

La Trobe University
Bundoora, Victoria, Australia
Faculty of Science, Technology & Engineering
Department of Computer Science and Computer Engineering



**Reducing Ineffectual Computations in Water
Simulations**

Author: Jason Thompson
Student No: 15310338
Course: B. Comp. Sci. in Games Tech.
Submission Date: 26th of October 2009
Supervisors: Dr. John R Rankin

Abstract

This thesis explores the optimisation of water simulation through the reduction of ineffectual computations. This is achieved by breaking a terrain into regions then selectively applying an existing simulation technique based on the state of those regions. Test results indicate that this approach is viable with certain scenarios showing a decrease in computations with comparable results to the existing technique being managed. However, this method still needs to be further refined before it could be considered a robust addition to the current simulation approaches.

Acknowledgments

I would like to acknowledge Dr. John R Rankin and Dr. Eric Pardede for their support in compiling this thesis. In addition, the Computer Science teaching staff for their quality instruction over the past four years. Also, my parents for their support throughout this process.

Contents

Abstract	i
Acknowledgments	ii
List of Tables	v
List of Figures	vii
1 Introduction	1
1.1 The challenges in Real-time fluid simulation	1
1.2 Organization	2
2 Previous Work	4
2.1 Water Simulation Approaches	4
2.1.1 Procedural Methods	5
2.1.2 Height-field Methods	6
2.1.3 Eulerian Grid Methods	8
2.1.4 Particle Systems	9
2.2 Large Scale Water Simulation	11
2.2.1 Dynamic creation of large water systems	11
2.2.2 Generation of Surface Detail	13

2.2.3	Efficient Display of Large Water Systems.	15
2.2.4	Splashes	17
2.3	Summary	18
3	Problem Description	20
3.1	Aim	20
3.2	Selected Approach	21
3.3	Selected Physical Volume Model	23
4	Techniques and Methods	26
4.1	The Physical Volume Model	26
4.1.1	The structure of the model	27
4.1.2	Particle System and Sources	31
4.2	The Physical Volume Model Manager	33
4.2.1	Regions	36
4.2.2	Water Fall Region	39
4.3	Testing Method	40
5	Results	42
5.1	The Trivial Case	42
5.2	The Downhill Case	43
5.3	The Rough Case	46
5.4	The Multiple Downhill Case	47
5.5	The Step Case	48
6	Conclusion	50
	References	51

List of Tables

4.1	Pooling Region Behavior	38
4.2	Flowing Region Behavior	38

List of Figures

2.1	A conceptual view of creating an ocean surface from a random spectrum.	5
2.2	Two and three dimension height-fields	7
2.3	Example of an Eulerian grid based method	8
2.4	Interacting and non-interacting particles	9
2.5	An example of the results produced by Mitchell [14]	15
2.6	The projected grid used by [8]	16
2.7	Splashes and foam in a simulation.	17
3.1	A conceptual view of the proposed system.	22
4.1	The height-field model	27
4.2	An overview of the region model	34
4.3	The inflows and outflows of a region	34
4.4	The classification of columns	37
4.5	The different types of waterfalls	39
5.1	A view of the trivial case	43
5.2	Results of the trivial case	43
5.3	A view of the downhill case	44
5.4	Unsucessfull results of the downhill case	44

5.5	Successful results of the downhill case	45
5.6	Successful results of the downhill case	45
5.7	A view of the rough case	46
5.8	Results of the rough case	47
5.9	Example of the cumulative error of stabilisation	48
5.10	A view of the multiple downhill case	48
5.11	A view of the steep case	49
5.12	Results of the steep case	49

Chapter 1

Introduction

There are many reasons to study fluid simulations today. First and foremost are the challenges to real-time fluid simulations not only do you have to reproduce a complicated phenomena but it must be accomplished with limited resources. Exciting new technologies like multi-core CPUs and general purpose computations on a graphics processing units (GPGPU) have enabled more detailed simulations to be run at real-time rates without any improvement of the software. Finally, as the Games Industry consumes a lot of different physics technology, there is a commercial need for new and exciting physics algorithms.

1.1 The challenges in Real-time fluid simulation

Real-time fluid simulations have some specific characteristics that distinguish them from their off-line counter-parts. Bridson et al. [2] proposes four characteristics that real-time fluid simulations should have:

- Cheap to compute: Low computational complexity. Real-time applications run at 40-60 fps and in real-time applications a fluid simulation may only get a small fraction of the time to render a frame.

- Low memory consumption: Efficient memory usage. Real-time simulations targeted at personal computers and gaming consoles only have limited amounts of memory to use.
- Stability: The ability to handle different time steps and arbitrary physical quantities. This is especially important in Games where characters can move at arbitrarily high speeds and the simulation must be able to produce sensible results. Common examples of instability are dissipation of physical quantities like force or volume; infinite increases in physical quantities for example water increasing in volume when there is no reasons for it do so; and oscillations of a physical quantity to oscillate around some value.
- Plausibility: Is perhaps the most important, but most difficult to define. It refers to the results of simulation being visually plausible as opposed to being realistic.

1.2 Organization

The thesis begins with a review of Previous Work on water simulation for Computer Graphics. Followed by a description of the general approaches that are used to simulate water along with their properties, drawbacks and benefits. Another area of focus in this section is review of the issues of representing large bodies of water like lakes, rivers and oceans including the general approaches that have been composed and specialised to solve these issues.

In the chapter 3 a number of problems associated with current techniques used for large bodies of water are identified and a selection of these problems will be made. Those problems will be stated precisely and will form the aims of the thesis. A general approach to solving these aims is developed through the analysis of how large bodies have been simulated, the general methods of simulation and evaluating possible solutions with the characteristics of a real-time simulation. Finally a general approach will be selected.

The Techniques and Methods chapter, 4, defines a specific solution to the aims identified in chapter 3. The section concludes with a description of the testing method that is used to determine how well the solution meets the aims.

Chapter 5, details the outcome of the tests. Each test result is followed by a discussion analysing whether the results supported or deviated from the expected results.

Finally the chapter 6 presents the findings of the thesis, the strengths, the weaknesses and the future directions that this work may take.

Chapter 2

Previous Work

This chapter presents an overview of how water is simulated in computer graphics. The first section gives a brief introduction to the basic approaches to fluid simulation. The second section explores some of the issues involved with the simulation of large bodies of water and how they have been solved.

2.1 Water Simulation Approaches

The study of fluids has a long history dating back to time Archimedes of Syracuse who explained the concept of buoyancy. Through to the work Claude-Louis Navier and George Gabriel Stokes in the 1800's which resulted in the Navier-Stokes equations. It makes sense that this fascination with water would transfer to Computer Science.

In the mid 1980's the computer graphics community was introduced to fluids with Fournier and Reeves[4] and Peachey[19] who were able to display procedurally generated waves crashing on a beach. Since then these areas have been further developed by a number of people. In 1990 Kass and Miller [9] developed an interactive simulation using a simplified form of the Navier-Stokes equations, the shallow water equations. The application was able to show in real-time wave propagation on a free surface. In 1989 Miller and Pearce[13] presented an early method which used particles or "Globules" to

simulate powders and liquids. In 2003 Stam[21] used a semi-Lagrangian algorithm to simulate a two-dimensional velocity field in real-time. The above instances represent seminal work in procedural methods, height-field methods, Eulerian grid based methods and particle systems for fluid simulation. This section reviews these different methods as they are applied in more recent terms as well as explaining some of the limitations and advantages of each method.

2.1.1 Procedural Methods

Procedural methods do not attempt to do any form of calculations to generate a fluid. Instead they “simulate the effect, not the cause” Muller[16]. The effect of a physical phenomenon is generated using variables that describe the different properties of the effect. For example, in the case of an ocean simulation, properties like the choppiness of water, the depth of the water and the wind conditions might be used in generating a procedural oceans cape. Tessendorf [23] provides an overview of two of the main procedural methods: “Gerstner Waves” and “Statistical Ocean Sampling”. “Gerstner Waves” aim to simulate an ocean scene by summing two-dimensional sinusoids while “Statistical Ocean Sampling” uses the Inverse Fourier Transform to synthesize an ocean surface from a procedurally generated frequency spectrum (see figure 2.1). These techniques are used in real-time applications [14]

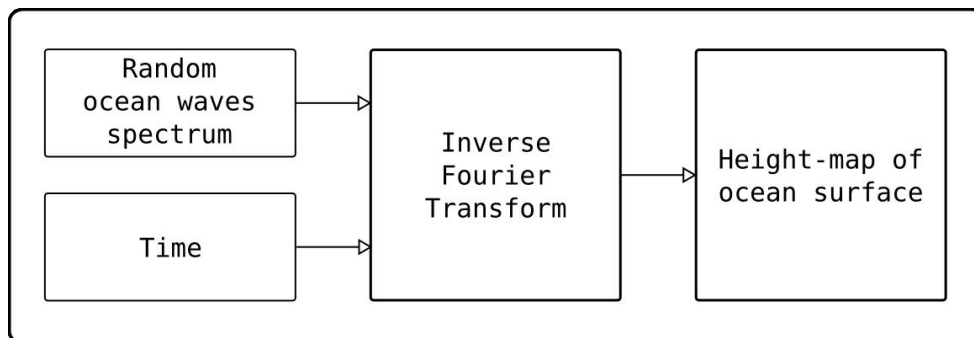


Figure 2.1: A conceptual view of creating an ocean surface from a random spectrum.

and in off-line applications as in movies like *The Titanic* because of their low memory usage, low processing time, relatively high quality output and ability to represent very large volumes of water. The fundamental limitation with the above methods is that they are not interactive, that is there is no inherent way to directly interact with the waves being generated. In response to this limitation, textures have been used to add disturbances to surfaces for instance a boat wakes[14].

Procedural methods are not restricted to generating height-fields and have also been applied to the generation of velocity fields. In [1] a turbulent velocity field is generated by modulating Perlin noise. Unlike the above procedural methods, this method, although aimed at a smaller spatial domain, allows objects to interact with the result. For example, a demonstration shows a circular obstruction forcing water to flow around it. This form of procedural method does not directly generate details like ripples. However, it can be used in conjunction with other techniques to generate the appearance of ripples flowing in a stream. Yu et al. [25] use a procedurally generated velocity field to advect particles with animated wave textures. The textures of the particles are combined to create a bump map representing the ripples on a surface.

