fluid Depth

A possible solution is to interpolate the boundary from the depth of the nearest cell. For instance at time $t = 2\Delta t$ we could use d_1 to determine the boundary between c_1 and c_2 as

$$\Delta x + a_0 d_1,$$

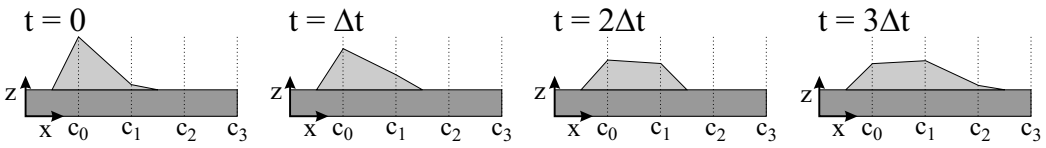

Figure 7.5: The figure shows a drop of fluid moving from left to right without time smoothing.

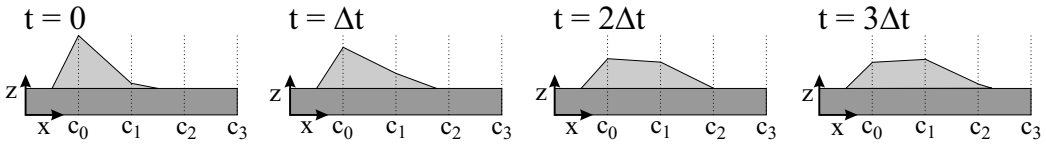

Figure 7.6: The figure shows a drop of fluid moving from left to right with time smoothing.

where a_0 is a constant. In this way the boundary moves to the right when the depth increases. The problem with this solution is that $\frac{\partial d}{\partial t}$ is changing quite much at the boundary. This means that the boundary velocity will not be constant, e.g. on figure 7.6 at time $t = \Delta t$ the boundary velocity is almost zero, whereas at $t = 2\Delta t$ the boundary velocity is relatively high.

7.2.2 A Boundary based on the Future

Instead we use a solution where the fluid simulation is a_{future} frames ahead of the visualization[1]. In this way we can determine when the fluid boundary passes a given cell in the future. Furthermore the position of the boundary is stored in a variable $m_{i+1/2}$ which denotes the distance from the boundary to c_i (assumed that there is a boundary between c_i and c_{i+1}).

We will now describe the boundary algorithm by the example shown on figure 7.6 at time $t = 0$. The algorithm should move the boundary so that we reach the situation shown on figure 7.6 at time $t = \Delta t$. Since the fluid simulation is a_{future} frames ahead of the visualization we know that the boundary of the fluid will pass c_2 at time $t = 2\Delta t$. By assuming that the boundary moves with constant velocity to c_2 we can determine the velocity of the boundary and upate $m_{i+1/2}$ correspondingly.

[1]The constant a_{future} should be chosen so large that the velocity $\frac{\Delta x}{a_{\text{future}}\Delta t}$ is so small that it appears stationary for the viewer.

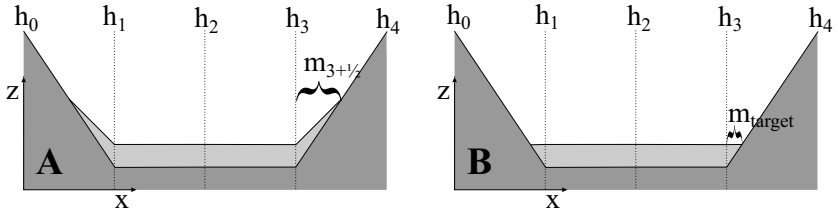

Figure 7.7: Obtaining a horizontal fluid surface at the boundary.

We have now described the position of the boundary when it is in motion, however, we need to describe to determine the position when the boundary does not pass any cells during the next a_{future} frames.

7.3 Smoothing the Boundary in Space

In this section we will described an algorithm which supports the algorithm from section 7.2.2 when the fluid boundary does not pass any cells during the next a_{future} frames.

7.3.1 Obtaining a Horizontal Fluid Surface

This algorithm moves the fluid boundary to make the fluid surface as horizontal as possible at the boundary. For instance at figure 7.7A the algorithm will move the boundary towards the situation shown on figure 7.7B. The distance between the right boundary and c_3 at figure 7.7B is determined as

$$m_{\text{target}} = d_3 \frac{\Delta x}{b_4 - b_3}.$$

This means that the boundary distance $m_{3+1/2}$ on figure 7.7A should move towards m_{target}.

The algorithm works well for lakes, however, in waterfalls a situation may occur where $b_4 < h_3$ (think of figure 7.7A as the cross section of a waterfall in a 2D height map). In this case we handle the situation as if $b_4 = h_3$ thus m_{target} becomes Δx. This implies that we get an angular boundary in the xy-plane as shown on figure 7.4. Therefore we will develop an extra smoothing algorithm in the next section.

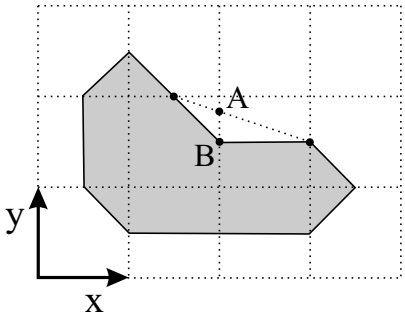

Figure 7.8: Smoothing in the xy-plane

7.3.2 Smoothing in the xy-plane

This algorithm is used after the algorithm in section 7.3.1 to ensure that the boundary is sufficiently smooth in the xy-plane. The algorithm uses a standard smoothing approach. It moves each vertex B in the boundary curve towards the average A of the two neighbour vertices as shown on figure 7.8. If the distance between A and B is below a given threshold $a_{\text{smoothness}}$ no smoothing is done. In this way we keep some of the details in the boundary while we remove the sharp edges.

7.4 Blurring the Boundary

The primary purpose of this section is to obtain a smoother transition between the fluid surface and the flying foam described in chapter 8. Figure 8.10 on page 78 shows the advantage of the blurring. A secondary purpose is to antialiase the boundary of the fluid.

7.4.1 Internal Boundary Blurring

The boundary could be blurred by making all the vertices on the boundary curve transparent. In other words all the asterisks on figure 7.3 become transparent. This is very easy to implement, however, there are several problems with the solution.

The worst problem is shown on figure 7.9 where the fluid is almost completely transparent at the dot-and-dash line even though it should be almost

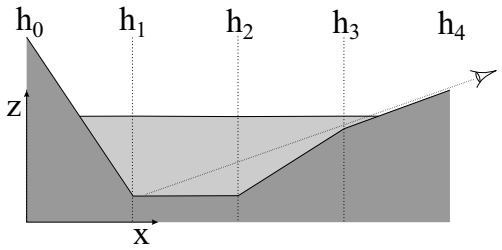

Figure 7.9: Unwanted Transparency.

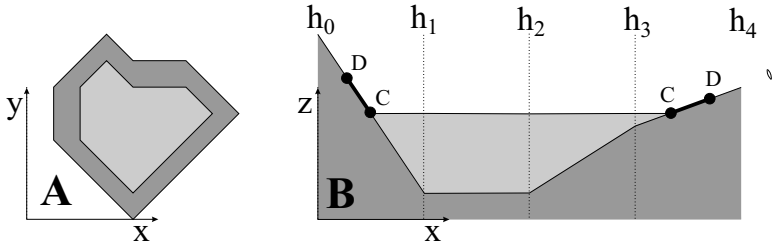

Figure 7.10: External Boundary.

opaque for the viewer. Naturally this is unrealistic due to the reflection and the loss of light from the long distance inside the fluid[2].

7.4.2 External Boundary Blurring

The external boundary blurring is more comprehensive to implement, since it adds an extra layer of triangles to the fluid surface as shown on figure 7.10. Figure 7.10A shows a 2D height map where the original fluid is light shaded whereas the external boundary is dark shaded. Figure 7.10B shows a 1D height map where the external boundary is opaque at position C, whereas it is completely transparent at position D. The thickness of the boundary is defined as the distance between C and D. To simplify the implementation of the boundary the thickness should be smaller than $\min(\Delta x, \Delta y)$.

The implementation is simple as long as there is no cell between C and D. Figure 7.11 shows how to handle the three special cases from figure 7.3 when there is no cell between C and D.

[2]Recall that the goal in this project is to visualize large-scale fluid thus the distance under the fluid is expected to be relatively long

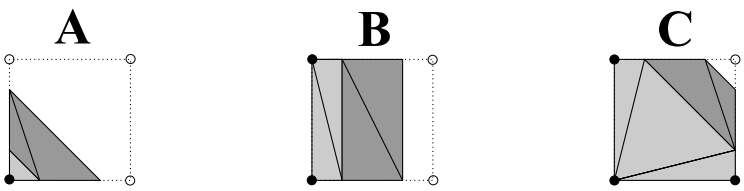

Figure 7.11: Three simple cases for the external boundary.

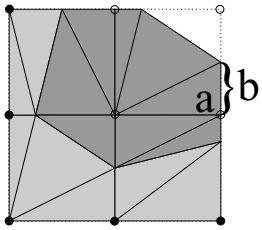

Figure 7.12: External boundary on a concave fluid area.

The algorithm gets more tricky when there is a cell between C and D, especially when the fluid area is concave as shown on figure 7.12. The problem is how to find and draw the square which does not contain any cells which are inside the fluid (the upper right square). It is inefficient to search through all the squares in the height map, since you need to examine all neighbour squares for each cell. It would be smarter to draw the upper right square when you are about to draw the three other squares on the figure. However, it is very difficult to divide the square between the three other squares on the figure, since the division depends on whether the fluid area is concave or convex. Therefore we introduce an extra grid at which we can register the external squares which should be drawn. The grid ensures that no square is drawn twice. In order to draw the upper-right square we need to determine the distance b. b should be the "remaining thickness of the external boundary" thus b is equal to the thickness minus the shortest distance from a to the fluid surface.

The thickness of the boundary should be dependent on the amount of foam in the particular area. In areas without foam the external boundary should just be used for antialiasing thus the thickness should be small. For simplicity we use the boundary velocity q_v to determine the amount of foam thus the thickness becomes

$$\min\left(a_{\text{minThickness}} + q_v a_{\text{vfactor}}, \min\left(\Delta x, \Delta y\right)\right).$$

Chapter 8

Flying Particles

The height map has certain limitations, of which some have been pointed out in section 2.4.2. A considerable limitation is that the height map can only represent objects which have a clear boundary, thus the height map cannot model foam flying in the air.

8.1 Motivation

The primary reason why we decided to develop flying foam is that we want to obtain a better control of the viscosity and the scale of the fluid.

