Improving visual quality of synthetic terrains using weathering simulation on graphics processing unit

Nguyen, Hoang Anh

2008

Nguyen, H. A. (2008). Improving visual quality of synthetic terrains using weathering simulation on graphics processing unit. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/14523>

<https://doi.org/10.32657/10356/14523>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**IMPROVING VISUAL QUALITY OF SYNTHETIC
TERRAINS USING WEATHERING SIMULATION ON
GRAPHICS PROCESSING UNIT**

NGUYEN HOANG ANH
SCHOOL OF COMPUTER ENGINEERING
2008

IMPROVING VISUAL QUALITY OF SYNTHETIC TERRAINS USING WEATHERING SIMULATION ON GRAPHICS PROCESSING UNIT

NGUYEN HOANG ANH

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering

2008

Acknowledgements

This project would not have been possible without the support of many people. Many thanks to my academic supervisor, Associate Professor Alexei Sourin, Nanyang Technological University and my industrial supervisors Mr. Parimal Aswani, from TQ Global, Mr. Doug McNabb and Mr. Mark De Simone, from USA 2XL Games, for their invaluable advice and constant guidance during the last two years. I would also like to thank all my TQ Global colleagues: Fuqian, Quang, Gary, John, Sheikh and Keith, who always suggested creative ideas to me for the erosion algorithm. Last but not least, thanks to my parents and many friends who always give generous support and love.

Table of Content

1	INTRODUCTION	1
1.1	MOTIVATION	1
1.2	OBJECTIVES, SCOPES AND LIMITATIONS	6
1.3	RESOURCE	8
1.3.1	<i>Tool and Software used</i>	8
1.4	REPORT ORGANIZATION	8
2	PHYSICAL GEOGRAPHY BACKGROUND	10
2.1	INTRODUCTION TO PHYSICAL GEOGRAPHY.....	10
2.1.1	<i>Geologic Time</i>	11
2.1.2	<i>The Earth's Crust</i>	11
2.2	WEATHERING AND EROSION	11
2.3	HYDRAULIC EROSION	12
2.3.1	<i>Stream Erosion</i>	13
2.3.1.1	Hydraulic action:	14
2.3.1.2	Abrasion:	15
2.3.1.3	Solution:	16
2.4	DESERT	16
2.4.1	<i>Desert Features</i>	16
2.4.1.1	Streams:	16
2.4.1.2	Flash floods:	17
2.4.1.3	Basin and Range topography:.....	17
2.5	CONCLUSION	18
3	GENERAL PURPOSE COMPUTING ON GPU BACKGROUND.....	19
3.1	INTRODUCTION: WHY GPGPU?	20
3.1.1	<i>Powerful and Inexpensive</i>	20
3.1.2	<i>Flexible, Programmable and Precise</i>	21
3.1.3	<i>Limitations and Difficulties</i>	22
3.2	OVERVIEW OF PROGRAMMABLE GRAPHICS HARDWARE	23
3.2.1	<i>Overview of the Graphics Pipeline</i>	23
3.2.2	<i>Introduction to the GPU Programming Model</i>	24
3.3	HIGH-LEVEL SHADING LANGUAGES	26

3.4	CONCLUSION	27
4	HYDRAULIC EROSION SIMULATION.....	28
4.1	FRACTAL TERRAIN	28
4.2	EROSION TERRAIN	29
4.3	CONCLUSION	32
5	THE PROPOSED EROSION MODEL.....	34
5.1	FLUID DYNAMICS	34
5.2	DESIGN CONSIDERATION	36
5.2.1	<i>Data Structure.....</i>	36
5.2.2	<i>Water distribution.....</i>	37
5.2.3	<i>Acceleration and velocity calculation.....</i>	38
5.2.4	<i>Water movement to only ONE neighbor or all lower neighbors</i>	39
5.2.5	<i>Water movement of all waters or just part of the water</i>	39
5.2.6	<i>Water Evaporation.....</i>	40
5.3	THE PROPOSED EROSION MODEL.....	40
5.3.1	<i>Data Structure.....</i>	40
5.3.2	<i>Water distribution.....</i>	41
5.3.3	<i>Water Simulation</i>	43
5.3.3.1	Acceleration Calculation	45
5.3.3.2	Evaporation	47
5.3.4	<i>Hydraulic Erosion Function</i>	48
5.4	GRAPHICAL ILLUSTRATIONS OF EROSION MODEL	50
5.5	CONCLUSION	53
6	IMPLEMENTATION & EXPERIMENTAL RESULTS.....	54
6.1	PLATFORM AND LANGUAGE.....	54
6.2	CPU IMPLEMENTATION	55
6.2.1	<i>Class Diagram</i>	55
6.2.2	<i>The Erosion algorithm.....</i>	57
6.2.3	<i>CPU version results.....</i>	59
6.2.3.1	1Kx1K Terrain Experiment.....	59
6.2.3.2	2Kx2K Terrain Experiment.....	62
6.3	GPU IMPLEMENTATION	65
6.3.1	<i>Debugger problem.....</i>	65

6.3.2	<i>From CPU to GPU</i>	66
6.3.3	<i>Data Packing</i>	67
6.3.4	<i>Reverse Erosion Algorithm</i>	67
6.3.5	<i>Memory Requirement</i>	69
6.3.6	<i>GPU shading program</i>	70
6.3.6.1	<i>Control Parameters</i>	71
6.3.7	<i>GPU version results</i>	72
6.3.7.1	<i>Statistics Performance</i>	74
6.4	OPTIMIZATION	74
6.4.1	<i>Save data in a smaller type</i>	74
6.4.2	<i>Stop transferring data in between GPU and CPU memory</i>	75
6.4.3	<i>Move neighbor texture coordinate calculation to Vertex Shader</i>	75
6.4.4	<i>Reduce the algorithm's complexity</i>	76
6.5	CHANNEL EROSION	76
6.6	TOOL TO CREATE EROSION ON ANY TERRAIN	79
6.7	OTHER GRAPHIC CARDS TESTING	81
6.7.1	<i>Testing on GeForce 8600 GTS</i>	83
6.7.2	<i>Testing on GeForce 8800 GTX</i>	85
6.7.3	<i>Statistics performance between various GTX graphics cards</i>	86
6.8	PERFORMANCE COMPARISON WITH COMMERCIAL EROSION TOOL	86
6.9	CONCLUSION	90
7	CONCLUSION AND FUTURE WORK	91
7.1	RESEARCH AND DEVELOPMENT RESULTS	91
7.2	PUBLISHED PAPER	93
7.3	SEMINAR PRESENTATION	94
7.4	FUTURE WORK	94
7.4.1	<i>Revisit the State of the art in the erosion field</i>	94
7.4.2	<i>New shading CUDA language</i>	95
7.4.3	<i>Simulate erosion in caves</i>	96
7.4.4	<i>High resolution terrain generation</i>	96
7.4.5	<i>Texture Generation at loading time</i>	97

Summary

Visual simulation of natural eroded terrains using computer graphics has always been a fascinating research field. While many algorithms have been developed to improve terrain visual quality, physically-based hydraulic erosion algorithms have gained much attention in recent years for its ability to generate realistic natural-looking terrains.

In spite of that, most of the algorithms proposed so far have only been tested with low resolution terrains. When these algorithms are used on higher resolution terrains, they can take up to several hours to run, which make them unsuitable for many applications, especially real-time ones. That is the reason why in many graphics applications such as games nowadays, terrain is mostly generated offline and only loaded during game play. This situation will restrict the number of terrains to be stored when we have limited storage capacity.

Recently, graphics hardware has become a new mechanism to improve the speed of computation. This has given us the idea to develop the next generation of General Purpose Computation on Graphics Processing Unit (GPGPU) erosion algorithms for faster terrain generation.

In this thesis, we propose a fast and efficient GPU-based hydraulic erosion procedural technique which runs at interactive rates on today's computers. This procedural technique can be used to abstract low resolution terrain data and generate high resolution realistic terrains on the fly during the application loading time.

This technique is not based on the complicated process of fluid dynamics because simulating that process is computationally expensive which makes the equations inappropriate for large scale terrain simulations in computer graphics. The technique is mainly based on the velocity field of running water. This velocity field is used to determine the erosion

and deposition process, and the sediment transportation process. The algorithm has been carefully designed to be implemented totally on GPU, and thus takes full advantage of the powerfully computational capabilities of current graphics hardware. The results from experiments demonstrate that the proposed method is effective and efficient. It can create realistic erosion effects by rainfall and river flows, and produce fast simulation results for terrains with large sizes.

The major work of this thesis has been published by ACM in the proceedings of Graphite 1-4 December 2007, 5th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, a conference sponsored by SIGGRAPH, ACM Special Interest Group on Computer Graphics and Interactive Techniques [1].

List of Figures

Figure 1: Terrain in the Cities Unlimited game.....	2
Figure 2: Terrain features modeled with fractional Brownian motion	3
Figure 3: Fuerteventura Canary Islands.....	4
Figure 4: Panorama of "The Walls of China", the eroded lunette on the eastern side of Lake Mungo, carved into fantastic shapes by the occasional rain.....	4
Figure 5: Eroded Mountain from latest erosion paper in 2005. [8]	6
Figure 6: An example of large terrains for today application terrains.....	7
Figure 7: Grand Canyon erosional landscape	13
Figure 8: An illustration of hydraulic action	14
Figure 9: An example of running water on the mountain slope in Zion National Park.....	15
Figure 10: Sahara desert. Erosion sculpts the landscapes.....	17
Figure 11: The programmable floating-point performance of GPUs has increased dramatically over the last five years when compared to CPUs. [17].....	21
Figure 12: Graphics Pipeline	24
Figure 13: Eroded Mountain from latest erosion paper in 2005. [8]	32
Figure 14: Two important cases for the water transportation	39
Figure 15: An example of Hammersley Point Set rain distribution	43
Figure 16: Acceleration Calculation	46
Figure 17: Sediment Erosion and Deposition Calculation.....	50
Figure 18: Water distribution over the terrain.	51
Figure 19: Water movement from high positions to lower positions.	51
Figure 20: Water moves down and erodes the higher positions of the terrain.	52
Figure 21: Water evaporates and leaves the deposition mostly on the terrain bottom	52
Figure 22: Erosion Model Class Diagram	56