2.1.2 Height-field Methods

Height-field methods can be characterised by the assumption that a body of water can be represented by columns of water. There are some inherent limitations of this assumption some of which are that water can not overlap itself; so, no breaking waves. Water cannot separate from itself without the use of a particle system e.g. no splashes or spray. Finally, water must always be connected to the terrain which eliminates cases like waterfalls where water spills downwards towards the terrain. That being said, height-field methods offer strong performance and interactive results.

One of the earliest Computer Graphics papers which modeled water as a height-field was Kass and Miller [9] which used a simplification of the Navier-

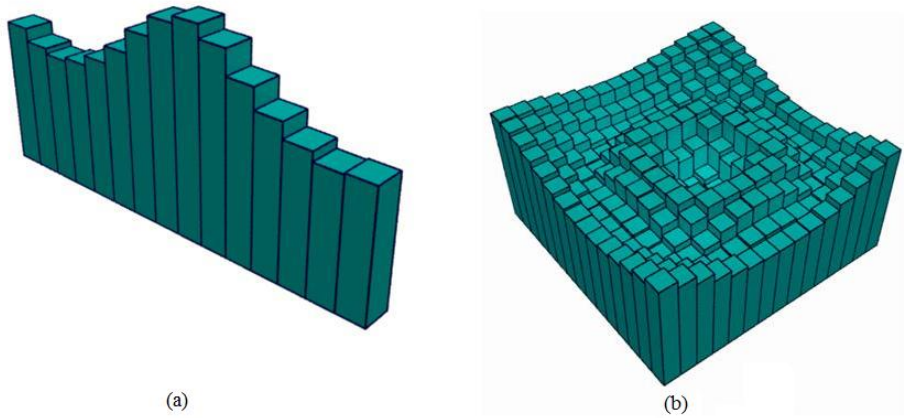


Figure 2.2: Two (a) and three (b) dimensional height-fields.

Stokes equations to determine the flow of water between water columns. O'Brien et al. [18] took a different approach and used the concept of hydrostatic pressure. 'virtual Pipes' are constructed to and from adjacent water columns. Using the hydrostatic pressure between the 'virtual pipes', the flow of water between columns is calculated. [18] was also able to simulate two-way rigid body coupling, through the use of a separate surface sub-system. This allowed rigid bodies to apply forces on the free surface of the water and for the water to apply forces on the rigid body. Both approaches have been extended recently[11][24].

Maes et al. 2006 [11] used the hydrostatic pressure model and extended it to simulate water flowing over an irregular terrain. However, this method was unable to produce real-time frame rates for large spatial domains. Wang et al.[24] used a more general form of shallow water equations to simulate small scale effects like surface tension on small drops of water as well as water waves in larger bodies of water. The simulation ran at real-time rates with the actual frame rate varying depending on parameters like resolution and the surface on which the fluid was located.

2.1.3 Eulerian Grid Methods

Eulerian grid based methods are characterised by the assumption that a fluid can be represented by a vector field. Usually this assumption results in a two-dimensional or three-dimensional grid with velocity, forces and pressures defined at discrete points. It has been used extensively in scientific and engineering domains; however, because of its high memory and processing requirements it has not been used as much in real-time applications. That being said, Stam [21] presents a Semi-Lagrangian two-dimensional algorithm for the simulation of fluids in real-time. Though the algorithm has Lagrangian aspects, the result of the algorithm is still Eulerian, a velocity field. The algorithm convects a density field through an evolving velocity field. The demonstration application has the appearance of smoke moving in air. One of the main problems with using grid structures to simulate fluids

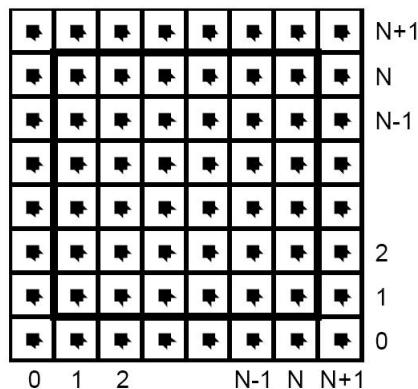


Figure 2.3: Stam [21] places velocity and density at the centre of the cell.

is that the grid structure must cover the entire spatial domain where the fluid is going to appear. If a river is going to be simulated, it would not be efficient or computationally feasible to have the entire river covered in a three-dimensional grid. One way to make this more usable is to simplify the equations from three-dimensions down to two-dimensions as was done by Chen et al.[3]. Essentially the fluid is simulated on a two-dimensional surface and the third dimension is determined from the properties on the

two-dimensional surface, in this case pressure determines height. However, this begins to move the simulation into a height-field representation.

2.1.4 Particle Systems

Particles systems are a flexible tool for simulating physical phenomena. They were introduced by Reeves [20] in 1983 for use in the motion picture *Star Trek II: The Wrath of Khan*. Since then, particles have been used in many different areas, more relevantly in fluid simulation. There are two main types of particle system used in fluid simulation: non-interacting particles where particles do not interact with each and interacting particles where particles interact with each other. A number of papers [18][11][12] (see figure 2.4 (c)) use non-interacting particles to model when water disconnects from itself, for example when water splashes. Mass conserving particles would be emitted when the upward velocity of the water reached a critical amount. On returning to the water they would be reabsorbed. Particles used in this fashion have the advantage of being simple and efficient.

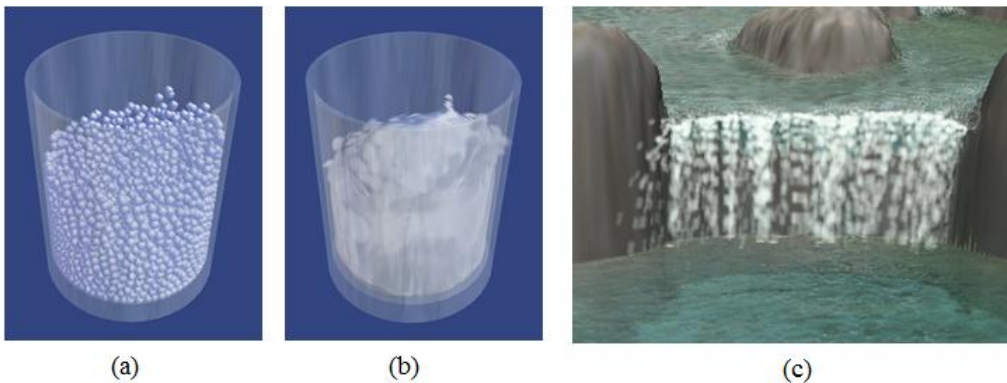


Figure 2.4: (a) Müller et al. [17] SPH fluid simulation (b) with a rendered surface. Maes et al. [12] non-interacting particles (c).

While simple and efficient, non-interacting particles can only represent a limited number of phenomena such as spray, splashes or waterfalls. On the other hand, interacting particles can be used to represent more general cases

like water being poured into a cup. This generality comes with a cost, interacting particle generally require a lot of computations to determine particle-particle interactions. One of the earliest computer graphics techniques to model particle to particle interactions was put forward by Miller and Pearce [13]. Each particle applies attractive, repulsive and drag forces on every other particle within a defined distance. This approach was able to model a number of types of substances like powders, fluids and solids. The main limitation of this method is that it does not appear to be based on an experimentally determined model. So it is not a simple task to model substances from physical properties; instead, an ad hoc tuning process would have to be used to find appropriate values for the different variables. This method was determined in 1989 and because of the limited computing power of the day the results were not real-time, however, today's hardware may be able to produce faster results.

Müller et al. [17] put forward a real-time physically based approach to simulating interacting particles (see figure 2.4 (a)). Using concepts borrowed from Smoothed Particle Hydrodynamics (SPH) and fluid dynamics. Fluid properties are defined at discrete points, particles, and interpolated into field quantities using smoothing kernels. At a given point in space, a smoothing kernel will sum up the contribution of all the particles in the system. The weighting of the contribution of a particle is determined by the distance of the point to the particle. The Navier-Stokes equations were adapted to use smoothing kernels instead of fields. This method produced visually acceptable results.

One of the main bottlenecks for the above particle systems is the extraction of a visually pleasing renderable surface. Two methods are used for surface extraction: Point 'splatting' and Iso surfaces. Point splatting renders point sprites to show elements of fluid. This method is the fastest running at 20 frames per second for 2200 particles(see figure 2.4 (b)). Iso-surfaces construct a mesh from the particles which is then rendered. This method requires a longer computation time running at 5 fps with 2200 particles, arguably with a greater visual quality.

2.2 Large Scale Water Simulation

The focus of this section is how large bodies of water have been simulated and what issues and challenges arise. The goal of this section is to present a thorough picture of how past research have met these challenges and how successful they were. The following general areas will be considered.

- **Dynamic creation of large water systems.** The ways in which water has been made to flow over large arbitrary terrains or surfaces.
- **Generation of Surface Detail.** How features like waves and ripples have been simulated.
- **Efficient Display of large water systems.** The ways in which the display of large water systems have been made more efficient.
- **Splashes** how waterfalls and similar features have been simulated.

2.2.1 Dynamic creation of large water systems

Most systems that simulate the dynamic creation of water systems have a similar high level view. The system requires an arbitrary terrain, sources and sinks of water. The simulation then solves the position of water over finite time steps. What makes this area so challenging is that real-time performance must be maintained even though there is a large spatial domain and large amounts of water. One factor which affects performance is scalability. Scalability expresses the relationship between the computational domain, where the simulation must be performed, and some property of the simulation like the size of the spatial domain or the amount of water. If a simulation is not scalable, the computational domain will grow very quickly. This will result in more computations every time step which negatively affects real-time performance.

Kipfer et al, [10] uses SPH to simulate the water and as a result the computational domain is directly related to the amount of water in the simulation. Chen et al. [3] uses two systems one to track the footprint of water

and another to perform a height-field simulation on the footprint. The computational domain is related to the surface area of the water. Maes et al. [12] uses a height-field simulation over the entire spatial domain, but does not perform the full simulation on areas without water. The computational domain is related to the entire spatial domain, but the cost of calculation varies based on the presence of water. Using a height-field approach adapted to a GPU they were able to simulate water moving on irregular 200mx200mx50m terrain at 20 fps.

Kipfer's et al. algorithm can produce reasonable results in terms of performance and visual appearance it still has some drawbacks. One such drawback is that the quality of this method becomes questionable on a very large spatial domain. The system can only handle a certain number of particles at usable rates as evidenced by the drop in frame rate as the number of particles increase. If the water were to flow over a large area the density of particles would be very low because the total number of particles is constant. As the particle density decrease so to does the dynamism of the water, e.g. the ability to produce splashes or fill basins.