8.1.1 Controlling the Viscosity

The fluid simulation has been tested on a number of people and many of them say that the fluid has a high viscosity. This is a serious problem since we cannot change the viscosity of the physical model. We will now make a hypothesis why users experience a high viscosity.

In the breaking dam experiment the user observes that the fluid starts accelerating as it should. However, after a very short time the acceleration becomes zero thus the user believes that there is a high resistance. The user knows that there are two cases in the real world where such a situation occurs:

- A fluid with low viscosity moving at large-scale with high velocity on a rough bed. In this case it is the rough bed and the high velocity which have caused the high resistance.

- A fluid with high viscosity moving at small-scale with low velocity. In this case it is the high viscosity which makes the high resistance.

The user observes that the surface of the fluid is rather calm and smooth, thus it can only be the latter situation which has occured.

Therefore we will implement foam so that the user does not think the fluid has a smooth surface. If the hypothesis is true the user should believe that the fluid moves faster, and hence it has a lower viscosity.

8.2 Survey of Models

Usually particles are used to model foam flying in the air. We can divide the particle methods into two groups, those with mass and those without. The next two sections describe the two groups.

8.2.1 Particles with mass

When the particles have a mass, they affect the original height map since each time a particle is created, an amount of mass is removed from the height map. In [21] and [12] a height map is used in combination with particles with mass. The method described in [21] converts mass from the height map to foam particles when the z-velocity gets above a given threshold. We will use a similar approach in this project, however the method cannot handle waterfalls since the z-velocity is negative thus we need to extend the model.

The method in [12] is quite different since it uses the particles to model fluid with a clear boundary, which is visualized by marching cubes. The method converts the mass from the height map into particles when the speed $|\mathbf{w}|$ gets above a given threshold.

8.2.2 Massless Particles

The massless particles do not affect the fluid in the height map. A brief description of foam particles exists in [16]. In this article the particles are spawned when the fluid collides with obstacles. The particles are created in the surface of the water, and hence the velocity is equal to the velocity in the water surface.

8.3 Particle Spawning

In this project we will use particles to model the foam. First we will discuss where the foam occurs in real fluid, and then where we should spawn the foam particles.

8.3.1 Waterfalls

The amount of foam in waterfalls is very dependent on the velocity and the bed. Figure 8.1 shows the top of four different waterfalls. Image A shows a small-scale waterfall where no foam is spawned due to the low velocity and the smooth bed. Images B and C show waterfalls where foam is spawned but the boundary remains fairly clear at the top of the waterfall. At image D a large amount of foam is spawned at the top of the waterfall and hence there is no clear boundary.

It is beyond the scope of this project to develop an extensive physical analysis of the foam spawning criterion at waterfalls. Instead we will combine our intuition with figure 8.1 to obtain a relatively simple spawning criterion. The foam spawning density f_{wdensity} should be a monotonous function of the velocity $|\mathbf{w}|$, and the curvature of the fluid surface $\nabla^2 h$, therefore we have chosen the formula:

$$f_{\text{wdensity}} = a_{\text{roughness}} |\mathbf{w}|^{a_v} \left(\nabla^2 h\right)^{a_c}. \tag{8.1}$$

The extra powers a_v and a_c are used to control the influence of velocity and the curvature respectively. The constant $a_{\text{roughness}}$ is included so that we can approximate bumps which are smaller than the resolution grid. Naturally, we could have modelled this roughness in the grid, but it would require a much higher grid resolution, and hence decrease the running time.

For each time step we will spawn particles corresponding to the density from equation (8.1) at each cell. According to the volume interpretation from section 2.4.1 each cell represents an area $\Delta x \Delta y$ so during a time step Δt, we should spawn

$$\lambda_{\text{waterfall}} = a_{\text{roughness}} |\mathbf{w}|^{a_v} \left(\nabla^2 h\right)^{a_c} \Delta x \Delta y \Delta t, \tag{8.2}$$

particles on average.

It is assumed that the spawning of a particle does not influence the way particles are spawned in the future. Consequently, the spawning of the particles follows a poisson distribution with parameter λ as described in [6].

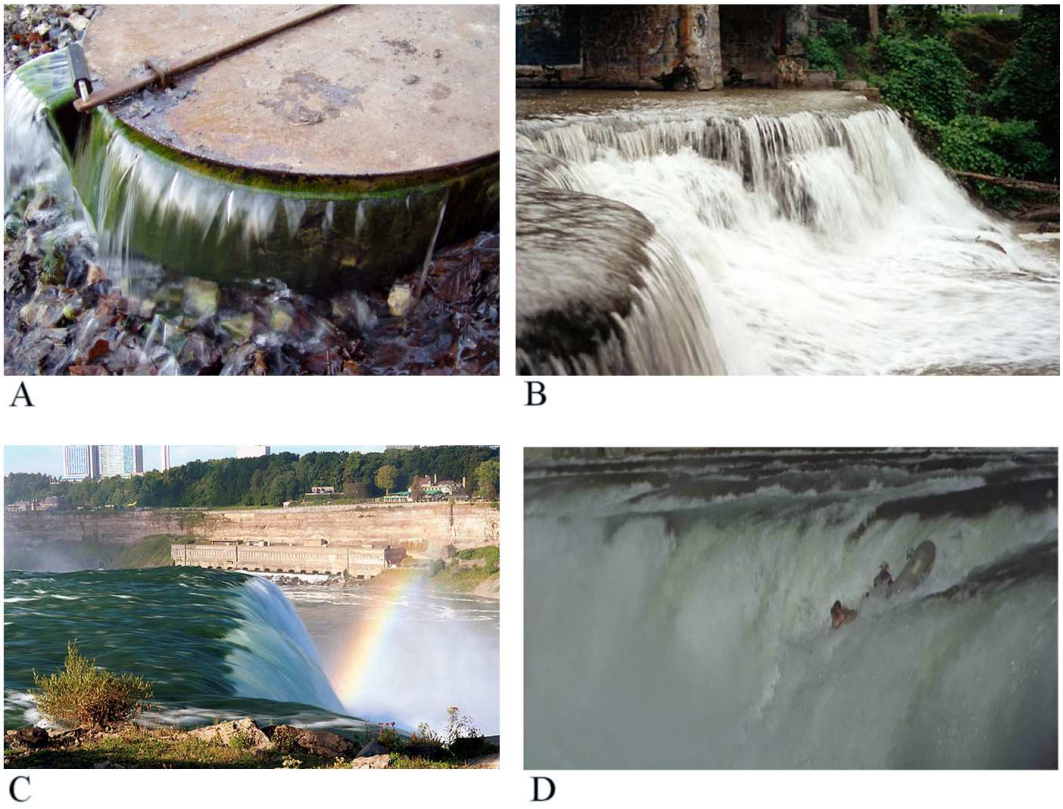

Figure 8.1: The figure shows the top of four waterfalls.

Formally, the number of particles X spawned at a cell during a time step becomes $X \in P(\lambda)$.

For each cell c_i we extract a sample X from the poisson distribution and spawn X particles at the cell. This completes the algorithm which determines the number of spawned particles per time step.

Now we should determine the initial position and velocity of the particles. Clearly, we need some random generators so that the particles can be spawned at any point in the landscape. In this section we will just determine the average position and velocity of the particles, and then we will include some random generators in section 8.4.2. The initial position is directly retrieved from the position of the fluid surface at the cell, whereas the initial velocity demands more attention.

The Initial Velocity of the Particles

The horizontal components of the particle velocity $\mathbf{w}_f$ are directly retrieved from u and v in the height map. The vertical component of the velocity is obtained by an equation which states that the velocity should have the same direction as the fluid which is floating into c_i.

Figure 8.2A shows the principle in a 1D height map where $0 < \tilde{u}_i$, thus $\mathbf{w}_f$ should be parallel to the fluid surface from h_{i-1} to h_i. At figure 8.2B $\tilde{u}_i$ is negative, thus $\mathbf{w}_f$ should be parallel to the fluid surface from h_{i+1} to h_i. In general we can use the sign $\text{sgn}(\tilde{u}_i)$ of the fluid velocity to determine which surface $\mathbf{w}_f$ should be parallel to. This means that $\mathbf{w}_f$ should be parallel to the fluid surface from $h_{i-\text{sgn}(\tilde{u}_i)}$ to h_i in general, thus formally the velocity becomes:

$$\mathbf{w}_f = \begin{bmatrix} \tilde{u}_i \\ \frac{|\tilde{u}_i|}{\Delta x}\left(h_i - h_{i-\text{sgn}(\tilde{u}_i)}\right) \end{bmatrix}. \tag{8.3}$$

In a 2D height map we apply equation (8.3) along the x- and y-axis. The particle velocity is determined as the sum of the two contributions:

$$\mathbf{w}_f = \begin{bmatrix} \tilde{u}_{i,j} \\ 0 \\ \frac{|\tilde{u}_{i,j}|}{\Delta x}\left(h_{i,j} - h_{i-\text{sgn}(\tilde{u}_{i,j}),j}\right) \end{bmatrix} + \begin{bmatrix} 0 \\ \tilde{v}_{i,j} \\ \frac{|\tilde{v}_{i,j}|}{\Delta y}\left(h_{i,j} - h_{i,j-\text{sgn}(\tilde{v}_{i,j})}\right) \end{bmatrix}, \tag{8.4}$$

where the first term comes from equation (8.3) along the x-axis whereas the second term comes from the y-axis.

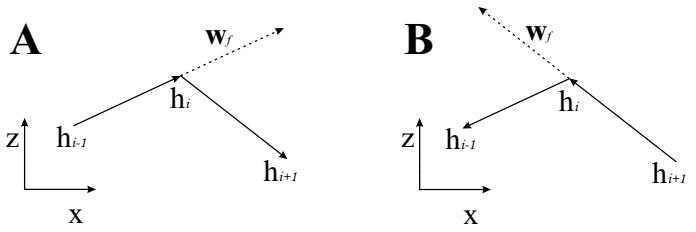

Figure 8.2: The figure shows the initial velocity of the foam particles at a waterfall. The dot-and-dash line denotes the initial velocity $\mathbf{w}_f$ of the foam. Figure A shows the situation when $0 < \tilde{u}_i$ whereas figure B shows the situation when $\tilde{u}_i < 0$.

8.3.2 Accumulation

When the fluid collides into another drop of fluid we would expect foam to occur as it is described in [21]. The fluid will accumulate at the collision point and hence the z-velocity will increase. Consequently the density of the foam becomes:

$$f_{\text{adensity}} = a_{\text{afoam}} \left(\frac{\partial h}{\partial t} \right)^{a_a}. \tag{8.5}$$

Similarly to equation (8.2) we get that

$$\lambda_{\text{accumulation}} = a_{\text{afoam}} \left(\frac{\partial h}{\partial t} \right)^{a_a} \Delta x \Delta y \Delta t, \tag{8.6}$$

particles are spawned on average when the fluid accumulates. The spawning of the particles follows the poisson distribution simlarly to the waterfall particles described in section 8.3.1.