Figure 23: Original 1024x1024 Synthesis Terrain	58
Figure 24: Eroded Synthesis Terrain after 50 steps in 1 min and 7 seconds	60
Figure 25: Eroded Synthesis Terrain after 100 steps in 1 min and 22 seconds (deposition is slightly more).	61
Figure 26: Original Terrain.....	62
Figure 27: Eroded Terrain with Deposition.....	62
Figure 28: 2Kx2K eroded terrain when using the same parameters with 1Kx1K simulation.....	63
Figure 29: 2Kx2K eroded terrain when increasing Sediment Capacity KC with 1Kx1K simulation (100 steps).....	64
Figure 30: 2Kx2K eroded terrain when increasing Sediment Capacity KC with 1Kx1K simulation (150 steps).....	65
Figure 31: CPU - GPU Data Structure Analogy	66
Figure 32: Water flow calculation between normal and reverse erosion algorithm.....	67
Figure 33: Shading Program	71
Figure 34: Comparison between 2Kx2K eroded terrains from CPU and GPU	73
Figure 35: Erosion with Deep and Long Gullies	77
Figure 36: Step 1 - Creating the deep and long gullies on the terrains.....	78
Figure 37: Step 2 - Running erosion simulation after that.....	78
Figure 38: A closer look of the channel erosion result.	79
Figure 39: GPU-based erosion tool	80
Figure 40: Original Terrain in Color Mode	82
Figure 41: Eroded Terrain in Color Mode	82
Figure 42: Eroded Terrain in Texture Mode.....	83
Figure 43: Erosion result if another graphics application is not concurrently used.	84
Figure 44: Erosion result if another graphics application is concurrently used.....	85

Figure 45: World Machine erosion tool.....	87
Figure 46: World Machine erosion result	88
Figure 47: Time taken to simulate erosion on 1Kx1K terrain in World Machine.....	88
Figure 48: Time taken to simulate erosion on 2Kx2K terrain in World Machine.....	89
Figure 49: Delicate Arch, located at Arches National Park in Moab, Utah. The LaSalle Mountains are seen in the background.	96
Figure 50: Texture is used in current game	97
Figure 51: Texture applied on mountain cliff.....	98

List of Tables

Table 1: Data grids used in simulation	41
Table 2: Material constants and their default values	48
Table 3: GeForce 7900 GS specification.....	55
Table 4: Erosion Simulation Time Comparison	64
Table 5: Erosion Simulation GPU data packing	67
Table 6: GPU Memory Requirement.....	70
Table 7: Time Comparison between CPU and GPU version.....	74
Table 8: GeForce 8600 GTS specification	83
Table 9: GeForce 8600 GTS specification	85
Table 10: Time Comparison among 7900 GS, 8600 GTS and 8800 GTX graphics card ..	86
Table 11: Time Comparison between erosion algorithm on GPUs and World Machine ..	89

CHAPTER 1

1 Introduction

1.1 Motivation

Amongst the visual elements in graphics applications, there is no doubt that terrain plays an important role. It is a key feature of such applications as aerial and land traffic training and control software, architecture design, flight simulators, battleground simulations, feature film special effects, and computer games, and is essential to the creation of a realistic feel for these and other forms of simulations. Much research work has thus been placed in the field of efficient and realistic terrain modeling techniques, and the progress made has allowed artists and designers to generate believable synthetic terrains for these applications. An example of such work is shown in Figure 1, which shows a realistic terrain generated by the game Cities Unlimited.

In order to create a good foundation for realistic terrain simulation, understanding how the earth's geology process works is inevitably the first step. Natural terrains exhibit a lot of complicated shapes due to 2 main processes: 1) massive, catastrophic changes such as earth quakes and floods, which are infrequent and sporadic, but which can cause rapid dramatic changes to the landscape, and 2) small and slow changes such as hydraulic erosion caused by rainstorms over a period of thousands of years.

Figure 1: Terrain in the Cities Unlimited game.

The rapid development of dedicated and powerful graphics hardware, as well as the fast and fully programmable graphics processing units available on most graphics cards today have resulted in the expanding possibilities for more real-looking objects and terrain in applications of today.

In this thesis, we will be concerned exclusively with terrain modeling using hydraulic erosion simulation. Hydraulic erosion is the most noticeable effect that molds terrain landscape [2, 3]. More specifically, we are considering a realistic geometry representation of a terrain at a very large scale, and reproducing the details on this terrain surface caused by hydraulic erosion.

Previously, there were two main works on terrain synthesis: fractal terrain modeling and physical hydraulic erosion. At first glance, fractal terrains are convincing imitations of

natural mountain terrains. The following picture shows an example of fractal features in mountain.

Figure 2: Terrain features modeled with fractional Brownian motion

Courtesy of R.V. Voss and B.B. Mandelbrot, adapted from [4, 5].

However, closer examination reveals an unnatural characteristic in these surfaces. Specifically, fractal terrains have no global erosion features due to isotropy [6, 7]. Figure 3 is a Fuerteventura canary island where there is a lot of erosion caused by running water. Figure 4 shows the eroded lunette on the eastern side of Lake Mungo which was carved into fantastic shapes by the occasional rain. These are examples of eroded terrains which fractal terrain modeling can not represent.

Figure 3: Fuerteventura Canary Islands

Source: <http://www.canaryventura.com>

Figure 4: Panorama of "The Walls of China", the eroded lunette on the eastern side of Lake Mungo, carved into fantastic shapes by the occasional rain.

Source: Don's Maps, <http://donsmaps.com/mungowallschina.html>

The physically-based hydraulic erosion methods which were introduced later were geared towards solving this problem of fractal landscapes' inability to simulate global erosion features. These hydraulic erosion methods were not only able to create erosion ridges, canyons and gullies, but could also add fractal details to the terrain.

Although most of hydraulic erosion terrain synthesis methods proposed so far produce realistic results, they still have limitations. Firstly, the methods normally have many physically-based parameters, which make it hard for users to control terrain generation toward a desired style. Secondly, most of the current methods are only tested with small or low resolution terrains. Figure 5 shows the eroded result from one of the latest papers in erosion by Neidhold *et al.* in 2005 [8]. As we can see, the authors only tested the algorithm in a terrain with one mountain in 256x256 grid cells. The results were implemented in .NET framework, C# and tested on a 2.4 GHz Pentium IV PC; however, the algorithm still runs at 4 frames per second. The author explains that this is because the process is computationally expensive.

Simulating current hydraulic erosion method on larger terrains, such as 2048x2048 grid cells, using these methods takes a long time to produce realistic results. Furthermore, most of the terrains are generated offline and this provides very little flexibility for usage in real-time applications. Figure 6 shows an example of large terrain which, to our knowledge, no current methods are capable of simulating hydraulic erosion on.

Fortunately, there have been vast improvements in graphics hardware computational performance recently [9]. In addition, graphics hardware also features improved programmability, so much so that they have become an ideal platform for computationally demanding tasks in many application domains. This has led us to come up with the novel solution of using the General Purpose Computation on Graphics Processing Unit

(GPGPU) to develop the next generation of GPGPU erosion algorithms for faster terrain generation.

Figure 5: Eroded Mountain from latest erosion paper in 2005. [8]

1.2 Objectives, Scopes and Limitations

This project is one of many research projects supported by the collaboration between Nanyang Technological University, TQ Global Singapore and 2XL Games, USA. The objective of this project is to propose a fast procedural technique, which utilizes the powerful computational capabilities of graphics hardware, in order to allow applications to ship with lower resolution height map data, such as United States Geological Survey (USGS) Digital Elevation Model (DEM) data, and perform erosion at load time, thus generating a highly detailed high-resolution terrain from the height map data.

One of the most valued features of the procedural technique is abstraction. Rather than explicitly specifying and saving all terrain data in high resolution, we abstract them into lower resolution terrain data and utilize that procedure when needed [10]. A storage saving is obtained, as the details are already implicit in the procedure. For example, a stored 256x256 terrain saves 64 times the storage amount required to store 2048x2048 terrains. This also allows us to create inherently multi-resolution eroded terrain models that we can then expand to the resolution needed.

Another advantage of procedural technique is parametric control. A few parameters provide large amounts of different controls pertaining to erosion. This parametric control relieves the user from the burden of low-level control and specification of every detail. Procedural models also offer flexibility. Various types of eroded terrains can be generated at different times based on the parameters.

While there are many types of erosion, such as wind erosion, thermal erosion and others, we are only focusing on hydraulic erosion which is caused by running water as this is the most noticeable effect in most landscape.[2]

Figure 6: An example of large terrains for today application terrains.

1.3 Resource

1.3.1 Tool and Software used

World Machine [11] is a unique procedural terrain generator that combines powerful terrain effects, a graphical user interface, and fast, dynamic previews to allow for the rapid creation of extremely varied and realistic terrain. In this project, World Machine is used as a renderer for our terrain models.

3D Studio Max 8.0 is a powerful 3D rendering and modeling software which lets game developers, design visualization professionals, and visual effects artists maximize their productivity and tackle challenging animation projects. This version of 3D Studio Max was developed by Discreet and has features such as editable meshes or polygons. In this project, 3D Studio Max is used to generate the triangulation mesh from the eroded terrain heightmap to be loaded in the game.

Adobe Photoshop is a very user-friendly digital imaging tool. It is a popular application among professional graphics artists. All of the photos and textures used in this project are edited using Adobe Photoshop.

1.4 Report Organization

This report is organized as follows:

Chapter 1 provides a brief introduction to terrain generation methods and the motivations behind this research work.

Chapter 2 explains some of the natural geographical processes which shape the earth's terrain. This will give us the knowledge of how terrain surface are naturally created.

Chapter 3 gives the motivation for using General Purpose Computation on Graphics Processing Unit (GPGPU). This chapter discusses in detail the advantages and disadvantages of using GPGPU to develop next generation shading algorithms.

Chapter 4 reviews the state of the art techniques in terrain generation and improving terrain visual quality.

Chapter 5 presents the author's erosion model design. Water distribution, water simulation, velocity and acceleration calculation as well as the evaporation process will be explained in detail.

Chapter 6 describes how the author implemented his design in CPU C++ code and GPU High Level Shading Language code. Comparisons between the two implementations, their advantages, disadvantages and computational times will be also described.