As a way to improve the coverage of particles on large terrains whilst keeping the number of particles constant, different sized particles were proposed for different sized spatial domains. That is, larger particles for larger terrains. However, this affects the visual quality of the simulation. Consider a large terrain with large particles from an ideal viewing distance, the simulation produces acceptable results. As the viewing distance decreases the particle size remains constant which causes the visual quality of the simulation to decrease i.e. the water appears 'blobby'.

While the algorithm does have some drawbacks in terms of quality, it has some positive aspects. The most obvious is the flexibility of the system. There is no constraining grid structure within which the water must flow. The water is able to go anywhere on a terrain without affecting performance or needing extra memory to define grids. Another important feature is that the water has the inherent ability to exhibit a wide range of behaviors like splashes, pooling of water in basins and waterfalls without any extra sub-

systems or performance costs.

Chen et al. take a different approach where instead of using particles the volume of water is stored in regions of the terrain. Two main algorithms are presented in their paper. One is concerned with simulating the surface of the water with a two-dimensional form of the Navier-Stokes equations and the second algorithm focuses on tracking the flow of water over an arbitrary terrain.

This paper is somewhat old and the concept appears not to have been extended by other researchers. The model does lack some generality: first, the model does not support features like waterfalls that form because of steep slopes. Second, the model only calculates the footprint of the water. The velocity field resulting from water flowing over the terrain is not calculated. That being said, the model is still fundamentally very flexible. The model can simulate water moving over a dynamic terrain, the terrain may change over time.

Each of these papers is in some way able to show the movement of water over a relatively large spatial domain. Techniques used by Maes et al. and Kipfer et al. couple the propagation of water system with the generation of surface details like ripples. Whereas, Chen et al. use two separate sub-systems: One for the propagation of water and the other for the generation of surface details.

2.2.2 Generation of Surface Detail

Surface detail is an important part of simulating large bodies of water. It adds texture to the visual appearance with features like ripples, swells and waves. It also has the important role of providing information on the type and state of the water. For example, to indicate that a section of water belongs to a stormy ocean the surface details will be large choppy waves. The goal of surface details creates some challenges: how to produce meaningful surface details and how to maintain real-time performance. The latter is often a trade-off with the amount of surface detail that can be generated. Yu et

al. [25] generate surface details for a flowing river using advected particles representing samples of wave textures. Whereas, Mitchell[14] uses a GPU implementation of procedural waves to provide surface detail for an ocean scene at real-time rates.

The method proposed by Yu et al. of generating surface details is quite efficient and produces meaningful surface details. The memory usage is quite efficient as the calculations for the velocity field, particles and texture generation are all done on the fly. This means very little is pre-calculated and stored in memory. The memory usage only depends on the initial fluid network and the number of visible particles. Also this method is quite efficient in terms of processing requirements as the surface details are only generated for the visible surface area of the river. This allows this method to be applied to very large river systems without any performance loss. This method is able to produce real-time results ranging from 25 – 100 frames per second depending on how much of the river surface is visible.

Yu et al. are able to show a number of different properties of the river such as changes in flow rate due to changes in the width of the surrounding bank. The rate of flow increases as the river banks narrow and decreases as the river banks widen. Also Yu et al. is able to show changes in the flow rate due to islands located inside the banks i.e. water appears to flow around an island. One of the main drawbacks of this method is that the resultant river system does not exhibit any turbulence or vorticity which arises from the fact that the velocity field is stable i.e. it does not change over time.

Mitchell takes a different approach to generating surface geometry. Instead of composing a number of wave samples together to form a bump-map, he uses the Inverse Fast Fourier Transform (IFFT) to generate a wave height-map.

This method is conceptually very straightforward. The real contribution of this paper is that it implements this process on a GPU which improves the performance. The overall method is separated into two height-maps: A low band height-map and a broad-band height-map. The low band height-map contains low frequency sinusoidal waves and it is used to displace vertices. The

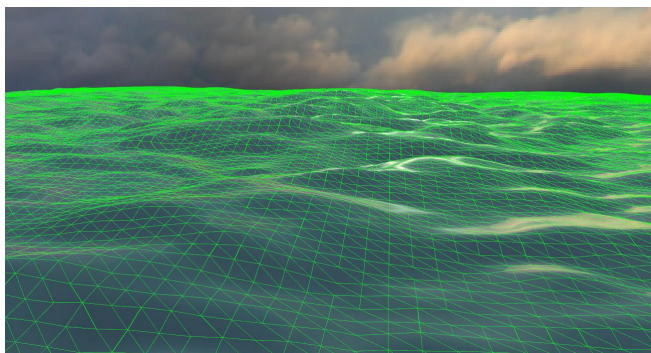


Figure 2.5: An example of the results produced by Mitchell [14]

broad-band height-map contains low and high frequencies sinusoidal waves and is used to create a normal map to add finer features to the water surface. By dampening the broad-band map it gives the appearance that depth of the water is decreasing.

This method has a number of advantages that make it suitable for large scale generation of surface details. It has a low memory requirements, only the initial Fourier domain spectrum must be stored. Real-time rates are easily produced with the implemented system running at 140 Hz and a rendered scene running at 65 fps.

2.2.3 Efficient Display of Large Water Systems.

Rendering large water systems is very time consuming because of the amount of rendering that must be done and the cost of rendering itself. It is not always feasible or desirable to try to display the entire scene every frame. A more efficient approach is required. Generally in Computer Graphics a view dependent sub-set of a virtual scene is rendered in place of the entire scene to reduce unnecessary rendering operations. This notion has been applied to unbounded water by Hinsinger et al. [5] who projects a water surface from screen space to world space which ensures only visible water is rendered. Johanson [8] provides a more thorough treatment of a similar projection scheme. Hu et al. [7] takes a similar approach but restricts the real-time generation waves to water close to the viewer.

Hinsinger et al. present a method to procedurally generate an ocean scene using Gerstner Waves as the underlying physical model. A height-map is generated by summing sinusoidal waves in the spatial domain. Two main optimisations are created to improve the efficiency of displaying the ocean scene: Only generating waves on the visible section of the ocean scene and only using wavelengths that are visible to the observer.

Second optimization is relevant to this thesis, to generate waves only on visible sections of the ocean, a grid is defined in screen space that covers the entire viewport. Different distributions of the grid points in screen space can be used to add more detail to near or far sections of water. This grid is then projected onto the ocean surface in world coordinates. Now, waves can be efficiently generated on this new grid. The cost of generating rendering does not change with amount of water being viewed, because the ocean scene geometry is created in screen space not world space. Hence, large surface areas of ocean can be viewed with no extra cost compared to small surface areas.

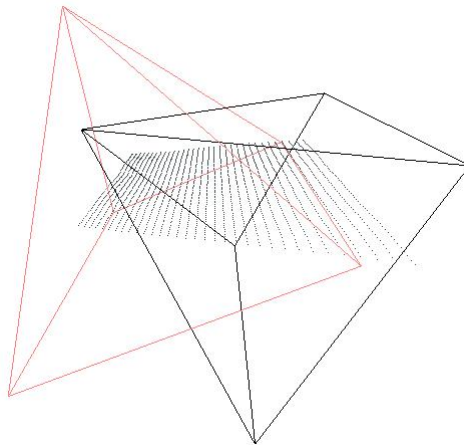


Figure 2.6: The projected grid used by [8]. The red frustum represents the projector and the black frustum represents the viewer.

2.2.4 Splashes

Water exhibits many dynamic behaviors like foam, splashes, spray, mist, bubbles etc... These features when added to virtual water increase the realism and provide extra information about the type of water. For example, if foam is present it can show that water is splashing into itself. These features are often treated as sub-systems of larger simulations. As in Takahashi et al. [22] a particle system is used to show spray, mist and foam (see figure 2.7). Similarly Maes et al. [12] use particles systems to show discontinuities in flow like waterfalls.



Figure 2.7: Splashes and foam in a simulation from [22]

Takahashi et al. [22] couple a Cubic Interpolated Propagation (CIP) solver with a particle system to produce splashes, foam and mist. The CIP solver is able to simulate both the water and air which allows splash particles to be affected not only by gravity but wind. Although focused on offline rendering, the comprehensive treatment of the particles, splash, foam and mist, provides a solid theoretical foundation for real-time adaptations. The paper describes the generation, vanishing and transition of splashes to foam. The volume of mist is defined on a grid and determined by the density of splash particles. Instead of rendering mist as many particles, the volume of mist present in the grid is rendered.

Splashes are generated when the curvature of the water surface reaches a

threshold. At that point, an amount of particles related to the curvature is emitted at a velocity equal to that of the water surface. The particles once emitted are affected by gravity, air resistance, initial velocity and the velocity of the surrounding wind. The splash particles are simulated until they make contact with the water again. At which point, the particle transitions from being a splash particle to a foam particle.

Similarly Maes et al. [12] use a particle sub-system, but instead of a CIP solver a water-column based method is used to simulate water in real-time. The particles in this system contain a mass and show when water disconnects from itself. Instead of using the curvature of the surface to indicate particle emission, a breaking wave model is used. When a column of water reaches a critical height, the volume of water above the critical height is converted into particles with an initial velocity equal to that of the top of the water column. Once particles are created, they move under gravity. The mass of the particles is reabsorbed once it collides with the water surface.

A unique feature of this model is that particles exert pressure on the water surface upon collision. As shown in the results, this pressure results in disturbances on the water surface. This is a useful feature for modeling waterfalls where the falling water produces recognisable ripples. Unlike the other methods there is no creation of foam or any other features. This method is implemented on a GPU in an attempt to increase performance by processing particles in parallel. The removal and creation of particles is done on the CPU and the simulation is done on the GPU. The researchers identify the transfer of data to and from the GPU and CPU as a bottleneck. To overcome this, they perform multiple height-field simulations to one particle simulation.

2.3 Summary

Real-time fluid simulation for Computer Graphics has very different goals and constraints compared to other types of simulation. The main goal being

to produce visually plausible results in a small amount of time. To accomplish this, a number of techniques have been developed over the years. Each technique has its own benefits and drawbacks. Procedural methods tend to be very stable and fast but not as interactive as other methods. Height-field methods offer a balance of flexibility and performance. Eulerian grid based methods tend to produce the most accurate results with a high computational cost. Particle systems offer very flexible results but again with a high computational cost. Using these methods a number of scales of fluid have been simulated: small scale drops of water; puddles and pools of water; lakes and rivers; and unbounded water i.e. oceans.