The Initial Velocity of the Particles

The initial velocity $\mathbf{w}$ is assumed to be parallel to the normal vector of the fluid surface when we are considering a coordinate system which has the horizontal velocity of the fluid. The situation is shown in figure 8.3. Formally the initial velocity becomes

$$\mathbf{w} = \frac{\partial h}{\partial t}\mathbf{n} + \begin{bmatrix} u \\ v \\ 0 \end{bmatrix},$$

where n is the normal vector of the fluid surface.

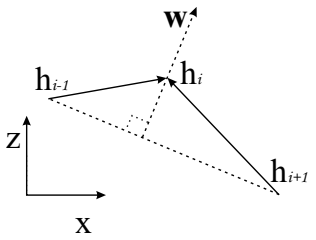

Figure 8.3: The figure shows a situation where the fluid accumulates at h_i. Consequently, foam particles are spawned with $\mathbf{w}$ as the initial velocity.

8.4 Moving the Particles

The movement of the particles is affected by various factors such as gravity, air resistance, buoyancy, the ground and several random generators. These factors can be combined in various ways, thus it seems convenient to develop a network of these factors. Each vertex in the network corresponds to a factor and the flying particles are transferred from vertex to vertex[1]. In order to archieve this, each vertex must implement the abstract class:

```
class FlyingParticleReceiver{
public:
   virtual ~FlyingParticleReceiver(){}
   virtual void receive(const FlyingParticle& fp) = 0;
};
```

Each edge in the network is a pointer to a `FlyingParticleReceiver` and the particles are transferred between the vertices by using the pure virtual function `FlyingParticleReceiver::receive`.

Figure 8.4A shows an example of a network. The vertex `ParticleContainer` contains all the flying particles which are in the visualization. For each time step all the particles inside `ParticleContainer` are sent through the loop inside the dot-and-dashed rectangle. The three vertices `AirResistance`, `Buoyancy` and `Gravity` apply forces to the flying particle, thus the resulting force becomes:

$$\mathbf{F}_{\text{res}} = \mathbf{F}_{\text{buoyancy}} + \mathbf{G} - a_{\text{air}} \frac{\mathbf{w}^3}{|\mathbf{w}|} \tag{8.7}$$

[1]In this section "vertex" is the term from graph terminology, however, in the rest of the report it refers to an OpenGL vertex.

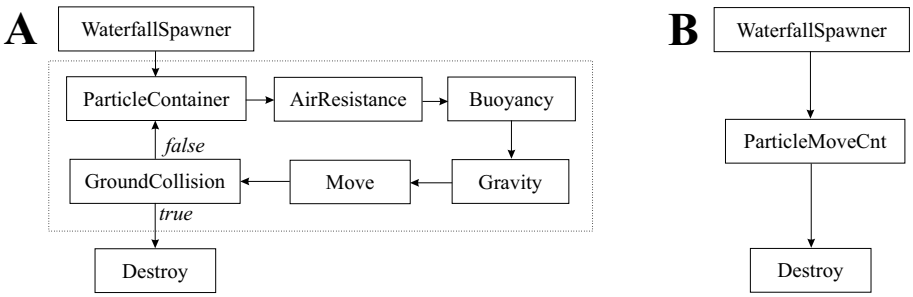

Figure 8.4: Simplifying the Flying Particle Network

where $\mathbf{F}_{buoyancy}$ is the buoyancy force, $\mathbf{G}$ is the gravity, $\mathbf{w}$ is the velocity of the particle and a_{air} is the air resistance[2]. The air resistance term builds on the assumption that the surrounding air is turbulent, however, for low velocities the air is laminar, and hence the term becomes $a_{\text{air}}\mathbf{w}$. According to [11] the max velocity of a falling raindrop causes turbulent air, thus the term in equation (8.7) is the most realistic.

After the force determination the particles arrive at the `Move` vertex where the Euler method is used to update the velocity and the position of the particles. Afterwards the particles arrive at `GroundCollision`, which determine whether the particles have collided with the ground. If the particles have collided they are destroyed, otherwise they are stored in `ParticleContainer` ready for the next time step.

8.4.1 Performance

For each time step all particles are transferred through the loop inside the dot-and-dashed rectangle. This means that for each time step the number of virtual function calls is proportional to the number of particles. It may cause a bottleneck in the simulation, thus the vertices inside the dot-and-dashed rectangle are merged into a single vertex called `ParticleMoveCnt`. In this way the number of virtual function calls is proportional to the number of spawnings in a time step, and hence it is not expected to give a measurable difference in the running time.

[2]The term a_{air} is composed by several constants as described in [11], however, we will not go into a deep discussion about resistance in this project.

8.4.2 Random Generators

The solution shown at figure 8.4B does not allow a cell to spawn multiple particles in a time step since all the particles follow the same path. Therefore it is necessary to use a random generator which affects the initial velocity of a particle. In this project a simple uniform distribution has been added to each of the coordinates in the particle velocity. Alternatively, a spherical distribution could be used. However, it has been chosen to skip the extra distribution since there are much larger errors in the simulation of the fluid.

Furtheremore, we will add a uniform distribution to the x- and y-coordinates of the particle position. In this way the particles spawn homogeneously in the landscape. The random generators for the velocity and the position are added as vertices `RandomVelocity` and `RandomPosition` respectively to the network.

8.4.3 Bouncing on the Ground

The random generators can be applied to figure 8.4B, and thereby we get a waterfall as shown on figure 8.5A. The particles are visualized by billboards parallel to the view plan as described in section 8.5.1. We observe that there is no foam at the bottom of the waterfall. This is undesirable since in a real waterfall the highest foam density appears at the bottom. Consequently, the particles should bounce when they hit the ground. Figure 8.5B shows a reflective bouncing whereas figure 8.5C shows a bouncing where the velocity of the outgoing particle is parallel to the normal of the fluid surface.

The reflective bouncing tends to develop outgoing particle velocities which are almost parallel to the fluid surface as shown on figure 8.6A, where the dot-and-dashed curve is the path of the particle whereas the other curve is the fluid surface. Therefore the particles are quickly destroyed and hence we get a poor visual realism. Naturally, the bouncing along the normal does not suffer from this problem as shown on figure 8.6B. However, we have to set a threshold of the dot product between the fluid surface normal and the incoming particle velocity to avoid a situation as shown on figure 8.6C. In other words, when the dot product gets below the threshold we do not launch an outgoing particle. Similarly, we do not launch the outgoing particle if the velocity gets below a threshold. Finally, a resistance factor is multiplied to the outgoing velocity so that we avoid infinity bouncing.

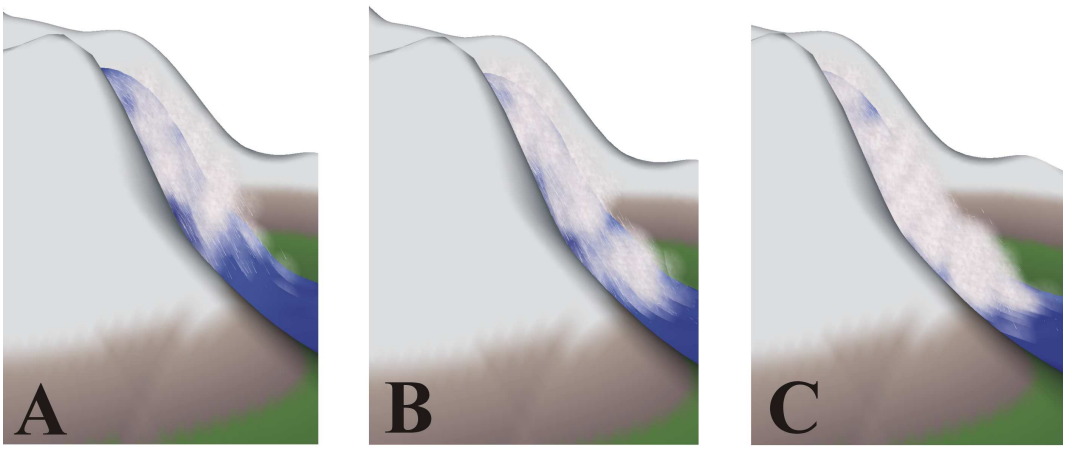

Figure 8.5: Foam bouncing comparison. Figure A does not contain any bouncing, whereas figure B and C shows reflective- and normal-bouncing respectively.

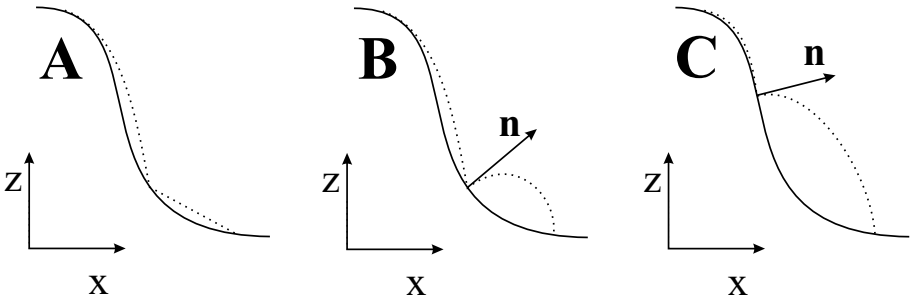

Figure 8.6: Foam Bouncing Schematic

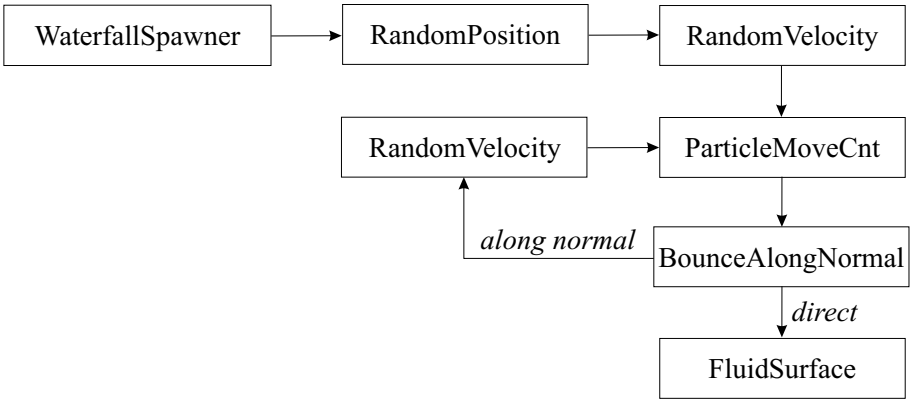

Figure 8.7: Flying Particle Network for Waterfalls

8.4.4 Setting up the Network

We have now described the most important factors in the network, and hence we are ready to set up the network. Each time a particle hits the fluid surface the particle is sent to a vertex called `FluidSurface`, which is used to spawn foam floating in the fluid surface as described in chapter 9. The network for the waterfalls is shown on figure 8.7. The edge which is labelled *direct* sends particles which are equal to the ones sent from `ParticleMoveCnt`. The network contains two random generators for the velocity. This is necessary since the particles should be more random at the bouncing than at the top of the waterfall. A similar network can be developed for `AccumulationSpawner`.