Chapter 7 concludes the report by reviewing the proposed method and conceptual models involved. Future work and the authors' suggestions for improvements are also included in this chapter.

All citations referred to in this report are included in the References.

CHAPTER 2

2 Physical Geography Background

Our landscape is formed by many natural phenomena, from massive and sporadic changes such as earthquakes, volcanoes to small and slow ones like rainfalls. The process that shapes our terrains, our landscape is the process of how the Earth has been formed for millions of years and is still changing.

The purpose of this chapter is to set the knowledge foundation in geography for subsequent designing and implementation chapters. This chapter will look at the Earth from a geologist point of view and describe the fundamentals of how the Earth's surface is shaped. We start this chapter by an introduction to physical geography, the Earth's geologic time and numerous external forces which affect the Earth's surface in section 2.1. Section 2.2 will describe the weathering and erosion process, how the earth surface is being changed as a result of interaction with many factors. Section 2.3 explains the geographical side of hydraulic erosion phenomena and last but not least, section 2.4 will describe the desert which is the main game play terrain discussed in this paper.

2.1 Introduction to Physical Geography

The Earth is always changing. Small, slow changes are continuous, while massive but rapid changes, like those produced by earthquake, hurricanes, are sporadic. The discipline that studies these changes, past and present, is geology [6].

Physical Geography Background

Physical geology is the study of the Earth's rocks, minerals, and soils and how they have been shaped through time. Complex interior processes such as plate tectonics and mountain-building have changed massively the Earth's surface bringing forth rocks found in the Earth's interior to its exterior. External processes such as glaciations, running water, weathering, and erosion have also contributed to the shaping process of our landscapes today.

2.1.1 Geologic Time

Geologists have anticipated the Earth to be about *4.6 billion years old* [6]. At the initial stages, as the crust cooled, early geologic processes were volcanic, structuring up the continental crust. Bacterial forms of life have been found in rocks that are billions of years old. Complex oceanic organisms began to emerge only about 600 million years ago. From about 66 million to 245 million years ago, dinosaurs and other reptiles flourished all over the world. In contrast, human beings have existed in only about the last 2 million years, less than a thousandth of the age of Earth.

2.1.2 The Earth's Crust

Numerous external forces affect the Earth's surface. Freezing, thawing, and running water all contribute to *weathering* and *erosion*, the processes that break rock down into small material particles. These particles are then dislodged and transported by running water, ice, or wind as *sediment*. The processes of erosion reduce mountains to hills, create canyons, valleys, gullies and deposit huge amounts of sediments.

2.2 Weathering and Erosion

When exposed to the atmosphere, no rock escapes the effects of weathering. The process of *weathering* changes rocks at the Earth's exterior and breaks them down over time into

Physical Geography Background

loose and tiny particles of sediment and soil. Weathering is the result of the interaction of many factors such as air, water, and heat which erodes the rock.

Erosion is the shifting of the small particles by ice, wind, or water. The particles are then *transported* by that agent until they are deposited to form sedimentary deposits, which can be later eroded again or transformed into sedimentary rocks. Weathering is generally a long, slow process that is continuously active at the Earth's surface. Mechanical and chemical are the two main types of weathering.

Mechanical weathering (or physical weathering) is the process by which rocks are broken down into smaller pieces by external conditions, such as the freezing of water in cracks in the rock. The rock is chemically weathered when there is a chemical reaction between the rock and the agent, especially water, to form new chemical combinations.

The effects of chemical weathering are most distinctive in areas where both precipitation and temperature are high. Chemical weathering diminishes the bonds in rocks and makes them more open to erosion. The most obvious effect is the staining of the surface which results from the breakdown of minerals.

2.3 Hydraulic Erosion

This section will explain the main phenomenon that we are going to simulate, that of hydraulic erosion. Stream erosion will also be described in details. The three main ways that streams erode the Earth's surface: hydraulic action, abrasion, and solution are also discussed while our simulation are primarily focusing on hydraulic action and solution.

The *hydrological cycle* is the continuous water exchanging process between the atmosphere, land, and ocean. It is the circulation of the Earth's water through precipitation, evaporation, and transpiration (the release of water into the atmosphere by plants).

Physical Geography Background

Running water is the most vigorous agent to produce landforms on the Earth's surface. Waterways erode, transport, and deposit rock and sediment to shape the landscape such as canyons, valleys, deltas, alluvial fans, and floodplains.

Any flow of water within a natural channel regardless of size is called stream. A stream's headwaters are where the stream starts, often from the higher elevations of mountainous terrain. The stream runs downward through lower elevations to its terminus, where it enters another stream, lake, or ocean. This terminus is called the mouth of the stream.

2.3.1 Stream Erosion

Erosion by water starts even before a distinctive stream has formed. It happens in two ways: by collisions as raindrops hit the ground and by overland flow during heavy rains. One raindrop does not have much effect, but the number of raindrops is so great that together they can realize a large amount of erosion. Erosion landscapes such as the Grand Canyon, Figure 7, have been created by constant erosion from running water over millions of years.

Figure 7: Grand Canyon erosional landscape

Besides eroding the bedrock and earlier deposited sediments along its route, a stream continuously weathers mechanically and chemically the individual rock and soil particles carried by its water. Hydraulic action, abrasion, and solution are the three main ways that streams erode the Earth's surface.

2.3.1.1 Hydraulic action:

The ability of flowing water to dislodge and transport rock particles or sediment is called hydraulic action. In general, the greater the velocity of the water and the steeper the grade of erosion are, the greater the hydraulic action capabilities of the stream is. Hydraulic action is also enhanced by a rough and irregular stream bottom. Figure 8 is an illustration of hydraulic action where the flowing water dislodges and transports the sediment from higher terrain positions to lower terrain position and mostly deposits the sediments at the bottom of the terrain. Figure 9 shows an example of flowing water on a mountain slope in Zion National Park of the United States.

Figure 8: An illustration of hydraulic action

2.3.1.2 Abrasion:

Abrasion is the process by which a stream's uneven bed is smoothed by the constant friction and scouring impact of rock fragments and sediment carried in the water. The individual particles of sediment also collide as they are transported, breaking them down into smaller particles. Generally the more sediment a stream carries is, the greater the amount of erosion of the stream bed. The heavier, coarser sediment strikes the stream bed more frequently and with more force than the smaller particles, resulting in an increased rate of erosion.

Figure 9: An example of running water on the mountain slope in Zion National Park.

2.3.1.3 Solution:

Rocks vulnerable to the chemical weathering process of solution can be dissolved by the slightly acidic water of a stream. Limestones and sedimentary rock cemented with calcite are vulnerable to solution. The dissolution of the calcite cement frees the sedimentary particles, which can then be picked up by the stream's flow through hydraulic action.

2.4 Desert

The term desert is now generally used as a synonym for land where annual rainfall is less than 25 centimeters (10 inches) or in which the evaporation rate exceeds the precipitation rate. Deserts are dry with sparse vegetation. Terrains tend to have sharp features because the lack of rain results in minimal chemical weathering, and flash floods create steep-walled scarps and gullies.

There are few plants to keep the soil from the wind, so the soil is blown away to expose rocky surfaces. Even in such a dry climate, most of the landforms are carved by the rare periods of heavy rainfall that result in flash floods, erosion, and sediment deposition.

2.4.1 *Desert Features*

2.4.1.1 Streams:

Because of the dry conditions, most deserts do not have streams or rivers that run all year long. Streams that flow irregularly as a result of periods of sudden rainfall are called ephemeral streams. Exceptions are the Colorado River in the southwestern United States and the Nile River in Egypt, which originate in mountainous regions and have enough stream flow to cross desert areas.

Deserts often exhibit an interior drainage pattern where streams empty into landlocked basins. The basins become temporary sources of water, and evaporation can precipitate salt beds and other evaporitic minerals.

2.4.1.2 Flash floods:

Most desert rainfall comes from short, violent thunderstorms. The rain is so heavy that it cannot soak into the hard ground or be controlled by the narrow stream channels. Rain flows rapidly over the land, creating flash floods in the stream beds that can be very destructive in populated areas. Lack of vegetation results in severe erosion carves new valleys and gullies. During these floods, the water can become so choked with sediment it becomes a mudflow. The rapid down cutting by floodwater produces narrow gorges with steep walls and gravel bottoms called arroyos or dry washes. Figure 10 is a photo of Sahara desert where we can see erosion sculpts the landscapes. This is an example of the desert terrain which has been shaped as a result of flash floods.

Figure 10: Sahara desert. Erosion sculpts the landscapes.

2.4.1.3 Basin and Range topography:

Several deserts display Basin and Range topography which is a series of steep mountain ranges separated by broad valleys. The mountain ranges results in a series of rain shadows that form the desert climate. If the water flows down further into the center of the

valley, it forms a playa lake. Typically thin and muddy from clay, it evaporates quickly, leaving a flat, hard, dried clay surface that is broken by desiccation cracks. If the water carries dissolved salts, salt flats will result.

2.5 Conclusion

This chapter has given us the foundation of how terrain is shaped through millions of years of weathering. The natural complex shapes of the terrains that we have seen so far are the result of 4.6 billion years of natural erosion. Although numerous external forces can affect the Earth's surface, running water is the most vigorous agent which shapes the landforms on the Earth. This concludes the importance of hydraulic erosion, or the erosion due to running water. Hydraulic action, abrasion, and solution are the three main ways that running water can erode the landscape. In general, the greater the velocity of the water and the steeper the grade of erosion, the greater the hydraulic action capabilities of the stream. A stream's uneven bed can be smoothed by abrasion, the constant friction and scouring impact of rock fragments and sediment carried in the water. Lastly, streams with slightly acidic water can solute the sediment which is vulnerable to the chemical weathering. Of the three processes mentioned, the hydraulic action and solution processes are the key elements that we focus on in our hydraulic erosion simulation.

The chapter also explains how terrain in deserts can exhibit erosion features even though there is not much rainfall throughout the year. Landscape can be carved by the rare periods of heavy rainfall which result in flash floods that cause hydraulic erosion and sediment deposition. In the next chapter, we will review the motivation of General-Purpose computing on Graphics Processing Units (GPGPU), and the reasons why we choose to implement our erosion model on graphics hardware.