There are number of different issues facing large scale real-time water simulation. These issues result from the inability of standard procedural, particle and height-field methods to scale to large spatial domains whilst maintaining real-time rates. Consequently, a number of optimizations have been developed including GPU implementations of algorithms, specialized spatial partitioning data structures and different approaches to reducing ineffectual computations. Still, there is no one general approach to real-time water simulation. This is because any improvement in simulation speed makes it possible to increase the accuracy and number of features that the water can exhibit which again slows the simulation down. The lack of a general approach and the diverse number of behaviours water exhibits (splashes, spray, waves, flow etc...) makes real-time fluid simulation a very rich area for novel research. The main area which lacks research is efficient algorithms that can simulate the propagation of water over a very large terrain in real-time - most algorithms assume that the path of the water does not change over time. This will be the focus of the proposal.

Chapter 3

Problem Description

3.1 Aim

Today's graphics cards and CPUs are very fast. They can produce amazing real-time results because of fluid simulations ability to be parallelised across the many cores of today's computational devices. However, these simulations are still bound by the processing capabilities of the device. There is still a need to reduce the processing time of fluid simulations. Often simulations done in games are not allocated large amounts of CPU cycles. So we start to see a need for managing simulation techniques rather than a single simulation technique. Managing a simulation would allow it to be either increased in size or function with less resources. The goal of the managing a simulation would be to reduce ineffectual and unnecessary computations. This gives rise to some questions: What is an ineffectual computation in a fluid simulation? How can they be identified and then removed from a simulation? Another optimisation can be made from the way water is represented. There are two salient approaches to large scale water simulation: one is having a global static representation with a local dynamic representation and the other having a dynamic global representation of the water. The first approach was used by Yu et al. [25]. They were able to develop a system where the surface of the water was dynamically calculated for the viewing frustum. The

obvious problem with this approach is that the water can never change its position on a global scale i.e. it must be defined at the beginning. However, this approach supports very large terrains by having the computations required related only to the visible amount of water.

The second approach where the water is simulated globally, results in the problem that the computations required grows with the spatial domain. So the performance is bounded by size of the spatial domain. The aim here is to find the middle ground between these two approaches where some of the global simulations work is offloaded to the a local graphical simulation. Therefore the aims of the thesis are identified as follow:

To show that it is possible to reduce computations of an existing water simulation technique and maintain comparable results by:

1. Identifying and reducing unnecessary computations.
2. Off-loading part of a global simulation to a local simulation.

3.2 Selected Approach

I would like to propose a method to dynamically create a large river system over a large terrain at real-time rates. The system is structured into four main components: the physical volume model, the physical volume model manager, the local simulation and finally the renderer. The goal of this structure is to have a low cost physical model that can be applied selectively by the physical volume manager and a local simulation that produces a more detailed representation of the system that can be rendered.

The physical volume model determines how the physical quantities change over time. Given some initial quantities like the volume of water in an area and the forces acting upon the water this model will determine the quantities after a given amount of time has passed. The physical volume model manager

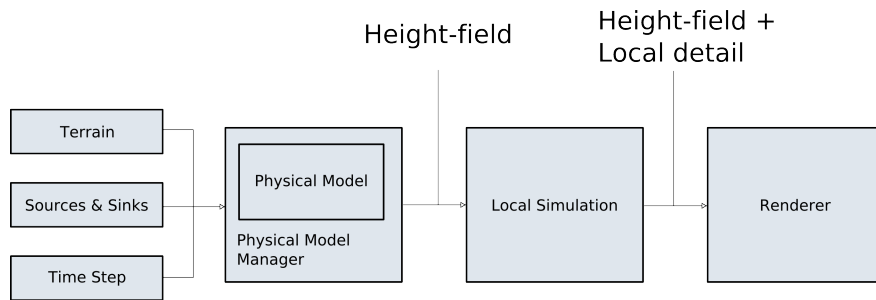


Figure 3.1: A conceptual view of the proposed system.

will be responsible for selectively applying the physical volume model; so that the system does not apply ineffectual computations. This concept is similar to “sleeping” done in rigid body simulations but applied to water simulations. It is likely that there will be water with varying levels of activity in a large spatial domain e.g. flowing water, a water fall, a calm strait etc... Some of these sections of water will not change over time. Consider water flowing down a river: Once the river is established it will essentially stay the same. It is only the ripples which change. By recognising parts of a large water system whose volume does not exhibit a net change in volume over time, we can then eliminate the volume calculations of that section. Regions that do not exhibit a net change in volume over time will be considered ‘stable’.

The local simulation will take a visible stable region and considering its state (volume, flow rates etc...) and will generate surface details like ripples. Furthermore, there is one more optimization which is to remove the particle systems used to transport water from the physical model. They will be replaced by a simple approximation described section 4.2.2. This simple approximation will then be converted by the local simulation into actual particles. This converts particles from a physical device to transfer volume to a graphical entity which represents splashes. This optimization maintains the accuracy in terms of net volume transfer but also, reduces a significant amount of computations that would have needed to be done. The local simulation will not be further developed in this thesis. The work described in section 2.2.2 provides an adequate foundation for a local simulation module.

3.3 Selected Physical Volume Model

In order to test the concept of the system, a physical volume model must be selected from those that have been described in the section 2.1. To test that this system is viable the simplest approach that fits the aims, general properties and below information will be selected. If the concept proves viable later iterations can focus on applying more advanced algorithms. The global water simulating technique must have the following attributes:

- The approach must be able to produce results at a real-time rate, if the simulation cannot produce real-time results then no amount of selective application will produce real-time results.
- The approach must be able to simulate the volume and flow rate of water in space and time.
- The approach must be a physically based model of water.

Following on from these attributes and some simple observations of water, some extra information is determined about water that will be simulated: The majority of water will be located on the terrain, exceptions being waterfalls and splashes where water is located in the air. The water being simulated will be very large in volume.

Each general approach will be evaluated in turn starting with the Eulerian Grid approach described in section 2.1.3. There are a number of issues which arise from taking this approach for real-time applications. First the grid must cover the entire space where the water will appear. It is unclear about how high this grid should be above the terrain e.g. Should it be high enough to allow water to travel 10 meters up? The larger the grid the more processing time and the larger the more memory required. The terrain must be voxelised so that its boundary conditions can be represented in the grid. It is unclear about the granularity that the grid should be. If we want small scale effects then the entire grid must be small enough to display those effects. The finer the grid the more processing time and the more memory

is required. There is also the added overhead of extracting a surface from the grid representation of the water. The water is generally represented as a scalar field which must be converted to polygons (ray tracing is too time consuming in real-time graphics at the moment). This would require the use of the marching cubes or marching tetrahedron algorithm which adds the cost of recreating a geometric representation of the water every frame.

Next are Particle Systems described in section 2.1.4, I refer to particle systems where the particles interact with one another. Again there are a number of issues with selecting this approach. The first being, that in order to model a large volume of water there must have a large number of particles to represent that volume. The more particles the more computations and the more memory that is required. Similarly to the Eulerian approach we must select the granularity of the simulation which is difficult especially if the water is viewed from many distances. Particles that are too small will incur a large computational overhead and particles that are too large make the water look and behave incorrectly. Particle systems are able to model a diverse set of behaviours e.g. waterfalls, flowing rivers etc... If the majority of the water to be modeled lies directly on the terrain the particles may never come into a situation where they are able to express their dynamic behaviour. Similarly with the Eulerian grid approach, one of the main problems with particle systems is extracting a temporally coherent surface from the particles.

There is no procedural model that is able to take volume into account. So it would be difficult, to procedurally model water flowing over a terrain. Finally, we are left with height field models, described in section 2.1.2. This model has a number of advantages over the other models. First it has a natural graphical representation i.e. a height-field of water can be trivially converted into a mesh. It fits with the requirement of water being located on the terrain. The computational overhead is related to the surface area not the volume.

The Eulerian and Particle System approaches require a lot more memory and processing than height-fields. In addition there is the difficulty of meeting their requirements in small scales or without GPUs. Height field

models offer the best fit for this thesis and will form the basis for the physical volume model. However, there are still two main approaches to choose from: pipe and volume model introduced by O'Brien et al. [18] and the Shallow-Water/Simplified NS equations approach Kass and Miller [9].

The Pipe and volume approach has many advantages, one being that it is able to be coupled with particle systems to handle discontinuities in water. Another being that it can be integrated overtime with a simple numerical integration scheme. Alternatively, the NS approach can provide a more realistic water simulation, however, at the expense of a more complicated integration scheme which involves solving a system of linear equations. For this thesis, the volume model appears to be the simpler and more flexible approach.

Chapter 4

Techniques and Methods

This chapter formally defines the components of the proposed system. Given the limited time available and other factors, only the two-dimensional case is looked at. This then removes the need to do any complicated rendering and hence the renderer will be omitted.

4.1 The Physical Volume Model

The physical volume model is based on the work of O'Brien et al. [18] which was later extended by Mould and Yang [15]. It represents the terrain as a height-field on top of which a number of columns are constructed. Columns are connected to their neighbors by pipes which allow water to flow from one column to its neighbors. The system simulates water by calculating the pressure at either end of a pipe and hence the flow rate across it. The

simulation process is described by the following algorithm:

Algorithm 1: Update the System

Input: Δt

```

1 begin
2   InitializePipes()
3   while simulation is running do
4     UpdateParticleSystem( $\Delta t$ )
5     UpdateFlowRates( $\Delta t$ )
6     UpdateVolumes( $\Delta t$ )
7     UpdateSources( $\Delta t$ )
8   end
9 end

```

The process of calculating flow rates and volumes of columns are given below; so **UpdateFlowRates** and **UpdateVolumes** will not be further refined.