8.5 Visualization

When the flying particles are visualized it is crucial that the particles are merged with the height map so that the viewer does not experience a sharp contrast between them.

8.5.1 Primitives

The flying particles can be visualized in various ways. In this project five different primitives have been tried.

Points

The simplest way to visualize particles is obtained by using points. The points tend to give a noisy look when the density of the points is high since it is difficult to follow the movement of a single point.

Lines Parallel to the Velocity

Small drops of fluid can be visualized by lines, which are parallel to the velocity. The rear of the lines is made transparent to obtain a rough approximation of a drop.

Lines tend to provide a more clean structure than the points, since the lines display the direction of the velocity, and hence it is easier to follow the movement of a single particle.

Billboards Parallel to the view plane

Continuous foam can be visualized by *billboards*, which are quads with transparent foam textures. Each billboard represents a flying particle, thus the billboard is placed so that the particle is at the center of the billboard. Furthermore, each billboard is parallel to the view plane.

Billboards Parallel to the Velocity

The billboards can also be used to obtain a surface which might be convenient at waterfalls. The billboards should be spanned by the velocity and a vector which is orthogonal to the velocity and the z-axis. In this way the normal of the billboard can be used for lighting calculations.

Textures on the Fluid Surface

The flying particles can also be used to affect the color of the fluid surface at the (x,y) position of the particle. This is simple and robust to implement but it does not provide the spatial properties like the other primitives.

8.5.2 Discussion of the Primitives

At waterfalls the foam tends to be more cloudy, and hence the foam becomes too flat if it is rendered as a texture on the surface of the fluid. The billboards parallel to the velocity are convenient for waterfalls with low velocity,

however, they are comprehensive to implement since the billboard should be parallel to the surface of the fluid at the take-off. Instead we have chosen to use a combination of lines and billboards parallel to the view plane. The billboards removes the high contrast between the lines and the fluid.

There are still a couple of important issues which we have not dealt with. We do not sort the billboards and render them from back to front. This is expected to increase the depth feeling. Furthermore, we should render fewer lines when the camera zooms out, or else the density of the lines will change on the screen.

8.5.3 Avoiding Intersections with the Ground

A considerable problem with the billboards is that they might intersect the ground as shown on figure 8.8A and 8.8B. For instance there is a sharp contrast between the foam and the ground at the right side of figure 8.8B. Such artifacts look disturbing especially when the foam is moving, thus we should strive to avoid them.

There are various ways to avoid the intersections. The intersection could be avoided by raising the billboard vertices which are below the surface of the ground. However, this solution raises the center of the billboard, and hence the density of the foam at the boundary of the fluid decreases. Figure 8.8D shows the raised billboards, and we observe that the density of the foam is lower than the density at 8.8A. Consequently, the boundary of the fluid at figure 8.8D has a higher contrast than the boundary at figure 8.8A.

In fact, the foam at figure 8.8A also has too low density at the boundary of the fluid, since the foam below the ground is not drawn. Figure 8.9 illustrates the problem for a flying particle E. The light shaded area shows the real amount of foam which the billboard should approximate. For camera C_1 the foam will stop at line I due to the intersection with the ground. In order to approximate the light shaded shape the foam should stop at R. The problem can be solved in two ways:

- By drawing the billboard so that it passes the depth test. However, it should not pass the depth test for camera C_2, thus it is necessary to displace the depth values by using `glPolygonOffset` from [18]. The solution is shown on figure 8.8C. In this solution thin walls on the ground become transparent. Naturally, the problem could be solved by drawing the entire scene from back to front, however, this is a quite

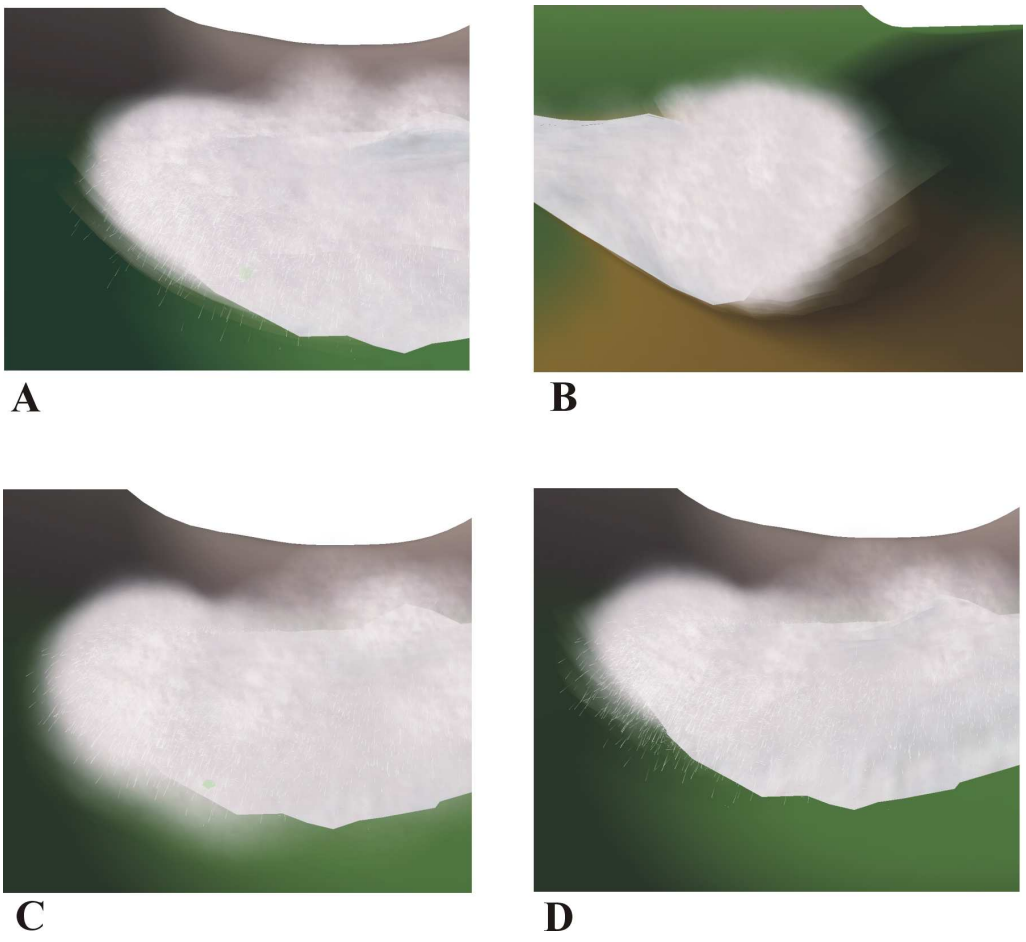

Figure 8.8: Figure A and B show the intersections between the foam and the ground. Figure C shows foam without the depth test whereas figure D shows foam which has been raised.

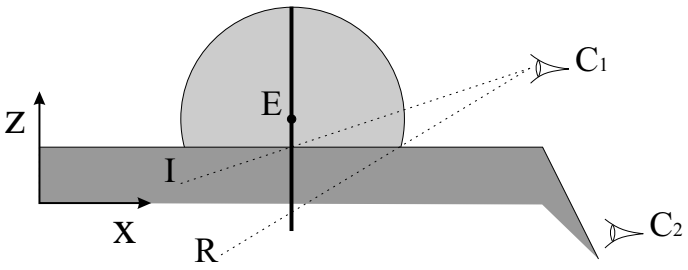

Figure 8.9: Illustration of a Foam Intersection

comprehensive solution which we will skip.

- By drawing the remaining part of the billboard on the ground. In order to draw on the ground it is necessary to add a vertex each time a cell in the height map is passed. The x-axis of the texture coordinate system is defined by the intersection between the billboard and the ground.

Determining the Alpha Value

The gauss function gives a smooth foam. However, the average value of the alpha channel is rather low, thus many blendings are needed to reach a realistic density of the foam. Instead the function $1 - x^2$ could be used to reach the same density with fewer blendings.

8.5.4 Blurring the Boundary

At the front of a boundary there is a lot of foam, and hence it is important that the boundary of the fluid is completely blurred. It is fairly computationaly expensive to ensure that the sharp boundary is completely hidden by the foam without reducing the realism of the foam. Furthermore, it is difficult to draw the billboards correctly without artifacts from the depth test. Therefore we have developed a blurred boundary as described in section 7.4. Figure 8.10A shows the fluid without blurring. We observe the contrast between the fluid and the foam which is not apparent at figure 8.10B where the boundary has been blurred.

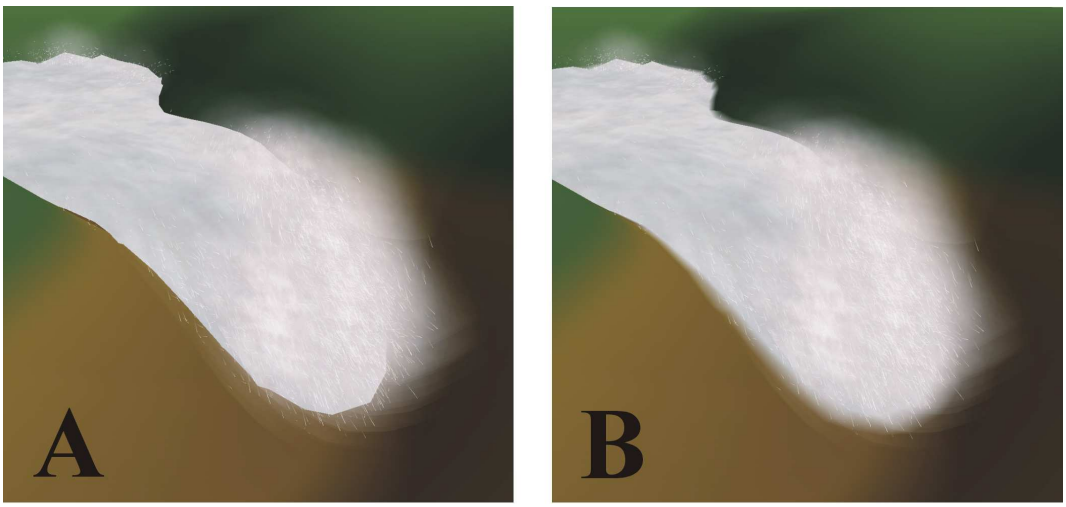

Figure 8.10: Bluring the Boundary

Chapter 9

Foam on the Fluid Surface

The flying particles clearly show the direction of the flow at the waterfalls. However, there are still large parts of the fluid surface where the direction of the flow is invisible. Furthermore, there is no foam floating away from the waterfalls, thus there is a sharp contrast between the flying particles and the surface of the fluid. In this chapter we will develop a solution to the two problems.