CHAPTER 3

3 General Purpose Computing on GPU Background

In the previous chapter, the geographical processes that shape the earth have been presented. That gave us the basic scientific knowledge on how terrain surfaces are created, so as to aid in our understanding of erosion simulation. This chapter will look at the erosion process from the implementation point of view. While the erosion algorithm can be implemented on and processed by the CPU, developing a slightly different erosion technique and porting the code to be processed by graphics hardware can give us a tremendous speed advantage.

General-Purpose computing on Graphics Processing Units (GPGPU) is the method utilizing commodity Graphics Processing Units (GPUs), which traditionally handles computation only for computer graphics, to perform computation in various applications handled by the CPU. It is made achievable by adding to the rendering pipeline the programmable stages and higher precision arithmetic, which permits researchers, developers to use stream processing on non-graphics data. This chapter starts with the technical motivations of GPGPU and describes the hardware and software developments which have led to the recent attention in this field in section 3.1.

Section 3.2 will describe an overview of programmable graphics hardware in which there will be an introduction to GPU programming model. The techniques used in mapping general-purpose computation to graphics hardware will be also explained. Last but not least, section 3.3 will give a short introduction of high level shading language. The

material in the chapter is referenced from many sources on information on GPGPU [9, 12, 13]. All of this information will be generally very useful for researchers who intend to develop the next generation of GPGPU algorithms.

3.1 Introduction: Why GPGPU?

GPUs are perhaps the most robust computational hardware nowadays [9]. The fast improvement in computational power has made GPUs a convincing platform for computationally demanding tasks in a wide variety of application domains. We begin by summarizing the motivation and challenges of general-purpose GPU computing. Why GPGPU?

3.1.1 *Powerful and Inexpensive*

The first reason to move into graphics hardware is its computational power and inexpensiveness but tremendous memory bandwidth. For example, the flagship NVIDIA GeForce 8800 GTX (\$550 as of February 2007) boasts 55.2 GB/sec memory bandwidth and can sustain a measured 330 GFLOPS, measured with GPUBench [12, 14]. Compare to 21GB/sec and 48GFLOPS theoretical peak for the 3.0 GHz Intel Core2 Quad (QX6850) [12, 15]. The arithmetic throughput of NVIDIA's current-generation launch product, the GeForce 8800 GTX (330 GFLOPS), is more than two times that of its predecessor, the GeForce 7800 Ultra (165 GFLOPS). In general, the computational capabilities of GPUs, measured by the traditional metrics of graphics performance, have compounded at an average yearly rate of 1.7 (pixels/second) to 2.3 (vertices/second). This rate of growth much outpaces the usually-quoted Moore's Law as applied to traditional microprocessors; compare to a yearly rate of roughly 1.4 for CPU performance [16] (Figure 11).

Figure 11: The programmable floating-point performance of GPUs has increased dramatically over the last five years when compared to CPUs. [17]

The reason graphics hardware performance is emerging faster than that of CPUs is that CPUs are optimized for high performance on sequential code, with many transistors dedicated to extracting instruction-level parallelism with techniques such as branch prediction and out-of-order execution. On the other hand, the highly data-parallel nature of graphics computations makes it easier for GPUs to use additional transistors more directly for computation, achieving higher arithmetic intensity with the same transistor count. Also, the multi-billion dollar video game market is a pressure cooker that drives innovation to exploit this property.

3.1.2 Flexible, Programmable and Precise

Modern GPUs now include intensely programmable processing units that support vectorized floating-point operations for programmable pixel, vertex and geometry engines.

Solid high level languages such as HLSL have emerged to support this new programmability of the vertex and pixel pipelines.

GPU functionality has traditionally been very limited. However, modern GPUs support real precision. The single IEEE 32 bit floating point is high enough for many applications and the double IEEE 64 bit are also going to be released soon. DirectX 10 class GPU added 32 bit integers.

Therefore, in general, the power and flexibility of GPUs makes them an ideal platform for general-purpose computation. Example applications range from in-game physics simulation to conventional computational science. Our goal is to make this inexpensive power of the GPU available as a computational co-processor. But the GPU is hardly a computational panacea. [9]

3.1.3 Limitations and Difficulties

The increasing flexibility of GPUs, together with some creative uses of that flexibility by GPGPU developers, has enabled many applications outside the original narrow tasks for which GPUs were originally designed to harness this power. However, there are still many limitations in the GPU's unusual programming model.

It remains tricky to apply non-graphics task to dedicated graphics hardware. The programming model is very unusual and programming idioms are tied to computer graphics. Underlying graphics architectures are still largely secret. Efficient GPGPU programming is not simply a matter of learning a new language and it is not as simple as just porting CPU code to GPU code. Instead, the computational algorithm must be transformed into graphics terms by a programmer familiar with the design, limitations, and evolution of the underlying hardware.

Despite these programming challenges, the potential benefits in terms of computational capabilities, and a growth which is much faster than traditional CPUs are too large to ignore [18].

3.2 Overview of Programmable Graphics Hardware

This section summarizes the development of the GPU and describes its current hardware and software.

3.2.1 *Overview of the Graphics Pipeline*

Interactive 3D graphics applications of today require high amounts of computation and have substantial parallelism. Designing custom graphics hardware that takes advantage of this parallelism permits better performance on graphics applications than can be attained on more traditional microprocessors.

All GPUs organize their graphics computation into structures known as graphical pipelines. These pipelines are separated into several stages to allow hardware implementations to maintain high computational rates through parallel execution. A block diagram of the programmable graphics pipeline is shown in Figure 12. All geometric primitives pass through each stage: vertex operations, primitive assembly, rasterization, fragment operations, and composition into a final image. Each stage is implemented as a separate piece of physical hardware on the GPU in what is termed a task-parallel machine organization.

Figure 12: Graphics Pipeline

3.2.2 Introduction to the GPU Programming Model

GPUs are a convincing solution for applications that require high arithmetic rates and data bandwidths. This section briefly introduces the GPU programming model using both graphics API terminology and the terminology of the more theoretical stream programming model, because both are common in literature.

The stream programming model exposes the parallelism and communication patterns intrinsic in applications by constructing data into streams and expressing computation as arithmetic kernels that work on streams. Purcell *et al.* [19] differentiate their ray tracer in the stream programming model; Owens [20, 21] and Lefohn *et al.* [22] discuss the stream programming model in the context of graphics hardware, and the Brook programming system [23] presents a stream programming system for GPUs.

General Purpose Computing on GPU
Background

Common scenes have more fragments than vertices, in modern GPUs the programmable stage with the highest arithmetic rates is the fragment stage. A typical GPGPU program uses the fragment processor as the computation engine in the GPU. Such a program is structured as follows [21, 24, 25]:

- ❖ First, the application is separated into independent parallel sections. Each of these sections can be considered a kernel and will be implemented as a fragment program. The input and output of each kernel program is stored in textures in GPU memory. In stream processing terms, the data in the textures are comprised of streams, and a kernel is invoked in parallel on each stream element.
- ❖ To invoke a kernel, the size of the output stream must be indicated. A typical GPGPU invocation is a quad oriented parallel to the image plane, sized to cover a rectangular region of pixels matching the desired size of the output array. GPUs excel at processing data in two-dimensional arrays, but are limited when processing one-dimensional arrays.
- ❖ The rasterizer generates a fragment for every pixel location in the quad, producing thousands to millions of fragments.
- ❖ Each of the generated fragments is then processed by the same active kernel fragment program. The fragment program can read from arbitrary global memory locations (with texture reads) but can only write to memory locations corresponding to the location of the fragment in the frame buffer (as determined by the rasterizer). The domain of the computation is specified for each input texture (stream) by specifying texture coordinates at each of the input vertices, which are then interpolated at each generated fragment. Texture coordinates can be specified independently for each input texture, and can also be computed on the fly in the fragment program, allowing arbitrary memory addressing.
- ❖ Output of the fragment program is a color value (or vector of values) per fragment. This output may be the final result of the application, or it may be stored as a texture and

then used in additional computations. Complex applications may require several or even dozens of passes (“multipass”) through the pipeline.

While the complexity of a single pass through the pipeline may be restricted, for example, by the number of instructions, by the number of outputs allowed per pass, or by the limited control complexity allowed in a single pass, using multiple passes allows the implementation of programs of arbitrary complexity.

3.3 High-Level Shading Languages

Most high-level GPU programming languages today share one thing in common: they are designed around the idea that GPUs generate pictures. As such, the high-level programming languages are often referred to as shading languages. That is, they are a high-level language that compiles a shader program into a vertex shader and a fragment shader to produce the image described by the program. Cg [24], HLSL [25, 26], and the OpenGL Shading Language [27] all abstract the capabilities of the underlying GPU and allow the programmer to write GPU programs in a more familiar C-like programming language. They do not stray far from their origins as languages designed to shade polygons. All retain graphics-specific constructs: vertices, fragments, textures, etc. Cg and HLSL provide abstractions that are very close to the hardware, with instruction sets that expand as the underlying hardware capabilities expand. The OpenGL Shading Language was designed looking a bit further out, with many language features (e.g. integers) that do not directly map to hardware available today.

Sh is a shading language implemented on top of C++ [28]. Sh provides a shader algebra for manipulating and defining procedurally parameterized shaders. Sh manages buffers and textures, and handles shader partitioning into multiple passes. Sh also provides a stream programming abstraction suitable for GPGPU programming. Finally, Ashli [29]

works at a level one step above that of Cg, HLSL, or the OpenGL Shading Language. Ashli reads as input shaders written in HLSL, the OpenGL Shading Language, or a subset of RenderMan. Ashli then automatically compiles and partitions the input shaders to run on a programmable GPU.

3.4 Conclusion

In conclusion, despite many of the limitations of the GPU's unusual programming model and the difficulty of applying this model to non-graphics tasks, vast improvement in computational power has made the GPU an ideal platform for computationally challenging tasks in a wide variety of application domains. The biggest motivation to use the GPU is its computational power and tremendous memory bandwidth. The computational capabilities of GPUs have compounded at an average yearly rate of 1.7 pixels/second and 2.3 vertices/second, compare to a yearly rate of roughly 1.4 for CPU performance. Modern GPUs now consist of intensely programmable processing units that hold up vectorized floating-point operations for programmable pixel, vertex and geometry engines and also support real precision. This allows us to program more easily on current generation GPUs than on previous ones. In chapter 4, we will review state of the art techniques in terrain generation and improving terrain visual quality.