4.1.1 The structure of the model

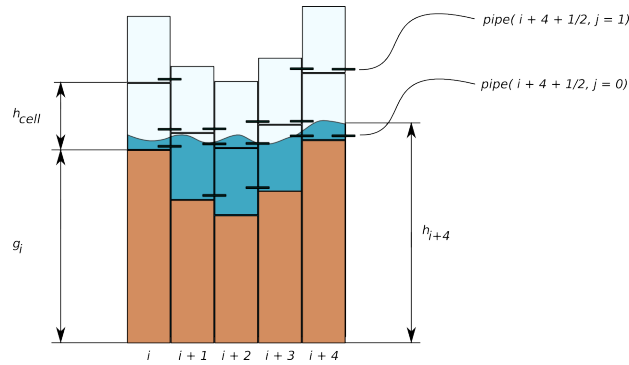


Figure 4.1: The height-field model

First the terrain is defined as a height-field where every spatial coordinate corresponds to one height value. The height of the terrain is g_i where $i \in \mathbb{N}_0$ represents the spatial index. At each point i on the terrain a column is

defined which holds a volume of water v_i^t , note that the superscript is not an exponent but a time index. From the volume of water in a column the water height, h_i^t at column i , can be determined with:

$$h_i^t = \frac{v_i^t}{wd} + g_i \quad (4.1)$$

where w is the width of a column and d is depth of the column. Because we are looking at a two-dimensional case, we can let $d = 1$, hence:

$$h_i^t = \frac{v_i^t}{w} + g_i \quad (4.2)$$

Each column is divided into a number of cells of a specific height h_{cell} . Each of these cells are connected to its neighboring cells by a pipe through which water will flow. Pipes are defined half way between the adjacent columns and are given another index to indicate their position above the ground $[i \pm \frac{1}{2}, j]$. This refers to the j^{th} pipe from the ground to the right, '+', or left '-' of the column i . A hydrostatic pressure is present at both ends of the pipe. The pressure at the left end of a pipe is given by:

$$P_{left, i+\frac{1}{2}, j}^t = (h_i^t - height_{i+\frac{1}{2}, j})\rho g + p_0 + E_i^t \quad (4.3)$$

$height_{i+\frac{1}{2}, j}$ refers to the height of the base of the pipe, ρ is the density of the water, g is gravity, p_0 is environmental pressure and E_i^t is the sum of any external pressure being applied to the water surface. Similarly for the right end of the pipe:

$$P_{right, i+\frac{1}{2}, j}^t = (h_{i+1}^t - height_{i+\frac{1}{2}, j})\rho g + p_0 + E_i^t \quad (4.4)$$

It is possible to calculate the flow rate of the water moving through a pipe using the pressure at each end of that pipe. The pressure across a pipe is then defined as:

$$\Delta P_{i+\frac{1}{2}, j}^t = P_{left, i+\frac{1}{2}, j}^t - P_{right, i+\frac{1}{2}, j}^t \quad (4.5)$$

The Flow rate across a pipe between two columns can now be defined. The flow rate from column i to column $i + 1$ through pipe j is given by:

$$\eta_{i \rightarrow i+1, j}^{t+\Delta t} = f\eta_{i \rightarrow i+1, j}^t + \frac{\Delta t}{\rho l} \Delta P_{i+\frac{1}{2}, j}^t \quad (4.6)$$

f is a friction coefficient and l is the length of the pipe. Note that the flow from column i to column $i + 1$ is opposite of $i + 1$ to i :

$$\eta_{i \rightarrow i+1, j}^t = -\eta_{i+1 \rightarrow i, j}^t \quad (4.7)$$

The flow of water into a given column is calculated by summing the individual contributions of its pipes.

$$\eta_i^{t+\Delta t} = \sum_{j=1}^{n_{i-1 \rightarrow i}} \eta_{i-1 \rightarrow i, j}^{t+\Delta t} s_{i-\frac{1}{2}, j} - \sum_{j=1}^{n_{i+1 \rightarrow i}} \eta_{i+1 \rightarrow i, j}^{t+\Delta t} s_{i+\frac{1}{2}, j} \quad (4.8)$$

Where $n_{i \rightarrow k}$ where $k = i \pm 1$ is the number of pipes going from column i to k , $s_{i \pm \frac{1}{2}, j}$ is the surface area of the pipe, which is the overlap of the cells the pipe connects. Now that the flow rate of water moving into a column is known, its change in volume and hence the change in height can be determined:

$$v_i^{t+\Delta t} = v_i^t + \Delta t \eta_i^{t+\Delta t} \quad (4.9)$$

It is now possible to calculate the heights of water on the terrain over time. But before the movement of water can be simulated, the location of the cells and pipes must also be calculated. More specifically, the height the of a pipe above the ground, $height_{i+\frac{1}{2}, j}$ and the surface area of a pipe, $s_{i+\frac{1}{2}, j}$. These values can be calculated from the number of columns in the terrain, $n_{columns}$, the height of each cell, h_{cell} , the maximum number of cells per column, n_{cell} ,

and the maximum surface area of a pipe, s_{max} .

Algorithm 2: Initialise Pipes

Input: $n_{columns}$, h_{cell} , n_{cell} , s_{max}

Output: set of pipes

```

1 begin
2   foreach  $i = 0$  to  $n_{columns}$  do
3     CellBase =  $g_i$ 
4     CellTop =  $g_i + h_{cell}$ 
5     foreach  $j = 0$  to  $n_{cells} - 2$  do
6       CellBaseRight =  $g_{i+1}$ 
7       CellTopRight =  $g_{i+1} + h_{cell}$ 
8       while CellBaseRight <  $g_{i+1} + h_{cell} \times n_{cell}$  do
9         if CellBase < CellTopRight and CellTop > CellBaseRight
10          then
11            PipeBase =  $\min(\text{CellBase}, \text{CellBaseRight})$ 
12            PipeTop =  $\max(\text{CellTop}, \text{CellTopRight})$ 
13             $s_{i+\frac{1}{2},j} = s_{max}(\text{PipeTop} - \text{PipeBase})/h_{cell}$ 
14             $\text{height}_{i+\frac{1}{2},j} = \text{PipeBase}$ 
15          end
16          CellBaseRight = CellBaseRight +  $h_{cell}$ 
17          CellTopRight = CellTopRight +  $h_{cell}$ 
18        end
19        CellBase = CellBase +  $h_{cell}$ 
20        CellTop = CellTop +  $h_{cell}$ 
21      end
22      PipeBase =  $\max(g_i + h_{cell} \times n_{cell}, g_{i+1} + h_{cell} \times n_{cell})$ 
23      PipeTop =  $\min(g_i + h_{cell} \times n_{cell}, g_{i+1} + h_{cell} \times n_{cell})$ 
24       $s_{i+\frac{1}{2},n_{cell}-1} = s_{max}(\text{PipeTop} - \text{PipeBase})/h_{cell}$ 
25       $\text{height}_{i+\frac{1}{2},n_{cell}-1} = \text{PipeBase}$ 
26    end
27  end

```

4.1.2 Particle System and Sources

There are two more structures that must be defined sources/sinks and the particle system. The particle systems is used to model discontinuities. Particles can be emitted from a water column carrying some mass which is re-absorbed into the first water column with which it collides. There have been a number of conditions used to indicate that a particle should be emitted: upward velocity [18], relative height of a water column to a neighbor[11][6] and similarly the mean curvature[22]. The simplest and most appropriate condition for this model is the relative height of a water column to its neighbour. Essentially the total amount of water emitted m_{total} is distributed to particles of mass $m_{particle}$ with one particle taking the remainder:

$$m_{total} = m_{particle}(n - 1) + m_{remainder} \quad (4.10)$$

Where n is the number of particles emitted. Now let a particle be defined as a triple $p = (\vec{x}_p, \vec{v}_p, m_p)$. Where \vec{x}_p is the particles position, $\vec{v}_p = \frac{d\vec{x}_p}{dt}$ is the velocity of the particle and m_p is the mass of the particle. Let the set of all particles be $P = \{p_1, p_2, p_3, \dots, p_{n_{particles}}\}$. Once particles are emitted they are acted upon by gravity and air friction:

$$\vec{F}^t = \vec{g} - f_{drag} \vec{v}^t \quad (4.11)$$

Where $f_{drag} \in [0, 1]$. When a particle comes in contact with the water its volume is then added to the column and an ad-hoc force is applied to the water:

$$F_{collision} = v_y mc \quad (4.12)$$

where v_y is downwards component of the particle's velocity, m is the mass of the particle and c is constant that is used to tune the results.

Algorithm 3: Update Particle System

Input: $\Delta t, p$

```

1 begin
2   foreach  $p \in P$  do
3      $\vec{v}_p^{t+\Delta t} = \vec{v}_p^t + \Delta t \vec{F}_p^t / m_p$ 
4      $\vec{x}_p^{t+\Delta t} = \vec{x}_p^t + \Delta t \vec{v}_p^t$ 
5     if  $p$  is in contact with terrain or water then
6        $ContactPoint = CalculateContactColumn(p)$ 
7        $ApplyForce(ContactPoint, p, \Delta t)$ 
8        $Remove(p)$ 
9     end
10  end
11 end

```

Sources and sinks add/subtract a certain amount of water into/from a particular column over some time. Traditional approaches have fixed the height of a water column to indicate a source or sink. However, this dampens or eliminates ripples that propagate over a source column. So a better approach is to introduce a certain amount of water every time step, analogous to a pump. Let a source be a pair $s = (\eta_s, j_s)$. Where $\eta_s \in \mathbb{R}$ is the sources flow rate and j_s is the index of the destination column. To update a source s we add its contributing volume $\Delta t \eta_s$ to its destination column v_{j_s} .

A set of sources and sinks are thus defined as $S = \{s_1, s_2, s_3, \dots, s_{n_{sources}}\}$. A condition that must be maintained is that the volume of a column's volume cannot be less than 0. So if a source makes a column less than 0 the column's

volume is set to 0.

Algorithm 4: Update Sources

Input: $\Delta t, S$

```

1 begin
2   foreach  $s \in S$  do
3      $v'_{j_s} = v_{j_s} + \Delta t \eta_{s_s}$ 
4     if  $v_{j_s} < 0$  then
5        $v_{j_s} = 0$ 
6     end
7   end
8 end

```

4.2 The Physical Volume Model Manager

The fundamental unit used in the model manager is the *region*. A region contains a set of adjacent columns, specifically it contains the height of the column and the height of the terrain as well as a set of sources and sinks. Regions have the condition that they do not overlap each other. Let H be the set of all water column heights $H = \{h_1, h_2, h_3, \dots, h_n\}, h_i \in \mathbb{R}_{\geq 0}$. From the previous section we have the set S of all sources. We have a set of the terrain heights $G = \{g_1, g_2, g_3, \dots, g_n\}, g_i \in \mathbb{R}_{\geq 0}$. So now a region can be defined as a triple $r = (H_r, S_r, G_r), H_r \subseteq H, S_r \subseteq S$ and $G_r \subseteq G$. So, let R be the set of all regions $R = \{r_1, r_2, r_3, \dots, r_n\}$.

Regions are created, destroyed and merged throughout the simulation. The first region is created when a source of water is added to the terrain. A pooling region is created around the source. This region will continue to grow until it comes in contact with a downward sloping piece of terrain. At which point the flowing region is created to contain the water as it flows down the hill. Alternatively, if the terrain is of sufficient steepness then a waterfall region will be created. The behavior of each of these regions will be described in section 4.2.1.