9.1 Survey of Methods

The foam and flow could be visualized by billboards floating on the fluid surface. However, it is difficult to place a billboard on the surface, and it is ineffecient due to the large number of vertices.

Instead we will visualize the foam and the flow by applying a texture to the fluid surface. We have not found any articles which solve both problems at the same time. Therefore the next two sections describe how the two problems have been solved separately.

9.1.1 Flow Visualization

Flow visualization is a classic problem, and hence various solutions have been implemented. The solutions we have found (see fx. [29], [19] and [3]) give a very detailed visualization of the flow. In order to obtain the detailed flow, the textures are manipulated quite a lot so it is not possible to include images of foam and fluid since they will be too distorted. Figure 9.1 shows

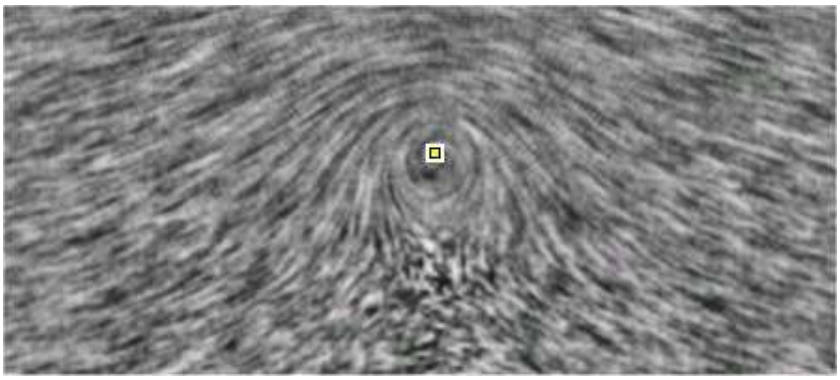

Figure 9.1: The figure shows a flow visualization from [29]. The box indicates the center of a vortex.

an example of a distorted flow visualization from [29] where it is dificult to add images which remain clear.

Instead we will use a method where the foam and fluid images are better preserved.

9.1.2 Foam Visualization

The visualization of foam at a surface is a more specific problem, thus we have only found a single article [16] on the topic. This article describes how to visualize foam in deep water by using a height map. For each cell there is a scalar $f \in [0, 1]$ which determines the amount of foam at the cell. For $f = 0$ there is no foam and for $f = 1$ there is the maximum amount of foam. In this way f is used as an alpha value of a foam texture on the fluid surface. The solution is effective, and the foam looks realistic since you can precalculate a foam texture.

In spite of this the solution cannot be directly implemented in this project, since the foam does not follow the flow in the height map. Consequently, we will extend the principle in this project.

9.2 The Basic Idea

In this section we will provide a survey of the method which is used to visualize the foam floating on the fluid surface. The foam is visualized by

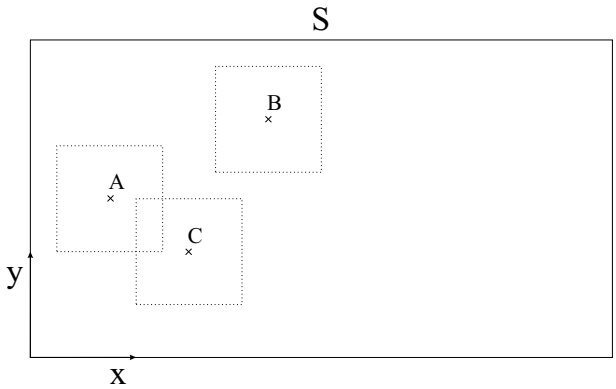

Figure 9.2: The figure shows three surface particles A, B and C, each of which is associated with texture. The textures are shown as squares with a dot-and-dash outline. The squares are placed so that the corresponding particle is at the center of the square, thus the figure shows how the textures are copied into texture S.

one large texture S covering the entire fluid surface in the height map. It is now the task to develop a method which determines S at each frame.

In order to obtain clear images of foam and fluid in S, we will use particles floating on the fluid surface. Each surface particle A is associated to a texture T_A thus basically we can obtain S by copying all the particle textures into S. Figure 9.2 shows an example where the textures from three particles A, B and C are copied into S. For each frame each particle is moved with the velocity, which is in the height map at the position of the particle.

Each time a flying particle hits the surface of the fluid an amount of foam is assigned to the textures of the nearby surface particles. If a surface particle does not receive any foam during a time step, it will lose a small amount of foam so that the foam slowly fades away.

Figure 9.3 shows a waterfall with three surface particles A, B and C at the bottom. The flying particle F will assign foam to the nearest surface particle A and B. Surface particle C is relatively far away from F so it will not receive any foam from F. Instead C will lose a small amount of foam in every frame.

Clearly, there are lots of issues which are not handled in the brief algorithm just described, thus the next sections describe improvements and deepenings to the algorithm.

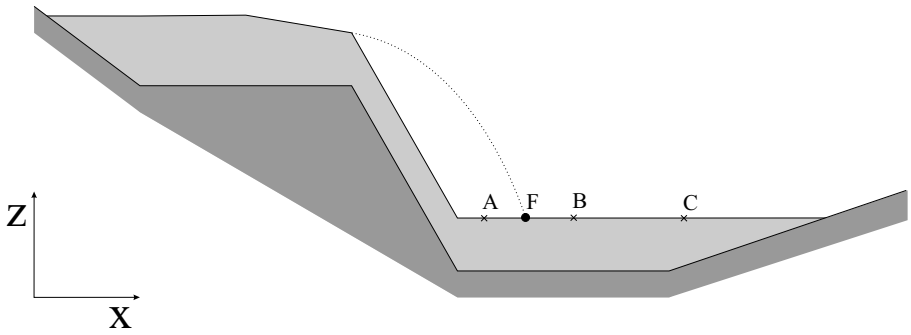

Figure 9.3: The figure shows a waterfall with surface particles at the bottom. The dot-and-dash curve shows the path of the flying particle F which has just hit the surface of the fluid. A, B and C are surface particles.

9.3 Even Distribution

It is important that the surface particles are evenly distributed in the fluid. If the particles huddle together we will copy more textures than necessary at the areas with high particle density while there might be areas in S which do not receive any texture information.

The problem will especially occur when there are sources and mouths since all the surface particles will float away from the source and huddle together at the mouths.

Consequently, we need an algorithm which removes the particles from areas with high density and inserts them at areas with low density. In order to determine the areas with low density we introduce a new 2D grid G which keeps track of the number of particles which are inside each cell in the grid. Now we can choose an interval of acceptable densities thus if the number of surface particles in a cell exceeds a_{maxsp} we will remove particles, whereas we will add particles if the count gets less than a_{minsp}. Algorithm 5 describes the principle.

9.3.1 Removing Surface Particles

In order to remove the particles from a cell C we need to get access to the particles inside C. We could search through all the surface particles in the fluid, but this operation is needed quite often thus it turns out to become a bottleneck which is increasing the overall running time. The problem is

Algorithm 5 (Adjust Closeness)

1. **for each** cell $C \in G$ **do**

   if C has more than a_{maxsp} particles **then**
   remove particles from C as described in section 9.3.1.

   else if C has less than a_{minsp} particles **then**
   add particles to C as described in section 9.3.2.

solved by extending the data structure with an auxilary 2D grid G_{search} as described in section 10.3.

The grid G_{search} is also used to find the distance to the nearest particle. This means that for a particle A we can define a function f_{dist} so that $f_{\text{dist}}(A)$ returns the distance to the nearest particle.

Each time we remove a particle from a cell we should choose the particle carefully since it is a complete waste of resources if two particles have exactly the same position. Consequently, we should remove the particle A which has the smallest value of $f_{\text{dist}}(A)$.

9.3.2 Adding Surface Particles

The objective for this method is very similar to the objective of the removal method. In this method we shall avoid that we insert a particle close to an existing particle. Therefore we will insert the particle A at a_{attempts} random positions and choose the position which has the largest value of $f_{\text{dist}}(A)$.

Note that if the first position is outside the fluid we will abandon the insertion immediately even though there might be some fluid in the cell. This is done since we do not want to waste time on the possibly large amount of cells which are outside the fluid. Remember that this method is executed every frame, thus if there is any fluid inside a cell we will soon find it with the random insertions. It could be done more effectively, but it does not cause a bottleneck since the number of cells in G is considerably smaller than the number of cells in the height map. Consequently, we will use this method since it is simple to implement.

9.3.3 Fading In and Out

Each time a surface particle A is added, the corresponding texture T_A will pop up in S. Similarly, the texture will pop away when A is removed. This looks very disturbing for the viewer thus we need to fade in and out when the particles are added and removed.

This is done by extending the data structure with an alpha value for each particle. When a particle is inserted the alpha value is initialized to zero and then the alpha value is increased by a small value every frame. Similarly, when a particle is removed it is kept in the data structure until the alpha value has been decreased to zero.

Should the fading particles be counted by the cells in G? Well, they should predict the future, thus the particles which are fading in should count and the particles which are fading out should not count. Any other choice will imply a wrong behaviour in algorithm 5.

9.4 Calculation of the Foam

In order to represent the amount of foam in a surface particle the data structure is extended so that each surface particle A contains a scalar f_A which denotes the amount of foam in the particle similar to the method from section 9.1.2.

When a flying particle hits the surface of the fluid, the nearest surface particles should receive an amount of foam. We will use G_{search} to quickly find the nearest surface particles. This means that the maximum interaction radius for a flying particle is the width of a cell from G. Our tests show that it gives a reasonable distribution to the surface particles. However, if for some reason it should be necessary with a longer interaction radius r it is necessary to have an extra grid with cell width r.

It seems reasonable that the amount of foam transferred from a flying particle to a surface particle should be a function f_{foam} of the distance between the two particles. The gauss function

$$f_{\text{gauss}}(s) = a_1 e^{-s^2} \tag{9.1}$$

has been tested with good results.