CHAPTER 4

4 Hydraulic Erosion Simulation

The previous two chapters have given us a good foundation on geography and general purpose computing on GPUs. In this chapter, we will look at the latest development in the field of hydraulic erosion simulation, as well as explain why there is a need for improvements to the current methods available.

During the past decade, many algorithms have been developed to create efficient models to approximate natural terrain. Euclidean-geometry methods are amongst the most popular ones used to create objects in computer graphics. However, these methods are only adequate for objects with smooth surface and regular shapes. Natural objects such as mountains and clouds, on the other hand, have irregular or fragmented features. Thus, Euclidean methods do not provide realistic representations of such objects [30, 31]. In contrast, natural objects can be more realistically described with fractal-geometry methods.

To generate a synthetic terrain, there are two main approaches which have been proposed so far: fractal landscape modeling and physical erosion simulation. Section 4.1 will give an introduction on state on the art technique to generate fractal terrains and section 4.2 will review our main topic, the most recent techniques to generate hydraulic erosion.

4.1 Fractal terrain

Fractal landscape modeling dates back to the pioneering work of Mandelbrot [4]. Since then, a variety of stochastic subdivision techniques have been introduced. Fournier *et al.*

[32] introduced the random midpoint displacement technique to create fractal surfaces. Voss [5] added successive random displacement to fractional Brownian surfaces. Miller [33] proposed a square-square subdivision scheme for generating a fractal terrain and a parallel processing algorithm for rendering height fields. Lewis [34] proposed a generalized stochastic subdivision. Szeliski and Terzopoulos [35] addressed the problem of user control by combining deterministic splines and stochastic fractals into constrained fractals. Sala *et al.* [36] presented applications of mathematics in the real world which is also based on fractal to describe a terrain. Recent fractal-based approaches are reviewed in [3].

4.2 Erosion Terrain

Fractal techniques used to be considered as the most efficient methods for creating realistic-looking terrains. However, fractal terrains in general have no global erosion features due to isotropy and stationary [7] which is the major processing affecting the configuration and evolution of terrain. Many algorithms introduced later used physically-based approaches to simulate erosion and deposition process and created a more realistically-looking terrain. One of the first fractal based algorithms for hydraulic erosion was introduced by Kelley *et al.* [37]. The model developed created topographic structure based on a pregenerated drainage network. The model creates an approximation to natural terrain by simulating the erosion of stream networks on an initially uneroded surface. Empirical models of stream erosion were used as a basis for the model. Stream networks of various sizes and shapes were created by the model from a small number of initial parameters. The eroded surface is represented as a surface under tension, using the tension parameter to shape the profiles of valleys created by the stream networks.

Another technique that also needs a pregenerated river network was published by Nagashima [38]. The pregenerated river network is created with a two-dimensional fractal

function. Then, the river banks are eroded by some physically inspired rules. In 2005, Belhadj *et al.* [39] introduced a bi-process method which also models realistic terrains with ridge line and drainage network. As opposed to methods that generate terrains constrained by a predefined set of drainage network, many other algorithms simulated the erosion process directly on a given terrain model. The first algorithm to approach this issue was described by Musgrave *et al.* [7]. Musgrave suggested a global and physical erosion processes which generate both local and global erosion features through a simple simulation. In the model, flow of water is assumed to be able to be calculated by integrating the motion of independent water particles. Soil will be dissolved, carried, transported and deposited at lower locations depending on the many conditions, e.g. the altitude difference, water amount and the amount of already-dissolved sediments. In 1993, this method was extended by Roudier *et al.* [40] who took into account possible geological heterogeneities of the ground and the differential erosion that would result. Chiba *et al.* [41, 42] introduced another simple but more physically based algorithm for hydraulic erosion which is based on velocity fields of water flows. The simulation obtains the velocity field using a simplified physics formula and simulates the water flow by applying calculated forces caused by the local gradient.

In 2002, Benes *et al.* [43] presented an improvement of the previous algorithms to simulate hydraulic erosion by distributing all materials to its eight neighbors and introduced an evaporation step to simulate drying pools of water. The authors separated the process into four independent steps which can be applied in arbitrary order to achieved various level of realism. This algorithm was further optimized for a faster method by Olsen *et al.* [44] so as to be used in computer game. For the first time, Olsen proposed a way to define erosion score which are used to measure how eroded a height map is. Olsen proposed many optimization methods to be used in computer games but due to their

optimized nature, physical exactness and visual appearance are secondary, and applicability in computer game is more important. In 2005, based on layered-data structure [45], Neidhold *et al.* [8] improved upon all of previous methods and presented a new and much more physically-based erosion method which is claimed to run in real-time. Semi-Lagrangian approximation for fluid called “stable fluid” [46, 47] and many physics formulas were implemented to enhance the erosion process realism. The paper used Hammersley and Halton points [48] to simulate rainfall and provided a detailed method to calculate acceleration, velocity and new positions of water particles. Also in this year, Benes *et al.* [49] proposed another method for visual modeling table mountains without relying on physically correct modeling. In 2006, Benes *et al.* [50, 51] presented another model based on Navier-Stoke equations. These equations form the basis for the model to balance erosion and deposition that determine changes in the layers between water and erosion materials. Both fractal and physical erosion techniques add terrain details through procedural refinement, which often involves non-intuitive parameter tuning. Recently, Brosz *et al.* [52] attempted to extract high-resolution terrain details from existing Digital Elevation Model data and applied it to a lower resolution terrain through multi-resolution analysis. In practice, their method requires both the source and target terrain to be fairly detailed and does not grant the user freedom to create an arbitrary terrain.

To provide the user with more intuitive control over the synthesized terrain, image-based alternatives were proposed by Lewis [53], Perlin and Velho [54]. Recently, Zhou *et al.* [55] also published a paper to improve this work. Zhou presented an example-based system for terrain synthesis. In his approach, patches from a sample terrain (represented by a height field) are used to generate a new terrain. The synthesis is guided by a user-sketched feature map that specifies where terrain features occur in the resulting synthetic terrain. In these works, terrain was viewed as a type of texture and user control was

provided through direct manipulation of the texture. The result was then interpreted as a height field to create a variety of terrain types. Because it is difficult for a user to draw a realistic natural height field by hand, these methods typically suffer from a lack of realistic detail.

Although most hydraulic erosion terrain synthesis methods proposed so far produce realistic results, they have some limitations. Firstly, the methods normally have many physically-based parameters, making it difficult to control terrain generation with a desired style. Secondly, most of the current researches are only tested with small or low resolution terrains. Figure 13 again shows the eroded result from one of the latest papers in erosion by Neidhold *et al.* in 2005 [8]. As we can see, the authors only tested the algorithm in a terrain with only one small mountain in 256x256 grid cells. The results were implemented in .NET framework, C# and tested on a 2.4 GHz Pentium IV PC; however, the algorithm still runs at 4 frames per second. It is explained mainly due to the process being computationally expensive.

Figure 13: Eroded Mountain from latest erosion paper in 2005. [8]

4.3 Conclusion

Although most of the hydraulic erosion terrain synthesis methods presented so far produce realistic results, they have only been applied or tested on low resolution or small terrains. Applying these erosion simulation algorithms on higher resolution or bigger

Hydraulic Erosion Simulation

terrains including hundreds to thousands of simulation steps can take up to hours to finish. Fortunately, the computational power improvements in the development of graphics hardware performance have been tremendous recently. The objective of this project is to propose an erosion algorithm that would be able to simulate the hydraulic erosion on large terrains, such as 2048x2048 grid cells, as this resolution is required in many graphics application of today. Simulating erosion using the current presented erosion algorithm will take such a long time to produce good results. This is unacceptable in such applications as games where such algorithms typically run while other game data is being loaded and where the whole process typically should not take longer than a minute.

In the next chapter, we will present a proposed erosion model which is an optimized version of current methods that can be implemented on graphics processing unit to improve the erosion simulation process's speed. This algorithm is proposed to be the next generation of GPGPU erosion algorithm.

CHAPTER 5

5 The Proposed Erosion Model

In the previous chapter, we have reviewed the latest development in erosion algorithm. Although the current proposed methods can produce realistic erosion results, they fail to meet the speed demands of real-time computation when simulating erosion on large terrain. Before expounding upon the erosion model which we have implemented on the GPU, let us start with a brief explanation of fluid dynamics to explore the physics of fluid movement, which is the main cause of hydraulic erosion in section 5.1. In section 5.2, data structure used and design considerations taken during the building process of the erosion model will be discussed, and lastly, we will discuss the erosion model proper in section 5.3.

5.1 Fluid Dynamics

Hydraulic erosion is caused by running water. This form of erosion is the most active landscape-transforming agent on the Earth's surface. Waterways erode, transport, and deposit rock and sediment to produce landforms such as canyons, valleys, deltas, alluvial fans, and floodplains.

Before going into the details of hydraulic erosion, let us first look at fluid dynamics, which is the science that describes the movement and interaction of liquids, gases and solids in a system.

Fluid dynamics is the sub-discipline of fluid mechanics dealing with fluids (liquids and gases) in motion. Here are the Navier–Stokes equations [51, 56, 57] to provide a model

The Proposed Erosion Model

that describes the movement of incompressible liquid at an approximately constant temperature and density:

$$\nabla \cdot u = 0 \quad (1)$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho} \nabla p + \nu \nabla^2 u + f \quad (2)$$

where:

- ∇ del is a vector differential operator $i \frac{\partial}{\partial x} + j \frac{\partial}{\partial y} + k \frac{\partial}{\partial z}$
- $\nabla \cdot u$ is divergence of vector u $\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z}$
- p is pressure for an initial time instant t_0
- u is the velocity field of a fluid
- ρ is the liquid density
- ν is the liquid viscosity
- f is an external force

Equation (1) reflects the incompressibility and mass conservation of liquid. Equation (2) expresses the conservation of momentum by combining the pressure field and liquid velocity.

The solution of the above equations is quite complex and has been used in many animation application [47, 56]. However it is computationally expensive which makes the equations inappropriate for large scale terrain simulations in computer graphics. Neidhold *et al.* [8] introduced a practical approach to model and animate the flow of liquids that has been widely accepted in computer graphics. As it is not really necessary to simulate all of the 3D-features like multiple water layers, vertical vortices or waves, these equations are optimized and then discretized to a two dimensional version. The result, com-

pared to real Navier-Stokes solvers is less physically accurate but much faster and meets the requirements of our erosion algorithm very well [8].