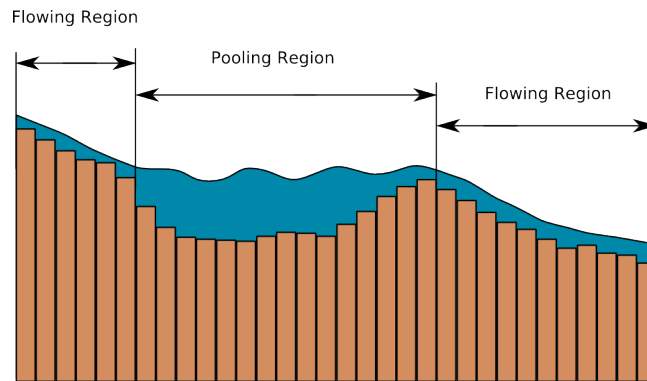


Figure 4.2: An overview of the region model

How ineffectual computations can be reduced can now be described. In a region there will be a number of inputs/outputs of water such as sources, water falls and an inflow and/or outflow of water from/to adjacent regions. If the assumption is made that the volume model is primarily concerned with the transport of water and not necessarily the shape of its surface than an optimisation can be made when the net flow of a region is zero. At that point, the simulation of that region can be stopped as that region's volume will remain constant and will no longer grow or shrink. The change in height of the individual columns can be represented in a local simulation. So formally, the net flow of a region, r , is defined by:

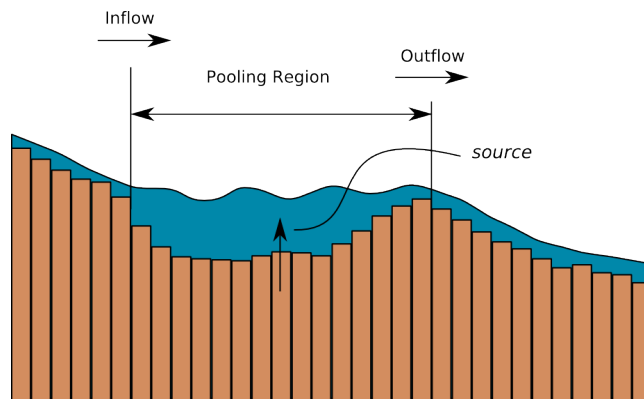


Figure 4.3: The inflows and outflows of a region

$$Netflow(r, t) = \sum_{s \in S_r} \eta_s + \eta_{left}^t + \eta_{right}^t \quad (4.13)$$

Where η_s is the flow of an individual source, η_{left} is the flow rate at the leftmost column of the region and η_{right} is flow rate at the rightmost column of the region. The volume in a region R_i at time t is:

$$Volume(r, t) = w \left[\sum_{h^t \in H_r} h^t - \sum_{g \in T_r} g \right] \quad (4.14)$$

The volume at time $t + \Delta t$ is:

$$Volume(r, t + \Delta t) = Volume(r, t) + \Delta t Netflow(r, t) \quad (4.15)$$

Now if the $Netflow(r, t) = 0$, it follows that the volume of that region won't change i.e. $Volume(r, t + \Delta t) = Volume(r, t)$. Essentially the simulation can be stopped in that region as its volume will not change. When the net flow becomes zero the region is considered stable. However, in practice this condition is not usually met. If however, it is met then it is for only a moment. The surface waves of the region create a variation in the flow rates at either end of the region. To reduce this noise two factors will be introduced. A constant $C_{stable} \in \mathbb{R}_{\geq 0}$ the magnitude of the net flow only needs to be below this threshold to be considered stable, however, introducing the constant will introduce an error either a loss or increase of volume to the system which will be discussed later. The second factor is to take a moving average of the net flow over time, this eliminates sudden sharp increases or decreases of the surface from affecting the stability, the average net flow is given by:

$$avg(Netflow(R_i, t)) = \frac{1}{n_{samples}} \sum_{j=0}^{n_{samples}} Netflow(R_i, t - j\Delta t) \quad (4.16)$$

Where $n_{samples}$ is the number of samples and we assume that Δt is constant and $t > j\Delta t$ this means that we choose not to calculate the average net flow and hence stability of a region until we have $n_{samples}$. So the condition for

stability now becomes:

$$\boxed{\text{A region is stable at time } t \text{ iff } |\text{avg}(\text{Netflow}(R_i, t))| < C_{\text{stable}}}$$

4.2.1 Regions

The basic algorithm for updating this region system is described below. It has a number of similarities with the other approach. The most important difference is that regions are updated and not individual columns.

Algorithm 5: Update Region System

Input: $\Delta t, R$

```
1 begin
2   foreach  $r$  in  $R$  do
3     UpdateRegion( $r$ )
4     if  $r$  is not stable then
5       UpdateFlowRates( $r, \Delta t$ )
6       UpdateVolumes( $r, \Delta t$ )
7       UpdateSources( $r, \Delta t$ )
8     end
9   end
10 end
```

The **UpdateRegion** function takes in a region and then looks at its boundaries, the height and contents of the adjacent columns to that boundary. It will then determine if any action should be taken. For instance, if pooling region's boundary contains water and the adjacent column, of suitable height, is empty the region will extend outwards. There are three types of regions: pooling, flowing and water fall. Each region is differentiated from each other by how they interact with their boundary, but they can be described as follows:

- *pooling regions*, marks a body water contained by a cup shape.
- *flowing regions*, marks a body of water flowing down a hill.

- *waterfall regions*, marks the beginning of a water fall.

Table 4.1 and 4.2 give a description the behavior of the different regions of the pooling and flowing region respectively. To determine how the boundary of a region should change there needs to be some understanding of potential new boundary. It is then classified into a number of different types. Let b be the index of the left or right boundary of a region and p be the potential new boundary of that region where $p = b \pm 1$. Now p can be classified in to the following types of terrain.

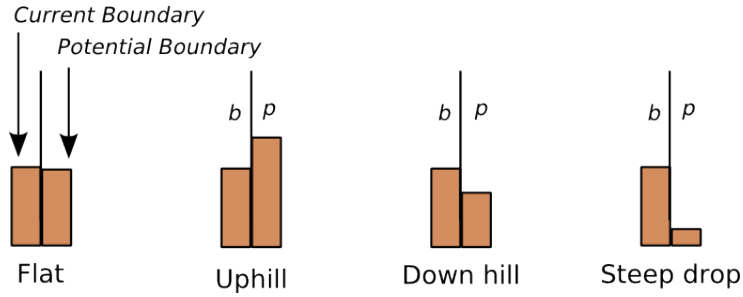


Figure 4.4: The classification of columns

- flat, $|g_b - g_p| < C_{tolerance}, C_{tolerance} \in R^+$
- uphill, $g_b + C_{tolerance} < g_p$
- downhill, $g_p - C_{sharpdrop} < g_p < g_b + C_{tolerance}$
- sharpdrop, $g_p < g_b + C_{tolerance} - C_{sharpdrop}$
- boundary, $p = 0$ or $p = n_{columns}$
- pooling region, $(\exists r \in R)[contains(r, p)$ and r is a pooling region]
- flowing region, $(\exists r \in R)[contains(r, p)$ and r is a flowing region]
- waterfall region, $(\exists r \in R)[contains(r, p)$ and r is a waterfall region]

$contains(r, p)$ is true if the p is located inside region r . The following are the operations that can be done on a regions:

TerrainType(p)	Region Operation
uphill	extend region
downhill	create a flowing region
sharpdrop	create a waterfall region
boundary	Nothing
pooling region	merge two regions
flowing region	extend region
waterfall region	connect regions

Table 4.1: Pooling Region Behavior

TerrainType(p)	Region Operation
flat	extend region
uphill	create a pooling region
downhill	extend region
sharpdrop	create a waterfall region
boundary	create a pooling region
pooling region	connect regions
flowing region	connect regions
waterfall region	connect regions

Table 4.2: Flowing Region Behavior

- *extend region*, extends the region to include new column
- *shrink a region*, excludes the boundary from the region
- *merge two regions*, merges the columns and sources of two regions
- *create a pooling region*, creates a new pooling region
- *create a flowing region*, creates a new flowing region
- *create a waterfall region*, creates a waterfall region
- *connect regions*, connects the two regions together
- *destroy region*, removes the region from the set of all regions

4.2.2 Water Fall Region

The waterfall region differs from the flowing and pooling regions by the fact that it contains only one cell. It also does not transfer water directly to its neighbor but it transfers the water through the air to another region see figure (a) 4.5 . To determine where the water will be transferred to, two particles are emitted from the waterfall region p_{max} and p_{min} . These particles represent the furthest we would expect a particle to travel p_{max} and the least amount of distance we expect the particle to travel p_{min} . The initial velocity of p_{max} and p_{min} is:

$$\vec{v}_{max}^{\rightarrow t_0} = \vec{i} (1 + C_{xsply})\eta_j^t + \vec{j} \frac{dh_j}{dt} \quad (4.17)$$

$$\vec{v}_{min}^{\rightarrow t_0} = \vec{i} (1 - C_{xsply})\eta_j^t + \vec{j} \frac{dh_j}{dt} \quad (4.18)$$

The path of both particles are traced until they make contact with the terrain at i_{max} and i_{min} . Where these two particles collide with the ground a new source and region will be added. The type of source added is different from the previous types because it transfers water not just to one column but to many columns. Its inflow is the outflow of the waterfall region. If there is region where the source is located then the source is added to that region otherwise a new pooling region is formed. This region is extended to encompass i_{max} and i_{min} .

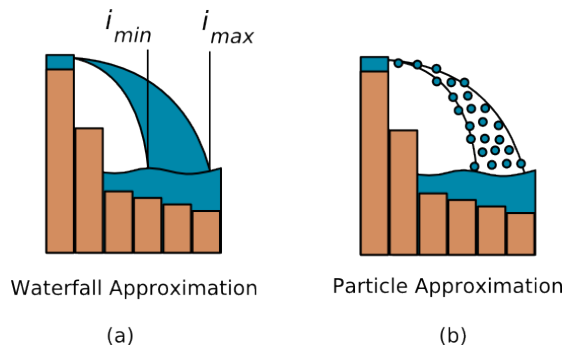


Figure 4.5: The different types of waterfalls

Algorithm 6: Updating a water fall

Input: $\Delta t, r, R$

```
1 begin
2   if  $r$  is not stable then
3      $p_{max} = (\vec{x}_0, \vec{v}_{max}^{t=0}, m_{particle})$ 
4      $p_{min} = (\vec{x}_0, \vec{v}_{min}^{t=0}, m_{particle})$ 
5      $i_{max} = \text{conactPoint}(p_{max})$ 
6      $i_{min} = \text{conactPoint}(p_{min})$ 
7     points =  $\{i \in \mathbb{N}_0 : i_{min} \leq i \leq i_{max}\}$ 
8     regions = IntersectedRegions( point, R )
9     if regions is not empty then
10      join regions together
11      extend regions to include all of points
12    else
13      create a region to encapass points
14      add a source to that region
15    end
16  end
17 end
```

4.3 Testing Method

Now that the system has been defined, the next section will test it to see if it satisfies the aims. A number of test scenarios will simulated. During every simulation step a number of quantities will noted:

- total update column flow and update column volumes
- total volume
- total update particle position
- total region operations

- total stability state transitions

These quantities will be taken for an implementation of the physical volume model and for a simulation using a physical model manager. By comparing the two algorithms we can see if the physical volume model manager offers any benefit.