When a new particle B is added as described in section 9.3.2, f_B is initialized to a weighted sum of the amount of foam in the nearest particles.

This ends the description of the simulation of the surface particles. In the next sections we will describe how to visualize the particles.

9.5 Visualization of the Foam

We will now describe how the amount of foam f_A can be used to determine the texture T_A of a surface particle A.

The idea is that we load a number of external images and each of these images I is associated to a scalar s_I. Each image I specifies how T_A should look if $f_A = s_I$.

In general f_A does not match any of the values from the images thus we will find the two neighbouring images above and below f_A and do a linear interpolation.

More precisely T_A can be determined by finding the external image E which has the smallest value s_E so that $f_A < s_E$. Furthermore, we find the exernal image D which has the largest value s_D so that $s_D < f_A$. Now by interpreting the images as matrices we can determine

$$T_A = \frac{s_E - f_A}{s_E - s_D} D + \frac{f_A - s_D}{s_E - s_D} E$$

by linear interpolation between D and E.

In this project we have used three external foam images as shown on figure 9.4.

9.6 Smoothing the Texture Boundary

Until now we have described that the texture T_A of each surface particle A should be *copied* to S. However, this will leave a sharp edge in S at the boundary of T_A. Therefore we need to smoothen the boundary by making T_A transparent. Due to the spherical nature of the particles it makes sense to let the alpha value of a texel be function f_{sp} of the distance to the center of the texture. Again the gauss function f_{gauss} from equation (9.1) could be used.

However, the average alpha value in the texture is fairly low, thus we need many overlapping textures in S to get an acceptable density. Instead

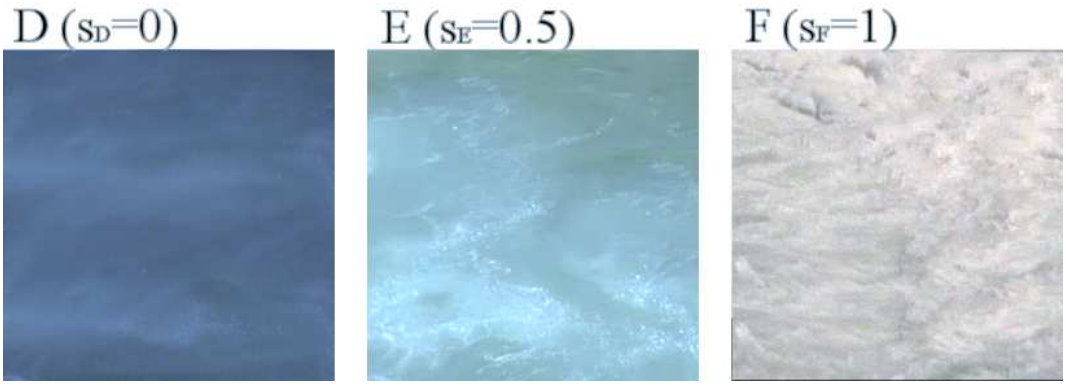

Figure 9.4: The figure shows the three external images which have been used for the surface particles in this project.

it is more effective to choose

$$f_{\text{poly}}(s) = 1 - \frac{s^2}{r^2}$$

as the alpha function since it has a higher average. The drawback of this function is that the boundary of the texture is slightly visible due to the larger "alpha slope". A proper function can be found by making a linear interpolation between f_{poly} and f_{gauss}.

9.7 Determining the Size of the Textures

The size of the textures plays an important role. In other words how much space should a surface particle occupy in world coordinates.

The disadvantage of a large texture is that the flow at the boundary of the texture might be quite different from the flow at the center. Therefore the boundary might not move with the velocity which is in the height map at the corresponding position. Figure 9.5 shows a situation with two surface particles A and B. A moves upwards whereas B moves downwards thus at position C we observe a texture which is moving upwards and a texture which is moving downwards. Naturally, this is not the flow in the height map, however the situation with two layers of fluid does occur in the real world, thus it is not a fatal problem.

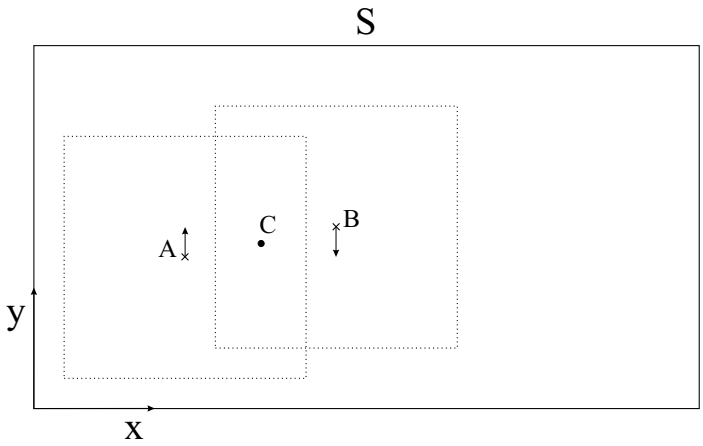

Figure 9.5: The figure shows two surface particles A and B. Due to the velocity in the height map A moves upward whereas B moves downward. Therefore the texture S will move with two different velocities at position C.

The problem with multiple layers of velocities can be reduced by using smaller textures. In this way we will get a more detailed and precise visualization of the flow in the height map. The disadvantage with this approach is that the particles cannot contain much image information, and hence S tends to become noisy and foamy even if we wish to visualize laminar fluid.

Based on experiments we have found that a side length at $20\Delta x$ of T_A gives a good balance between image quality and flow details.

Part III
Results

Chapter 10

Implementation

This chapter provides a discussion of various implementation issues which deserved extra attention.

10.1 Performance

Currently the simulation runs in real-time 20 fps in a 50x100 grid on a tbird 1.33 Ghz with a Radeon 9600 Pro graphics card. In this project the main focus has been kept on effective algorithms and improvements to the realism of the fluid. Therefore important issues such as cache performance, size in memory and branches in the inner loops have not been addressed. Consequently we expect that the running time can be improved significantly.

10.1.1 Removing the Branches

Currently there are three branches in the inner loop of the fluid simulation at each volume transfer between two neighbour cells c_i and c_{i+1}:

- **Is the velocity positive?**
  This branch is required by the imperative VEAT from section 4.1. The algorithm needs a pointer to the source cell (where the depth decreases) and a pointer to the target cell (where the depth increases). The branch can be removed by extracting the sign bit s from the velocity as described in [7] using a bitwise and[1]. Then we can retrieve the source

[1]Note that this is not a portable solution since it deals with the internal floating point representation.

and the target from c_{i+s} and c_{i+1-s} respectively.

- **Is the transfer stable?**
  This branch can be moved out of the inner loop in this way:

```
bool stable = true;
for(each cell){
   get(i).v = ...;
   stable = stable && eMath::abs(get(i).v) < maxVelocity;
}
if(!stable)
   ensureStability();
```

- **Is it a boundary behaviour cell?**
  In order to remove this branch we exploit the fact that the number of boundary behaviour cells is an order of magnitude smaller than the total number of cells. Consequently we store pointers to the boundary behaviour cells in a stack and executes these cell transfers after the original transfer loop has finished. The pointers are pushed on the stack in this way[2]:

```
int ds = applyBoundaryBehaviour ? 1 : 0; // delta stack size
stack[stackSize] = p; // p is a pointer to the cell
stackSize += ds;
```

10.1.2 Potential Bottlenecks

There are a couple of functions which are not primary bottlenecks at the moment however they might become bottlenecks if we succed to remove the current bottlenecks. Therefore we will briefly describe how to remove these bottlenecks.

Foam to Surface Particles

Each time a flying particle hits the surface we need to find the nearest surface particles. Since the number of flying particles is large you will probably not

[2]We expect that the compiler is able to remove the branch of the delta stack size by using bitwise operations.

notice any difference if it is only a small fraction $\frac{1}{a}$ of the flying particles which apply foam to the surface particles. Each of these particles should apply a times the original amount of foam to get the same amount of foam in average.

Spawning Particles

In the future we might increase the realism of the spawnings and hence increase the computational cost. Therefore it might be considerable only to update the spawning density for a small fraction of cells. Naturally we should not update the same cells every frame. Instead every cell should be updated with the same frequency.

10.2 Render to Texture

In chapter 9 we render small into one large texture which is used as texture for the entire fluid surface. There are various ways to render to a texture with the current graphic hardware. When the target texture has low resolution `glCopyTexSubImage` from [18] tends to be most effective while for high resolution it is most effective to render into a pbuffer as described in [30]. Naturally the exact resolution where the two solutions has the same running time depends on the hardware which is used. However in this project there is only a single target texture, which should cover the entire fluid surface thus we will use the pbuffer.

It was a bit problematic to find an implementation of the pbuffer which is compatible with both Nvidia and ATI. ATI has developed an implementation where the function call

```
wglGetProcAddress("wglMakeContextCurrentARB")
```

returns an error on the cards from Nvidia. Luckily the implementation from Nvidia is compatible with the cards from ATI.

10.3 Grid Searching

In chapter 9 we have used the grid from [22] to search for the particles which are inside a specified circle with radius r. The grid is defined so that the distance between two neighbour cells is equal to r as shown on figure 10.1.

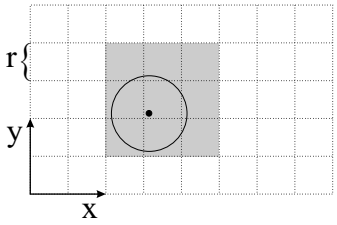

Figure 10.1: Grid Searching

In this way we can find the particles inside the circle by examining nine cells in the grid. Figure 10.1 shows an example where we find the particles inside a circle by examining the nine light shaded cells. By assuming that the particles are spread randomly in the grid the search operation has a constant running time in average.

Chapter 11

Comparison

The imperative velocity advection term works well for the breaking dam experiment, however there are problems when the bed slope varies as described in section 6.2. Anyway no other test results have been found in the literature where a true SWE simulation handles a bed slope that varies significantly compared to the depth of the fluid.

Some of the comparisons where we test the motion of the fluid are most suitable at the presentation thus they are not showed in the report.

All the tests in this chapter uses the model without the VEAT.

11.1 Comparison with other Models

Generally it has been difficult to make comparisons with the other models thus we have only provided a single comparison.

11.1.1 Comparison with Kass

Figure 11.1 shows a comparison between the model from this project with the model from [15]. Figure 11.1A shows a test from Kass where the fluid is sent from a source on a mountain. The fluid is transparent and there is no reflections from the lighting thus it is difficult to see the surface of the fluid.