5.2 Design Consideration

Hydraulic erosion is a result of thousands or even millions of years of erosion by water on terrain. Before going into the details of our erosion model, these were some of our design considerations. Each consideration was crucial as a small change in any of these choices can create a completely different erosion result. Let us start with the data structure discussion in section 5.2.1. Section 5.2.2 will consider how to distribute water all over the terrain, while section 5.2.3 discusses how to calculate acceleration and velocity in previous papers. In section 5.2.4 and section 5.2.5, how water is moved will be presented and last but not least, evaporation will be considered in section 5.2.6.

5.2.1 Data Structure

Data structure is the mechanism to save simulation information data, especially the terrain height. The data is always stored to be used in other simulations or just be saved for the application (terrain height data). There are many data structure available that can be used to save the data. A two dimensional regular height field is the simplest and most commonly used data structure for terrain visualization and environmental simulations. Many papers on erosion topics have used this simplest data structure [6, 40, 42, 45, 60]. This is also used by 2D/3D fluid simulations [46, 58, 59]. The low memory usage is the advantage of this representation in which each grid cell stores the height value of the underlying terrain. But not every terrain type can be represented by a height field data structure, for example it is difficult to store the terrain with caves using this data structure.

The Proposed Erosion Model

A voxel grid is a three dimensional data structure where each grid cell stores an id of the local material type such as air, water, sand or granite etc., so that the terrain data and the fluid data can be stored together in one grid. Dorsey *et al.* [60] and Ozawa *et al.* [61] have used this voxel data structure in their modeling and rendering of weathering stones. Unlike a two dimensional regular height field, a voxel grid data structure can represent caves, however, it requires much more storage space [62, 63]. Therefore, erosion simulations with voxel grid are not appropriate to be used in erosion models aiming to achieve real-time results.

Benes *et al.* [45]] introduced the idea of using a layer-based data structure. Unlike the above-mentioned data structures, the grid can now have many layers and for each cell position (x, y) a static array of attributes (material-id, width) is stored. This approach is a good alternative whenever only a few different layers lie on top of each other and can also be used for fluid simulation.

Another efficient data structure for simulating fluid is using particle systems [64]. Holmberg *et al.* [65] proposed a combination of a particle system and a height field to model and render turbulent water over natural terrain which also turns out to be very efficient. Sets of triangles are a rapid representation for terrain and fluid in real-time rendering for graphical information systems (GIS). However, they are not appropriate for erosion simulation in combination with a fluid because the re-triangulation that is necessary after an erosion step can be quite expensive.

5.2.2 Water distribution

Rainfall is the most obvious factor that causes hydraulic erosion. Rainfall in nature is very much affected by a complicated system of wind and air humidity, called adiabatics. However, even with very large amount of rain each time, running water only can affect a

very small amount of erosion. We are approximating the eroded terrain, not simulating the real erosion process, therefore, we must consider whether there is a need to distribute rainfall thousands of times or maybe just a few times. Simulating many rainfalls can take very long time and it might be unnecessary. Many previous works in academic literature did not address this issue very clearly. Some papers [7, 38, 40] proposed modeling water distribution only at the beginning of the simulation. The commercial software World Machine [11] method (published online by the author) suggested distributing water only once. Another paper [8] proposed the distribution of rainfalls many times at random intervals.

Another issue in water distribution is how much water is distributed over the terrain. All of the cells will receive the same amount of water [7, 11, 38, 40, 42-44] or the higher the cell is, the more water it receives [8].

5.2.3 Acceleration and velocity calculation

Water is moved from a higher position to a lower position by gravity. In order to have a physically correct simulation, acceleration and velocity calculation of this water movement is essential. In real life, water spreads from a higher point to all of its lower neighboring points and gaining velocity as it does so. How this acceleration and velocity are computed can vastly affect the final result. However, we wish to emphasize that whatever the method used, and regardless of whether it is physically correct or not, the point is that the result must look realistic. The very first paper on erosion [7] ignores the calculation of acceleration and velocity. The authors, Musgrave *et al.*, just assumed water would be spread from a higher point to all lower neighboring points based on the height difference. Later, Chiba *et al.* [42] proposed a new erosion model to calculate velocity fields of water flow. In his model, acceleration is supposed to occur in the direction with

the maximum angle of inclination. In another paper, Neidhold *et al.* [8] calculates acceleration by averaging all obtained direction gradient vectors from lower grid cells.

5.2.4 Water movement to only ONE neighbor or all lower neighbors

Water movement creates ridges, valleys, erosion lines all over the terrains. In simulation, we can choose to assume that water moves from a higher point to all lower positions [7] or just to the lowest position (World Machine method [11]). Alternatively we also can calculate velocity directions and move all waters according to these directions [7, 40, 44, 45]. Each of these choices can create different erosion effects.

5.2.5 Water movement of all waters or just part of the water

Gravity can move all of the water amount or just part of it from a high position to lower positions, depending on the case. Benes *et al.* in 2002 [43] addresses this issue most clearly. If the element is not the local minima, he explained two important cases as described in Figure 14.

Figure 14: Two important cases for the water transportation

The left diagrams show the situations before the water has been moved whereas the right diagrams after that has happened. The upper row describes the case where just a part of

The Proposed Erosion Model

water needs to be moved, whereas the bottom row depicts the case where all of the water needs to be moved.

5.2.6 Water Evaporation

One of the design considerations that we had was whether there is a need to simulate evaporation. In real-life, evaporation results in some of the sediments carried by the evaporated water being deposited on the terrains due to the reduced amount of water available to carry these sediments. Benes *et al.* [43] was the first author to introduce evaporation in an erosion model. To the best of my knowledge, no papers mentioned evaporation before that. Many papers proposed later used exponential equation to simulate the reduction of water due to evaporation [8, 43, 49].

5.3 The Proposed Erosion Model

In the previous section, we have considered various erosion model designs in term of data structure, water distribution, water movement and water evaporation. In this section, our proposed erosion model design will be discussed. Data structure selection and water distribution will be explained in section 5.3.1 and section 5.3.2 respectively. Water movement and evaporation will be elaborated in combination in section 5.3.3 Water Simulation. And lastly, a very important part of hydraulic erosion simulation, function formulas used to calculate how much sediment is eroded and deposited in each erosion simulation step will be presented in section 5.3.4, the Hydraulic Erosion Function.

5.3.1 Data Structure

In our design, the layered data representation of Benes [45] is selected as a trade-off between a height field and a voxel grid. Four layers are onto our two dimensional height field H to save all related data for the fluid and the erosion simulation. Each layer is represented as a two dimensional array. For our fluid simulation the actual fluid amount F ,

The Proposed Erosion Model

velocity $\vec{v}$ are saved in each discrete grid cell. At each step, the 3D vector acceleration $\vec{a}$ is also calculated. Our erosion model uses the dissolved sediment layer S which stores the amount of sediment in each cell.

Table 1: Data grids used in simulation

Velocity (3D)	$\vec{v}$
Fluid Amount	F
Dissolved Sediment	S
Terrain Level	H

The actual simulation is performed by applying finite differences algorithms on these layers. The layers can interact, for instance, dissolved material can elevate the terrain at a certain position. In the next section, we will explore things to be considered when modeling fluid movement and its erosion effects.

5.3.2 Water distribution

The most significant influence for a hydraulic erosion system is the allocation of water. The main type of water resources is rainfall or precipitation. As computation time is a very essential factor in the simulation, rainfall is simulated by dropping some amount of rain to each cell at the beginning of the simulation. Basically, it is assumed that the higher the cell is, the more water it can receive. The following proposed equation describes the amount of water distributed based on each grid cell's height:

The Proposed Erosion Model

$$W = W_{\min} + W_0 * \left(\frac{H_0}{H_{\max}} \right)^2 \quad (3)$$

where:

- ❖ W is the amount of water one cell can receive.
- ❖ $W_{\min}$ is the minimum amount of water that the lowest cell can receive.
- ❖ W_0 is the maximum add-in that the highest cell can receive.
- ❖ H_0 is current height of the cell.
- ❖ $H_{\max}$: maximum height of the terrain.

Besides the rain amount, rain distribution on the terrain, which is realized with a noise pattern, also has a significant effect on the erosion result. Figure 15 shows an example of a rain distribution where the water positions (in blue) are acquired from Hammersley point sets [48, 66]. For better performance, a very dense point set is pre-calculated.

Figure 15: An example of Hammersley Point Set rain distribution

5.3.3 Water Simulation

There have been many research works over the past 50 years which document how to find approximate solutions for the full 3D Navier-Stokes equations which describe realistic fluids animation. An approximation for smoke-like fluids called "stable fluids" was proposed by Stam [47]. As it is unnecessary to simulate all of the 3D-features like multiple water layers, vertical vortices or waves, these equations are optimized and then discretized to a two dimensional version. The result compared to real Navier-Stokes

The Proposed Erosion Model

solvers is less physically accurate but much faster and meets very well the needs of our erosion algorithm. The physics foundation for our approach originated in the paper [8]. The authors used the Newtonian physics approach for fluid simulation. The foundation for this approach is a system of first order differential equations describing the movement of water depending on velocity $\vec{v}$ and acceleration $\vec{a}$:

$$\dot{\vec{v}} = \vec{a} - K_A \cdot \vec{v} = \frac{\vec{F}}{m} - K_A \cdot \vec{v} \quad (4)$$

$$\dot{\vec{x}} = \vec{v} \quad (5)$$

where $\vec{F}$ is the force vector which moves the water (originated from gravity vector), m is the water mass and $K_A \in [0..1]$ is a friction parameter to control the sliding friction between water and terrain. With the value $K_A = 0.2$ used in the simulation, a very rough underlying terrain is emulated. It reduces the fluid velocity about by 20 percent in one time unit. For time-discretization of the differential equations the most basic kind of explicit Euler method is used [67]:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \cdot \Delta t - K_A \cdot \vec{v}_t \cdot \Delta t \quad (6)$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t} \cdot \Delta t \quad (7)$$

When the actual acceleration is calculated from the gravity vector, the new velocity after the time step Δt , $\vec{v}_{t+\Delta t}$ can be computed at each discrete grid cell.