Chapter 5

Results

This chapter presents the results of the tests performed. The tests aim to determine whether or not computations are culled and to identify the types of errors introduced by the optimisation. Each test has one source added initially and after some time, a sink of equal magnitude is added. The expected result is that the total volume in the system will increase until the sink is added at which point the total volume will remain constant. The number of computations of the optimised version is expected to grow then shrink suddenly as regions become stable. When the sink is added shortly thereafter, it is expected that all the regions will stabilise and no computations will take place. Ideally, the difference in volume between the optimised and unoptimised versions, or the error, should be small and not grow over time. The results compare the unoptimised version of a simulation and the optimised version of the simulation; the goal being to produce volume similar to unoptimised version for less cost.

5.1 The Trivial Case

This is the simplest scenario is analogous to a box being filled with water. A source of a value '3' is initially added and sink of value '-3' is added some time after. The stabilisation threshold, $C_{stable} = 0.05$ is used.

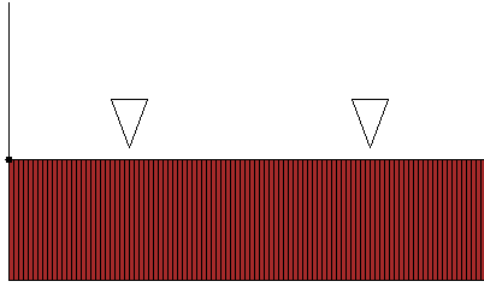


Figure 5.1: A view of the trivial case

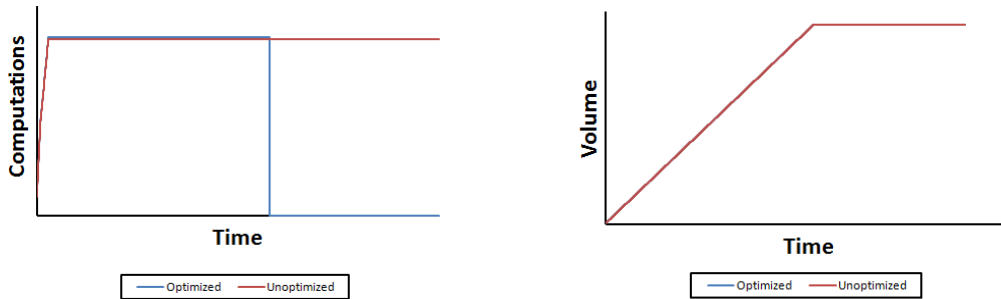


Figure 5.2: Results of the trivial case and

The results for this test case are as expected. The optimised version suddenly decreases in the number of computations when the sink is added. The error appears to be negligible.

5.2 The Downhill Case

This scenario tests the ability of the program to eliminate ineffectual computations on a hill. In figure 5.2 a source of a value ‘6’ is initially added and sink of value ‘6’ is added some time after with $C_{stable} = 0.0005$. The threshold is too small and the system is unable to stabilise; the performance is bounded by the unoptimised version.

A number of issues become apparent from this test:

- Inaccuracies in the underlying physical simulation affect the results.

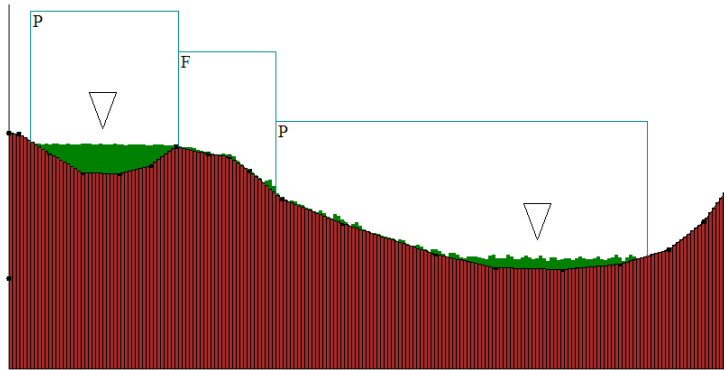


Figure 5.3: A view of the downhill case

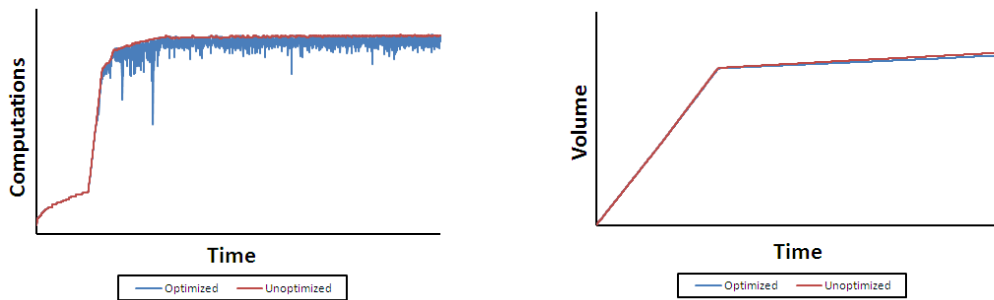


Figure 5.4: Unsuccessful results of the downhill case

The chart shows that the unoptimised system's volume keeps growing after the sink has been added. This affects possible measures of accuracy.

- The selection of a stabilisation threshold is an ad-hoc process, a number of different values have to be attempted before a successful one is found.
- If the stabilisation threshold is too close to the source's value, the region will stabilise because the inflow is considered to be non-existent.

A simple solution for the extra water is to use a sink with larger flow rate than the initial source to absorb any extra water. In figure 5.2 a source of a value '6' is initially added and sink of value '9' is added some time after with $C_{stable} = 0.01$. This produces results closer to expectations. The optimised version eventually stabilises and the error does not increase over time.

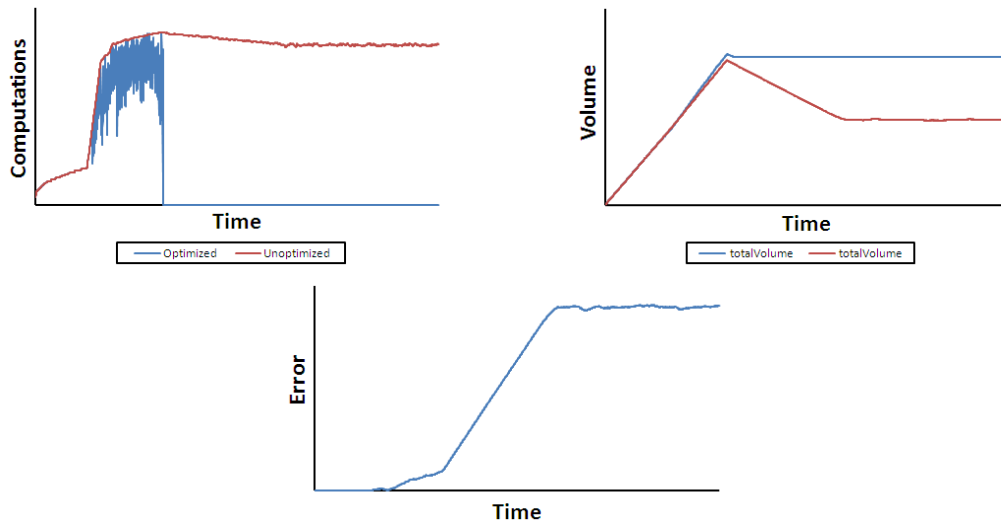


Figure 5.5: Successful results of the downhill case with larger sink than source

Alternatively, a better threshold value can be determined which will produce the desired behavior. In figure 5.2 A source of a value '6' is initially added and sink of value '6' is added some time after with $C_{stable} = 0.007$. The number of computations behaves as expected, however, the error grows over time as can be seen from the error in unoptimised simulation.

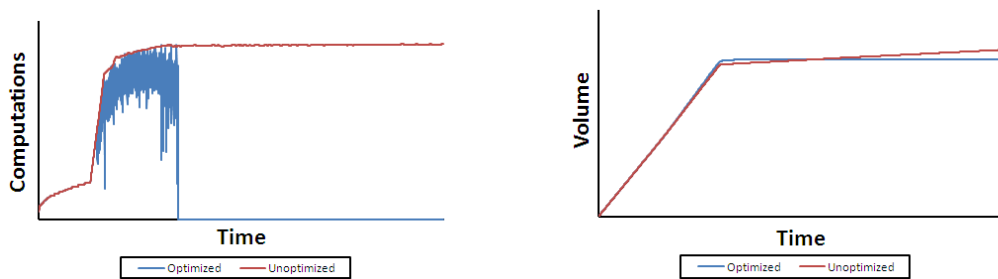


Figure 5.6: Successful results of the downhill case

5.3 The Rough Case

This tests for the presence of any cumulative effect of the region approach and multiple hills. A problem arises where regions stabilise and produce less outflow than inflow consequently the more regions the less flow. The joining of regions creates larger regions. These larger regions cause large spikes in computations when they transition from stable to instable. A source of value '6' is initially added and sink of value '6' is added some time after, with $C_{stable} = 0.01$. The number of computations repeatedly drops and grows as regions stabilise and as new regions form. Finally, once the system stabilises the number of computations. However, there is a subsequent difference in volume between simulations. This is because of a cumulative error caused by stabilisation, when a region stabilises the outflow is frozen. If the outflow is less than the inflow then some volume will be lost. If this occurs across multiple regions the error becomes apparent. See figure 5.9. A possible solution is to fix the outflow of a region to its inflow when it stabilises.