The three other images show the simulation from this project where we have tried to imitate the landscape from Kass. Figure 11.1B shows the simulation with foam[1]. The foam at the waterfall works reasonably well

[1]Keep in mind that it is much more difficult to see the structure of foam on a still

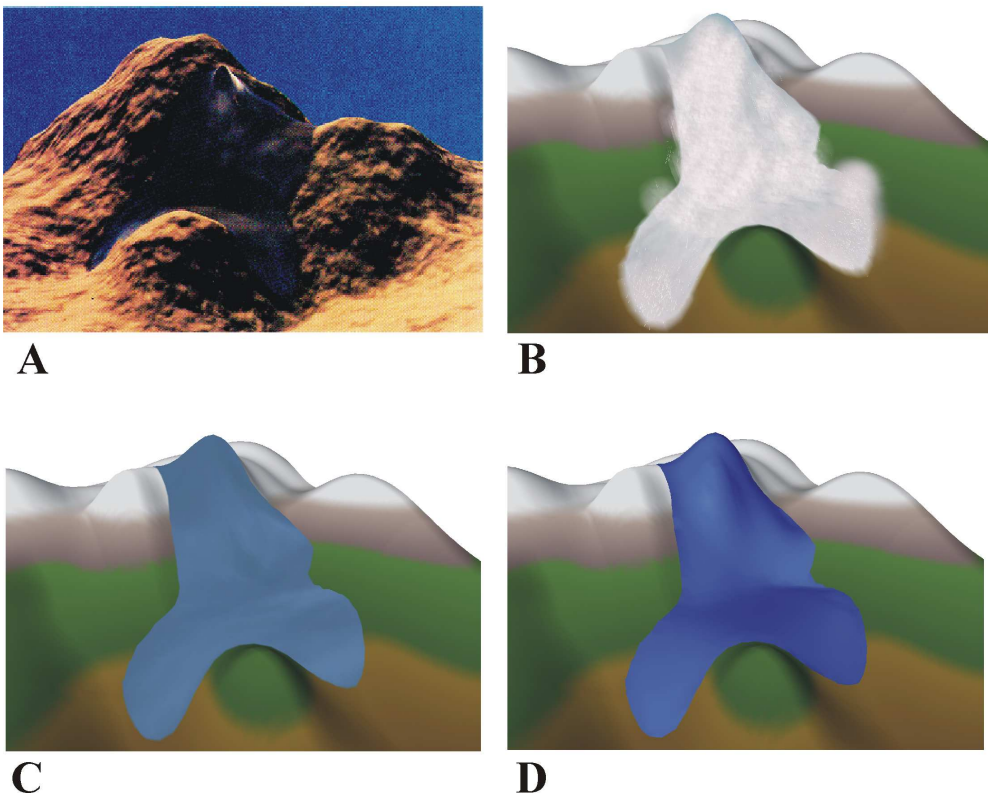

Figure 11.1: Comparison with Kass

whereas the low depth at the front of the flooding implies a lack of flying foam at the front. Figure 11.1C shows the simulation without foam thus the textures becomes equal to the images shown on figure 9.4. The last image 11.1D shows the simulation with OpenGL lighting.

11.1.2 Chens Simulation

Figure 11.2 shows the results from Chen [4]. The main focus in Chens work has been kept on boats and dams affecting the fluid neither of which have been considered in this project. The animation of the flooding has too low resolution to get a reasonable comparison with this project.

image than on in animation.

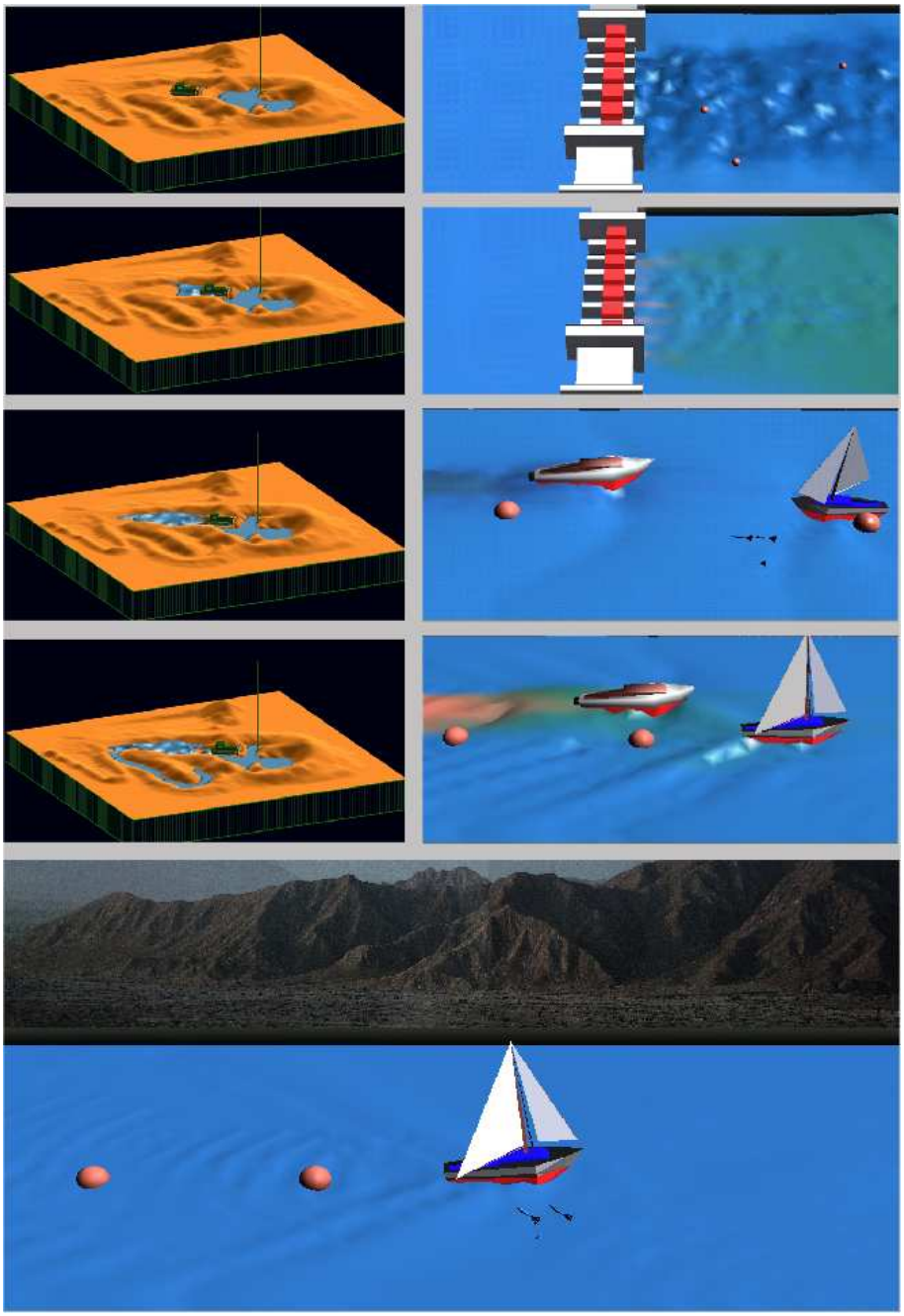

Figure 11.2: Chens Simulation

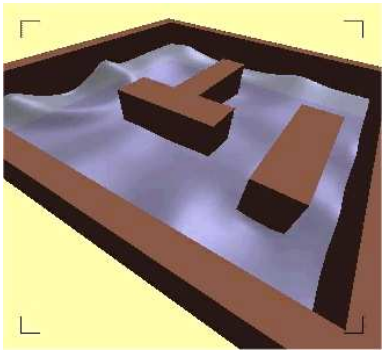

Figure 11.3: Laytons Simulation

11.1.3 Laytons Simulation

Figure 11.3 shows the results from Layton [2]. Various animations have been provided, however new waves are inserted during the animation thus it is difficult to imitate the animation.

11.2 Comparison with real Water

Figure 11.4 shows a waterfall from the movie called River Wild. The waterfall has been imitated by the fluid simulation in this project on figure 11.5. There are a number of artifacts at the imitated waterfall. The boundary of the waterfall tends to be too smooth and calm thus we need to spawn more foam at the boundary. Furthermore, the colors of the fluid does not match but it can be solved by changing the textures.

11.3 Animation

Figure 11.6 shows an animation of a waterfall and a flooding created by a source at the top of the mountain. At time $t = 9s$ there is insufficient foam at the front of the flooding. However, at time $t = 11s$ there is plenty of foam at the front of the flooding, thus amount of foam seems to be dependent on the slope of the ground. At time $t = 23s$ there is a close-up of the waterfall which looks fairly plausible. The last two images shows how a lake has been created where foam is coming from the waterfall.

Figure 11.4: Waterfall from a movie called River Wild.

Figure 11.5: Imitation of the waterfall from figure 11.4.