Although in this approach, the new water position is calculated from its velocity, our experiments show that the simulation, in which the amount of water is divided and moved to all lower neighbors, gives the best erosion result. Water is still accelerated by gravity and the velocity still needs to be calculated, but the velocity will not determine the next

The Proposed Erosion Model

water position. Assuming that the current cell has an amount of water W , the following equation describes the amount of water that will be moved to a lower neighboring cell:

$$\Delta W = W \frac{\Delta h}{\sum(\Delta h)} \quad (8)$$

where $\sum(\Delta h)$ is the total of the height differences between the current cell and all its lower neighboring cells. Hence, the amount of water moved to the neighboring cell is proportional to the height difference Δh between the current cell and the neighboring cell.

Another issue is how to deal with the velocities and the water vectors from different neighbor locations that are transported to the same discrete grid cell. The components of the water vectors are simply added up, for example, the water amount at the destination point W_{dest} and the transported water ΔW is summed up to the new water amount W_{new} . The corresponding velocity vectors also have to be combined. The new velocity $\vec{v}_{new}$ will be calculated based on averaging principle:

$$W_{new} = W_{dest} + \Delta W \quad (9)$$

$$\vec{v}_{new} = \frac{W_{dest} \cdot \vec{v}_{dest} + \Delta W \cdot \vec{v}_{\Delta W}}{W_{new}} \quad (10)$$

where $\vec{v}_{dest}$ is the velocity of the destination neighboring cell, respectively, and $\vec{v}_{\Delta W}$ is the velocity of the transported water ΔW .

5.3.3.1 Acceleration Calculation

The acceleration direction at the position (x, y) is usually assumed to be the direction of the biggest tilt angle α of the underlying terrain height field. In order to have a more

precise simulation, the water can be split up and accelerated with the directional derivatives of all lower neighboring cells.

Figure 16: Acceleration Calculation

Firstly, let us consider how the function $\text{TotalHeight}(x, y)$, which is used for gradient calculation, must be defined exactly. If it is only defined as the actual level of the underlying terrain $H(x, y)$, all fluid will flow to one of the local minima in the grid (hollows) and will just not be able to move else where. But if it is defined as the sum of terrain level $H(x, y)$ and the fluid amount $F(x, y)$, then water runs down a slope very fast. It can be slowed down by water in adjacent cells. A compromise between these two extremes is to use an influence factor $K_F \in [0..1]$ that describes how much of the water at a grid point influences the gradient calculation.

$$\text{TotalHeight}(x, y) = H(x, y) + K_F \cdot F(x, y) \quad (11)$$

Let us assume that $\text{TotalHeight}(x+1, y)$ is lower than $\text{TotalHeight}(x, y)$, so water will move from cell (x, y) to cell $(x+1, y)$.

Here are the formulas to calculate acceleration vectors from gravity vectors:

$$\vec{a} = \vec{g} \cdot \sin \alpha \quad (12)$$

$$\sin \alpha = \frac{\Delta h}{|\vec{X}|} \quad (13)$$

where $\vec{g}$ is the gravity vector and $\vec{X}$ is the vector from current cell to its lower neighboring cell and Δh is equal to $\text{TotalHeight}(x,y) - \text{TotalHeight}(x+1,y)$.

5.3.3.2 Evaporation

In nature, the amount of water evaporation depends on the temperature and the amount of water. Benes *et al.* [43] proposed an evaporation formula which reduces the amount of water exponentially over time.

$$w_t = w_0 \cdot e^{-K_E t} \quad (14)$$

Where w is the water amount and $K_E \in [0..1]$ is the evaporation constant which controls the process. However, this exponent function will slow down computation time and is not very essential for the simulation; the evaporation function has thus been optimized for better performance to become:

$$W_{t+\Delta t} = W_t \cdot K_E \quad (15)$$

Neidhold *et al.* [8] also proposed a varying evaporation based on low temperatures on high terrain levels and high temperatures for a more correct physical evaporation, however this also can slow down the calculation process.

5.3.4 Hydraulic Erosion Function

When water flows over a given terrain, rock particles, soil and stones will be dislodged and transported to lower areas. A hydraulic erosion function is thus needed to simulate this effect. This function is used whenever the water moves from one grid cell to another one during fluid simulation. The amount of transported fluid ΔW , the amount of sediment dissolved in this fluid ΔS and the velocity of the fluid $\vec{v}$ are the inputs to the functions. If erosion occurs, the transfer function is modified such that some sediment will be deposited or dissolved from the underlying terrain at the destination cell.

Our erosion algorithm uses some material specific constants which control the entire process. The Sediment Capacity Constant K_C describes how much sediment of the transported material can be dissolved in one unit of water at a velocity of one. The Deposition Constant $K_d \in [0..1]$ controls the rate at which material particles are deposited at the target grid cell. On the other hand, the Dissolving Constant $K_s \in [0..1]$ controls the rate at which the underlying terrain dissolves into the water after each simulation step. The relatively large default value of $K_C = 250$ can be used to match the different magnitudes of time steps for fluid simulation (normally measured in seconds) and erosion simulation (measured in years).

Table 2: Material constants and their default values

Sediment Capacity Constant	K_C	250
Deposition Constant	K_d	0.05
Dissolving Constant	K_s	0.04

To determine whether to deposit or to dissolve the sediment after each simulation step we compute the Sediment Capacity S_{cap} of the water. This indicates the maximum amount of

The Proposed Erosion Model

sediment that can be transported by the fluid ΔW . As our fluid simulation also provides the actual velocity $\vec{v}$, our erosion model allows us to compute S_{cap} in a more physically correct way compared to prior presented models [7]:

$$S_{cap} = K_c \cdot \Delta W \cdot |\vec{v}| \quad (16)$$

In general, the sediment capacity c_s is proportional to the Sediment Capacity Constant K_C , transported fluid amount ΔW and the magnitude of velocity $\vec{v}$.

This Sediment Capacity c_s will be compared to the actual amount of dissolved sediment ΔS . The following formulas allow us to make this decision:

If $\Delta S > S_{cap}$ deposit:

$$H' = H + K_d \cdot (\Delta S - S_{cap}) \quad (17)$$

$$S' = S + \Delta S - K_d \cdot (\Delta S - S_{cap}) \quad (18)$$

If $\Delta S < c_s$ dissolve:

$$H' = H - K_s \cdot (S_{cap} - \Delta S) \quad (19)$$

$$S' = S + \Delta S + K_s \cdot (S_{cap} - \Delta S) \quad (20)$$

where H and S are the height and the amount sediment of destination cell before the water moves from its neighboring cell while H' and S' are the updated height and amount of sediment of the destination cell. ΔS and S_{cap} are the dissolved sediment and sediment capacity of the transported water. Take note this does not refer to the water in the destination cell.

Figure 17: Sediment Erosion and Deposition Calculation.

In both cases the transfer function is modified for the height H and the sediment S dissolved in the fluid with an erosion term. This term is additionally scaled with one of the two material constants K_d and K_s that are particular for erosion. They are used since depositing and dissolving are continuous processes and hence the erosion process affects a specified portion of the difference $\Delta S - c_s$.

5.4 Graphical Illustrations of Erosion Model

In sequence, there are four main processes in modeling hydraulic erosion:

- ❖ Water is distributed over the terrain.
- ❖ Water moves from a high point to its lower neighboring points due to gravity.
- ❖ Water erodes mostly the higher positions of the terrain and deposits some of the dissolved sediments on lower positions.
- ❖ Water evaporates due to heat and deposits most of the sediments it was carrying on the bottom of the terrain.

On the next two pages are the graphical illustrations of these four main processes.

The Proposed Erosion Model

Figure 18: Water distribution over the terrain.

Figure 19: Water movement from high positions to lower positions.

: Terrain : Water : Sediment Deposition

The Proposed Erosion Model

Figure 20: Water moves down and erodes the higher positions of the terrain.

Figure 21: Water evaporates and leaves the deposition mostly on the terrain bottom

: Terrain : Water : Sediment Deposition

5.5 Conclusion

In chapter 4, the state-of-the-art developments of erosion algorithm have been reviewed. Despite the fact that the current proposed erosion simulation technique can produce good erosion results, they can not meet the requirements in term of efficiency needed to simulate the erosion process on large-scale terrains. In this chapter, a well-designed hydraulic erosion model has been proposed, which can be implemented on graphics processing unit. This model is not based on the complicated fluid dynamics equations due to the computational cost handling such equations which makes them inappropriate for large scale terrain simulations in computer graphics. Based on the Newtonian physics approach, these equations are optimized and then discretized to a two dimensional version. The result is much faster and meets the requirements of our erosion algorithm very well. In the next chapter, we will present the implementation on both CPU and GPU in order to give a comparison and show the general advantages of transporting an algorithm from CPU to GPU. Time performance statistics will be also discussed in the next chapter.

CHAPTER 6

6 Implementation & Experimental Results

In the previous chapter, we have presented the hydraulic erosion model to be transported over to GPU program with the purpose of achieving a faster simulation. In this chapter, the implementation on both CPU and GPU versions will be discussed. Section 6.1 starts the chapter with the development platform and implementing language. Section 6.2 and section 6.3 will describe in details the erosion implementation on CPU and GPU and especially the comparison of performance statistics. In order to speed up the simulation, an optimization over the implementation has been done and will be presented in section 6.4. Section 6.5 adds other add-on channel erosion features to improve realism of the terrain. Section 6.6 describes how the GPU implementation is incorporated with the renderer to deliver a GPU-based erosion tool. Section 6.7 gives a brief overview of how the GPGPU erosion algorithm performs on various graphics card in section and also a comparison of our tool with a commercially-available erosion tool is made in order to ascertain the advantages of using a GPU-based erosion algorithm.

6.1 Platform and Language

The platform which was used to develop and test the erosion model is a Pentium D 3.2Ghz dual core equipped with NVIDIA GeForce 7900 GS graphic card. In this project, Visual Studio C++ was used for the erosion implementation on CPU whereas DirectX 9.0 and High Level Shading Language (HLSL) were used for the implementation on GPU.

Implementation & Experimental Results

This Visual Studio development toolkit provides sufficient debugging utilities and programming tools to facilitate the whole development process.

Although the NVIDIA GeForce 7900 GS graphic card has powerful computational capabilities, it only has 256 MB of onboard video memory which limits the amount of data to be processed. The following table shows the specifications of 7900 GS graphics card.