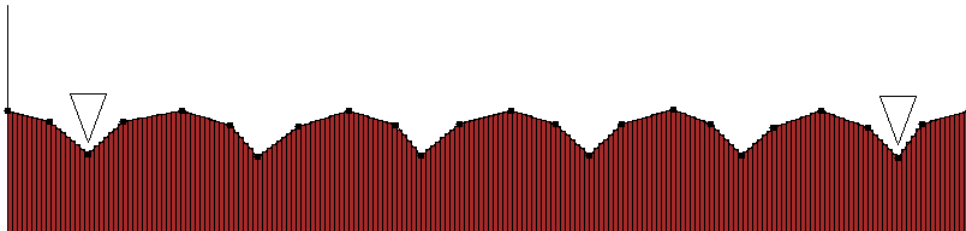


Figure 5.7: A view of the rough case

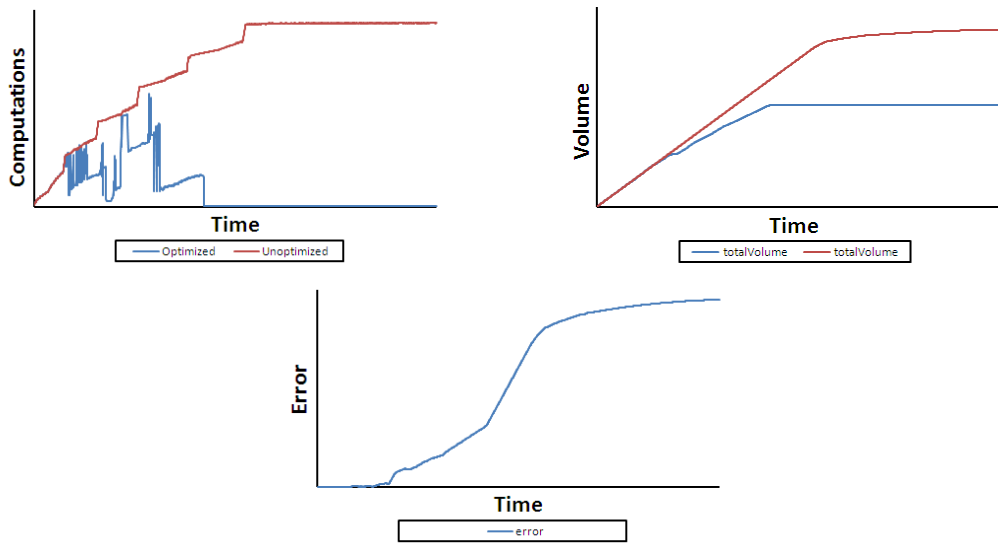


Figure 5.8: Results of the rough case

5.4 The Multiple Downhill Case

This tests for any cumulative effect of the region approach and multiple hills. The test did not yield any useful results. The flow rate of the regions changed too rapidly in too great a magnitude for any region to stabilise for any period of time.

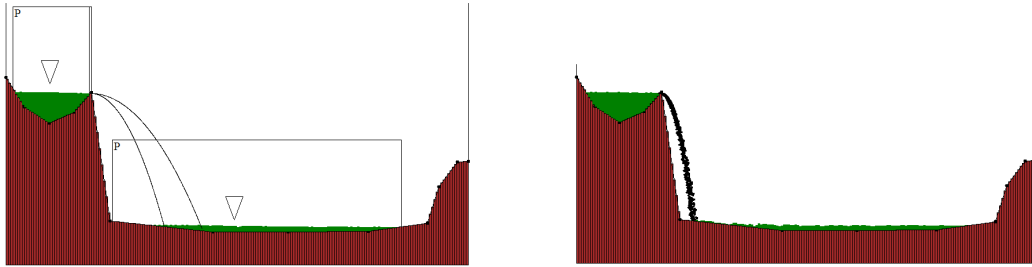


Figure 5.11: Volume approximation of a water fall (left). Particle approximation of water fall (right)

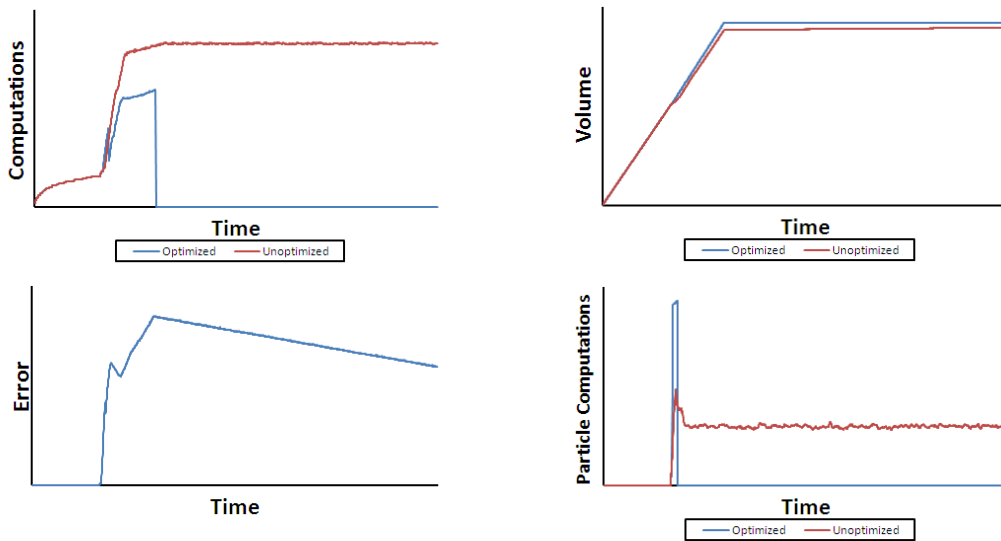


Figure 5.12: Results of the step case

ever, there are issues with actually determining that a region is stable. Errors are introduced by the underlying physical model in the form of increasing volumes and erratic flow; and stabilising regions in the form of a cumulative error. The former can be overcome by using a more correct and robust implementation of a physical model. The second will require a more robust metrics to determine if a region is stable. One such solution could be to calculate the outflow at the boundary and storing that instead of just freezing the region. The waterfall approximation proved to be the best optimisation, cutting computations without introducing significant errors.

Chapter 6

Conclusion

There is a large amount of work done on fluid simulations for Computer Graphics in real-time and off-line. After evaluating a number of simulation techniques for management, a height-field approach was selected. Using the inflow and outflow of water into a region, ineffectual computations were able to be reduced. Test results show that this type of optimisation approach has potential with certain cases showing decreased computations with comparable results to the existing technique being managed. A number of problems were encountered which must be solved before this approach can be considered a robust and useful method. Those being the cumulative error in the flow rate of regions created by reducing computations and the difficulty in determining when a region should be considered stable.

Some possible directions for future work include finding alternate metrics for identifying ineffectual computations. Another one is extending and testing the approach in three-dimensions to see if the concepts carry forward. Lastly, integrating the system with a local graphical simulation to represent waves and particles in stabilised regions. With further studies this approach can be a useful addition to real-time water simulation.

References

- [1] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 46, New York, NY, USA, 2007. ACM.
- [2] Robert Bridson and Matthias Müller-Fischer. Fluid simulation: Siggraph 2007 course notes video files associated with this course are available from the citation page. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 1–81, New York, NY, USA, 2007. ACM.
- [3] Jim X. Chen, Niels da Vitoria Lobo, Charles E. Hughes, and J. Michael Moshell. Real-time fluid simulation in a dynamic virtual environment. *IEEE Comput. Graph. Appl.*, 17(3):52–61, 1997.
- [4] Alain Fournier and William T. Reeves. A simple model of ocean waves. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 75–84, New York, NY, USA, 1986. ACM.
- [5] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–166, New York, NY, USA, 2002. ACM.
- [6] Nathan Holmberg and Burkhard C. Wünsche. Efficient modeling and rendering of turbulent water over natural terrain. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics*

- and interactive techniques in Australasia and South East Asia*, pages 15–22, New York, NY, USA, 2004. ACM.
- [7] Yaohua Hu, Luiz Velho, Xin Tong, Baining Guo, and Harry Shum. Realistic, real-time rendering of ocean waves: Research articles. *Comput. Animat. Virtual Worlds*, 17(1):59–67, 2006.
- [8] Claes Johanson. Real-time water rendering - introducing the projected grid concept. Master’s thesis, Department of Computer Science, Lund University, 2004.
- [9] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH ’90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, New York, NY, USA, 1990. ACM.
- [10] Peter Kipfer and Rüdiger Westermann. Realistic and interactive simulation of rivers. In *GI ’06: Proceedings of Graphics Interface 2006*, pages 41–48, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.
- [11] Marcelo M. Maes, Tadahiro Fujimoto, and Norishige Chiba. Efficient animation of water flow on irregular terrains. In *GRAPHITE ’06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 107–115, New York, NY, USA, 2006. ACM.
- [12] Marcelo M MAES, Tadahiro FUJIMOTO, and Norishige CHIBA. Low-memory and interactive-rate animation of water-column based flows. *The Journal of the Society for Art and Science*, 7(1):1–13, 2007.
- [13] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.
- [14] Jason L. Mitchell. Real-time synthesis and rendering of ocean water. In *ATI Research Technical Report*. Marlboro, MA, 2004.

- [15] David Mould and Yee-Hong Yang. Modeling water for computer graphics. *Computers Graphics*, 21(6):801 – 814, 1997. Graphics in Electronic Printing and Publishing.
- [16] Matthias Müller. Real time fluids in games. <http://www.matthiasmueller.info/talks/gameFluids2007.pdf>, 2007.
- [17] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.
- [18] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *CA '95: Proceedings of the Computer Animation*, page 198, Washington, DC, USA, 1995. IEEE Computer Society.
- [19] Darwyn R. Peachey. Modeling waves and surf. *SIGGRAPH Comput. Graph.*, 20(4):65–74, 1986.
- [20] William T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 359–375, New York, NY, USA, 1983. ACM.
- [21] Joe Stam. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference, March 2003*, 2003.
- [22] Tsunemi Takahashi, Hiroko Fujii, Atsushi Kunimatsu, Kazuhiro Hiwada, Takahiro Saito, Ken Tanaka, and Heihachi Ueki. Realistic animation of fluid with splash and foam. *Computer Graphics Forum*, 22(3):391–400, 2003.
- [23] Jerry Tessendorf. Simulating ocean water. In *SIGGRAPH 2004 Course Notes 31*, 2004.

- [24] Huamin Wang, Gavin Miller, and Greg Turk. Solving general shallow wave equations on surfaces. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 229–238, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

- [25] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Scalable real-time animation of rivers. *Computer Graphics Forum (Proceedings of Eurographics 2009)*, 28(2), mar 2009. to appear.