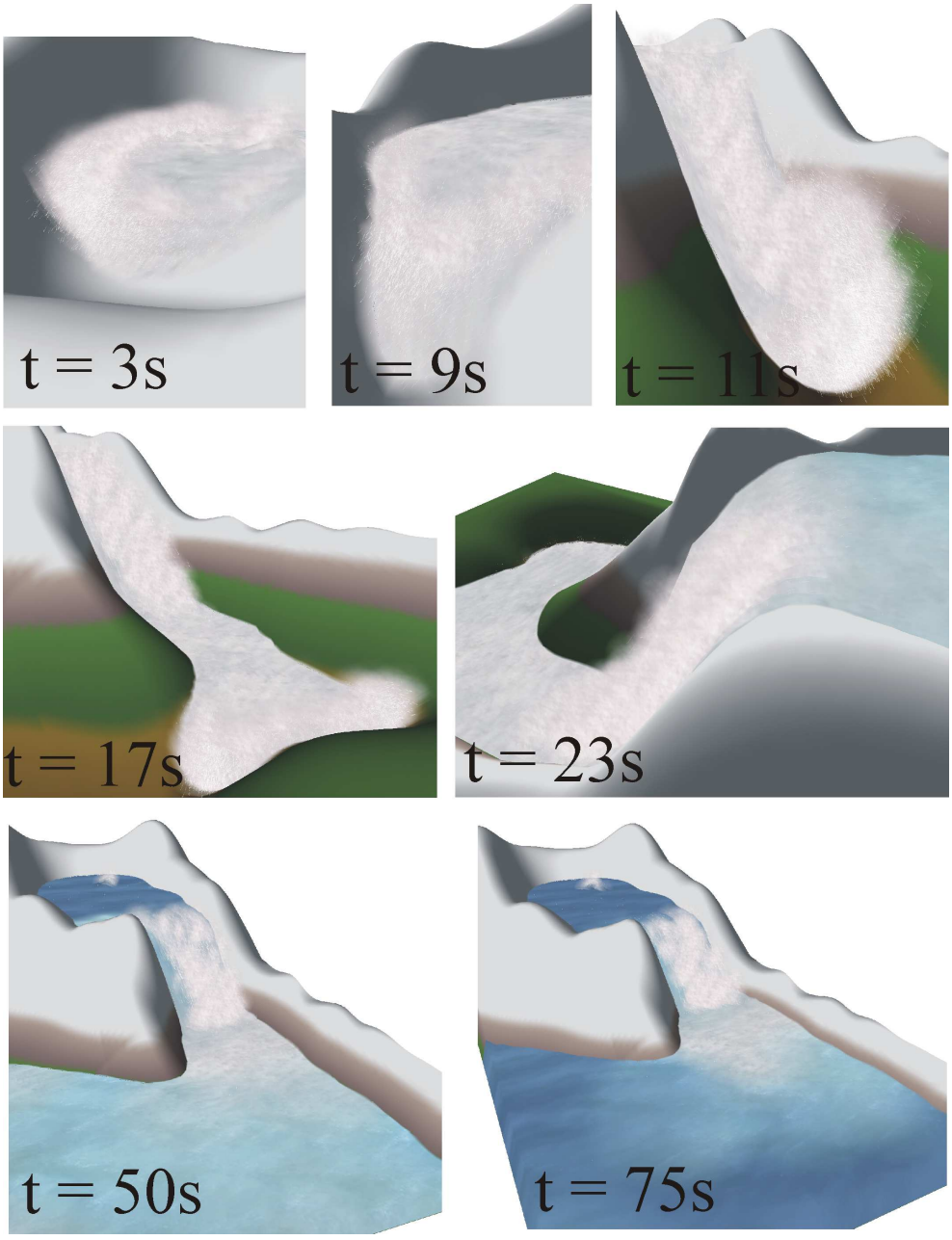

Figure 11.6: Animation

11.4 Discussion

During the tests we have found that there is insuficient foam at the front of a flooding when the fluid is flowing down a slope. The reason is that the fluid is shallow at the waterfall and since the total amount of accumulation spawned during a flooding in a cell is proportional to the depth we get less foam. We have considered alternatively spawning criterions based on the velocity of the fluid, however, two much foam is spawned if a small drop of fluid is flowing down a slope.

Furthermore the amount of foam tends to be insuficient at boundary of a waterfall thus the viewer experience a calm boundary at the waterfall. We might consider to spawn additional foam at the boundary of the waterfall due to the friction with the rough landscape.

Chapter 12

Conclusion

We have developed a new solution to the SWE using an imperative VEAT approximation. Comparisons with SPH have shown that the solution gives reasonably realistic results for the breaking dam experiment. However, there are artifacts in the bump experiment since the slope of the fluid surface is too large after the bump. It has not been possible to find a solution which includes the VEAT and solves the bump experiment. In order to handle the bump experiment we have developed an alternative solution which omits the VEAT as it has been done in [15]. The solution in [15] cannot handle waterfalls, thus an artificial resistance term has been added to our model.

The fluid tends to have a high viscosity if a simple visualization of the fluid surface is used. By extending the visualization with flying foam we have found that it is possible to convince the average viewer that the fluid has a low viscosity since it is believed to be moving with high velocity at large-scale.

The foam has a sharp contrast to the fluid surface which is not wanted, therefore a texture has been added to the fluid surface which matches the foam color. Furthermore, an extra transparent boundary layer has been added to blur the fluid boundary so that it looks like foam when it is needed.

The fluid simulation performs well in the comparisons with Kass [15] and the movie River Wild, however, the density of the flying particles is too low at the boundary of the waterfalls and at the front of floodings. It will require a more carefull analysis of the particle spawning to solve these problems.

The performance of the algorithms which are used seems to be effective. Currently the simulation runs real-time at a resolution of 50x100 cells and there are a lot of optimizations which have not been applied yet.

Appendix A

Notation

In this project a right-handed coordinate system has been used. The gravity is facing towards the negative direction of the z-axis.

Many problems can be described in one dimension thus whenever it is posible we will show 1D describtions even though it is implemented in 2D.

A.1 Symbols

$\tilde{h}_{i+1/2}$	Indicates that $h_{i+1/2}$ does not directly exists in the height map. It could be approximated as $\frac{h_i+h_{i+1}}{2}$.
$\left\langle \frac{\partial h}{\partial x} \right\rangle_{i+1/2}$	Denotes the approximation of $\frac{\partial h}{\partial x}$. E.g. the fraction could be approximated as $\frac{h_{i+1}-h_i}{\Delta x}$ by using finite difference operators.

A.2 Abbreviations

iff.	IF and only iF
NSE	Navier Stokes Equations
SLV	Single Layered Velocity
SPH	Smoothed Particle Hydrodynamics
SWE	Shallow Water Equations
VEAT	VElocity Advection Term

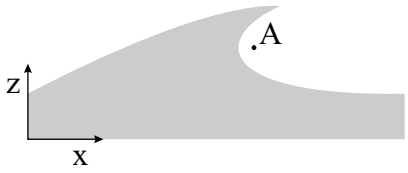

Figure A.1: An example of a breaking wave. The wave is said to be breaking since there is a point A in the air for which there is fluid above.

A.3 Terms

boundary cell	Defined in section 2.4.
breaking wave	figure A.1 shows an example of a breaking wave.
neighbour cell	Each cell $c_{i,j}$ has four neighbour cells $c_{i-1,j},\ c_{i+1,j},\ c_{i,j-1},\ c_{i,j+1}$.

A.4 Variables and Constants

Here is the most frequently used symbols used in the report:

a	a constant
b	height of the bed (ground height)
c	a cell
d	depth of the fluid
$\mathbf{F}$	a force
g	gravity scalar
$\mathbf{g}$	gravity vector along the negative z-axis
h	height of the fluid surface
i	integer index along the x-axis
j	extra integer index
k	extra integer index
p	pressure
r	the time that has elapsed since $d = 0$
t	time
u	velocity along the x-axis
v	velocity along the y-axis
U	discharge along the x-axis
V	discharge along the y-axis
$\mathbf{w}$	velocity vector (the dimension depends on the situation)

Appendix B

Accessories

The enclosed CD-ROM contains the source code and a compiled version of the simulation program in the bin directory[1]. You can start the program called "flood3d.exe". When the program has finished loading, a demo will run automatically in approximately 2 minutes.

The homepage `www.student.dtu.dk/~s973526` will contain updates of the simulation program.

[1]The program requires a windows operating system, and it is recommended at least to use a tbird 1 ghz cpu with a geforce 2 graphics card or correspondingly from Intel or ATI.

Bibliography

[1] A. Zanni A. Valiani, V. Caleffi. Finite volume scheme for 2d shallow-water equations application to a flood event in the toce river. CADAM, 1999.

[2] Michiel van de Panne Anita T. Layton. A numerically efficient and stable algorithm for animating water waves. *the Visual Computer*, 18(1):41–53, 2002.

[3] Gordon Erlebacher Bruno Jobard and M.Ỹousuff Hussaini. Lagrangian-eulerian advection of noise and dye textures for unsteady flow visualization. *Transaction on Visualization and Computer Graphics*, 8(3), 2002.

[4] Jim X. Chen. *Physically-based modeling and real-time simulation of fluids.* PhD thesis, University of Central Florida, 1995.

[5] S. Chippada, C. Dawson, M. Mart'inez, and M. Wheeler. A godunov-type finite volume method for the system of shallow water equations, 1997.

[6] Knut Conradsen. *En Introduktion til Statistik.* seventh edition, 1999.

[7] Mark Deloura. *Game Programming Gems 2.* Charles River Media, Hingham, Massachusetts, first edition, 2001.

[8] Nick Foster and Ronald Fedkiw. Practical animation of liquids. *Proceedings of the 2001 conference on Computer Graphics*, pages 23–30, 2001.

[9] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[10] J.-F. Gerbeau and B. Perthame. Derivation of viscous saint-venant system for laminar shallow water; numerical validation. *Discrete and Continuous Dynamical Systems-Series B*, 1(1):89–102, 2001.

[11] Erik Both Gunnar Christiansen and Preben Østergaard Srensen. *Mekanik*. Institut for Fysik, Building 307, Technical University of Denmark, second edition, 1997.

[12] Mark Hall. Combining particles and waves for fluid animation, 1992.

[13] R. Harpin J. Qin I. D. Cluckie, R. J. Griffith and J. M. Wicks. Forecasting extreme water levels in estuaries for flood warning. *R&D Project Record*, 2000.

[14] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge, Cambridge, United Kingdom, first edition, 1996.

[15] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57. ACM Press, 1990.

[16] Robert Golias Lasse Staff Jensen. Deep-water animation and rendering. *Gamasutra*, 26 2001.

[17] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82(12):1013–1024, 1977.

[18] Tom Davis Mason Woo, Jackie Neider and Dave Shreiner. *OpenGL Programming Guide*. Addison Wesley, Reading, Massachusetts, third edition, 1999.

[19] Nelson Max and Barry Becker. Flow visualization using moving textures. *Proceedings of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, 1995. Livermore, California.

[20] J. J. Monaghan. Simulating free-surface flows with sph. *Journal of Computational Physics*, 110:399–406, 1994.

[21] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *Computer Animation '95*, pages 198–205, 1995.

[22] Per Slotsbo, Thomas Krog. Forbedret vandsimulation med sph, 2002.

[23] Allan J. Acosta Rolf H. Sabersky and Edward G. Hauptmann. *Fluid Flow - A First Course in Fluid Mechanics*. The Macmillan Company, New York, second edition, 1971.

[24] Dirk Schwanenberg and J. Köngeter. A discontinuous galerkin method for the shallow water equations with source terms. *Discontinuous Galerkin Methods*, pages 419–424, 2000.

[25] Jos Stam. Real-time fluid dynamics for games. *Proceedings of the Game Developer Conference*, 2003.

[26] IB A. Svendsen and Ivar G. Jonsson. *Hydrodynamics of Coastal Regions*. Den Private Ingeniœrfond, Lyngby, Denmark, first edition, 1976.

[27] Eleuterio F. Toro. Riemann problems and the waf method for solving the two-dimensional shallow water equations. *Philosophical Transactions*, 338:43–68, 1992.

[28] Eleuterio F. Toro. *Shock-Capturing Methods for Free-Surface Shallow Flows*. John Wiley & Sons, Ltd, Chichester, England, first edition, 2001.

[29] Jarke van Wijk. Image based flow visualization. *SIGGRAPH*, 2002.

[30] Chris Wynn. Opengl render-to-texture. *Nvidia*, 2002.