Table 3: GeForce 7900 GS specification

Memory	256MB
Memory Interface	256-bit
Memory Bandwidth (GB/sec)	42.2
Fill Rate (Billion pixels/sec)	9
Vertices/Second	822.5 Million
Pixels per clock (peak)	20
RAMDACs (MHz)	400

6.2 CPU Implementation

6.2.1 Class Diagram

The Erosion Model Class was implemented as shown in the diagram in Figure 22. In general, the main challenge of the implementation is to modify the parameters so that the result looks realistic.

Implementation & Experimental Results

Figure 22: Erosion Model Class Diagram

The explanation is as follows:

- ❖ myMap is the class storing square terrain height data; the ‘size’ attribute is used to save the length and width of the terrain. Floating point data type is used to store the terrain height data for increased accuracy.
- ❖ myTerrain is the class modifying height map of the terrain. Its methods focus on changing the height map. In this class, the ErodeTerrain() method is implemented.
- ❖ myErosion is the class responsible for erosion simulation. Many control parameters and physically-based parameters are used to monitor the simulation. The erosion algorithm is implemented in the SimulateErosion() method.
- ❖ myMaterial class an implementation of the three-layered data structure (another layer, which is the heightmap layer, is stored in myMap class). It saves water amount, sediment amount and velocity data.
- ❖ myVector is the class used to save 3D data such as velocity and acceleration during implementation process.

6.2.2 The Erosion algorithm

The hydraulic erosion process can be described by the following four main processes:

- ❖ Water is distributed over the terrain.
- ❖ Water moves from a higher position to its lower neighbors due to gravity.
- ❖ Water erodes mostly the higher positions of the terrain and deposits some of the dissolved sediments on lower positions.
- ❖ Water evaporates due to heat and deposits most of the sediments it was carrying on the bottom of the terrain.

There are many physically-based parameters used in the simulation which have been described in the design chapter of this thesis. Any change in these parameters can return a different result. Here is one example of the parameter values which were used when testing this model with a 1024x1024 terrain.

- $\text{FRICTIONKA} = 0.2f$
- $\text{EVAPORATIONKE} = 0.9f$
- $\text{GRAVITYACCELERATION} = 10.0f$
- $\text{DEPOSITIONKD} = 0.05f$
- $\text{DISSOLVINGKS} = 0.04f$
- $\text{RAINSKALINGKR} = 0.001f$
- $\text{SEDIMENTCAPACITYKC} = 150.0f$
- $\text{RAINOFFSET} = 200$
- $\text{MINRAINAMOUNT} = 60$
- $\text{DELTALength} = 2.0f$
- $\text{DELTAHEIGHT} = 0.003f$
- $\text{DELTATIME} = 0.02f$
- $\text{VELOCITYTHRESHOLD} = 2.0f$
- $\text{HEIGHTTHRESHOLD} = 100.0f$
- $\text{NUMBEROFRAINPOINTS} = 40000$
- $\text{MAXDEPOSITION} = 100.0f$

Implementation & Experimental Results

- MAXDISSOLVE = 200.0f
- MAXDEPOSITION = 30.0f
- MAXDISSOLVE = 50.0f
- STEPNUMBER = 100
- PI = 3.14159265f

Figure 23: Original 1024x1024 Synthesis Terrain

Implementation & Experimental Results

The following is the pseudocode of the basic erosion algorithm:

```

1  while (step < StepNumber)
2  {
3      if (step = 0) Water Distribution; end if;
4      Water Evaporation;
5      for each cell in the Height Map
6      {
7          if there is water in the cell
8          {
9              for each lower neighbor
10             {
11                 calculate acceleration vector based on height difference
12                 calculate fluid velocity vector based on this acceleration
13                 move part of the fluid to this neighbor
14                 calculate Sediment Capacity based on the velocity magnitude, fluid amount
15                 to decide whether to deposit or dissolve the sediment
16             }
17             end for;
18             if there is water in the cell and there is lower neighbor
19                 remove all the water
20             if there is no lower neighbor (local minima)
21             {
22                 deposit part of the sediments;
23             }
24             end if;
25         }
26         end if;
27     }
28     end for;
29 }
30 end while;

```

6.2.3 CPU version results

In general, with 1024x1024 terrains, it is unnecessary to run the simulation for many simulation steps. 50 or 100 steps should typically be enough for the simulations.

6.2.3.1 1Kx1K Terrain Experiment

Figures 23, 24 and 25 show the original 1Kx1K original synthesis terrain and the terrains after 50 and 100 erosion simulation steps respectively. In this experiment, water is distributed only one time at the beginning of the simulation and water is slowly evapo-

Implementation & Experimental Results

rated after each step. Basically, both eroded terrains look pretty similar after the simulation except the simulation with higher steps has slightly more deposition. Higher resolution pictures are shown in the appendix B for comparison. It took 1 minute and 7 seconds to finish the 50 erosion simulation steps and 1 minute 22 seconds to finish the 100 steps variation.

Figure 24: Eroded Synthesis Terrain after 50 steps in 1 min and 7 seconds

Figure 25: Eroded Synthesis Terrain after 100 steps in 1 min and 22 seconds (deposition is slightly more).

As explained in the design section of this paper, the water erodes the terrain at the higher elevation, transports this eroded sediment and deposit it smoothly at lower elevation. Figures 26, 27 show the original terrain from top view and the eroded terrain with deposition respectively for comparison. For bigger photos, please see the appendix C.

Implementation & Experimental Results

Figure 26: Original Terrain

Figure 27: Eroded Terrain with Deposition

6.2.3.2 2Kx2K Terrain Experiment

When the resolution of the terrain height map is increased, more positions in the terrain are sampled. As a result, the difference of height between two pixels is now smaller. As such, the water velocity after each simulation step will be smaller and the terrain will

Implementation & Experimental Results

have less erosion. Figure 28 shows a 2Kx2K eroded terrain when using the same parameters as the 1Kx1K erosion terrain simulation.

Figure 28: 2Kx2K eroded terrain when using the same parameters with 1Kx1K simulation

It is easier to see the difference when the sediment capacity parameter is increased for 2Kx2K terrain simulation. Figure 29 and 30 show 2Kx2K eroded terrains with an increased Sediment Capacity KC after 100 steps and 150 steps respectively.

Implementation & Experimental Results

Figure 29: 2Kx2K eroded terrain when increasing Sediment Capacity KC with 1Kx1K simulation (100 steps)

Table 4 shows the time comparison between the 1Kx1K and 2Kx2K erosion simulation.

Table 4: Erosion Simulation Time Comparison

	100 steps	50 steps
1Kx1K	1 min 22 s	1 min 7 s
2Kx2K	5 min 42 s	4 min 50 s

Figure 30: 2Kx2K eroded terrain when increasing Sediment Capacity KC with 1Kx1K simulation (150 steps)

6.3 GPU Implementation

As the CPU version is not fast enough, the algorithm was rewritten in HLSL. The reason why GPU can perform computation faster than CPU is that all GPU computations are done in parallel, in other words many pixels are processed at the same time. However, there are many limitations on the implementation of the algorithm in shading language.

6.3.1 Debugger problem

Besides the fact that there are many limitations in GPGPU programming, an additional consideration is that there is no proper debugger to look for runtime errors. PIX has been claimed to be a good debugger for GPU programming [26], but it is only able to debug simple shading programs which are mainly used for shading of objects. When the code gets complicated like GPGPU code, PIX easily crashes and can not be depended on to recover from these crashes.

6.3.2 From CPU to GPU

GPU programming is graphics-centric. In CPU programming, data stream is saved in arrays, for example, the terrain height map is saved in a 2D array; in GPU programming, data stream is packed in textures. Figure 31 shows an analogy of the differences between data structures used in CPU and GPU.

Figure 31: CPU - GPU Data Structure Analogy

There are three color channels and one alpha channel in each texture (RGBA). It is thus possible to use up all of these four channels in texture to pack the data. In GPU programming, the algorithm steps are written as fragment programs (or Pixel Shader). Most algorithms which are implemented in GPU programming utilize feedback which means each algorithm step depends on the results of previous steps. Therefore, in GPU programming, the outputs of fragment program will be used as the input textures for the next algorithm step. The current state is stored in textures. The feedback is obtained via render-to-texture. To invoke the computation, a quad is drawn and these textures are mapped on that quad which is then passed to the pixel shader.

Implementation & Experimental Results

Some other useful analogies between CPU and GPU programming includes:

- Rasterization = Kernel Invocation
- Texture Coordinates = Computational Domain
- Vertex Coordinates = Computational Range

6.3.3 Data Packing

Because we have four layers of data to store in each step (height, fluid, sediment and the 3D velocity), one input/output texture is not enough.

Table 5 shows in details how all of the data is stored in two textures. Direct3D data format floating point D3DFMT_A32B32G32R32 (128 bits) is used to ensure the retention of the accuracy of the simulation.

Table 5: Erosion Simulation GPU data packing

	Red	Green	Blue	Alpha
Texture 1	Height	Fluid	Sediment	PreComputation values
Texture 2	Velocity.x	Velocity.y	Velocity.z	PreComputation values

6.3.4 Reverse Erosion Algorithm

Figure 32: Water flow calculation between normal and reverse erosion algorithm

In Pixel Shader, we can only gathering and not scattering. In other words, the GPU program can only write or modify the current pixel value and cannot write or modify the

Implementation & Experimental Results

pixel's neighbor values. Therefore, in order to implement in GPU fragment program, the erosion algorithm has to be implemented in reverse order.

In the normal erosion algorithm, each current cell is checked to see whether the cell has water. The water will be moved to lower neighboring points, if there exist any. In the reverse algorithm, the neighbors of current cell will be checked to see if they have any water that would be moved to the current cell. The following illustrates the pseudocode of the reverse erosion algorithm.

```

31  while (step < StepNumber)
32  {
33      if (step = some condition) Water Distribution; end if;
34      if (step = some condition) Water Evaporation; end if;
35      for each cell in the Height Map
36      {
37          for each higher neighbor
38          {
39              calculate acceleration vector based on height difference
40              calculate fluid velocity vector based on this acceleration
41              move part of the fluid from this neighbor to current cell
42              calculate Sediment Capacity based on the neighbor velocity magnitude, fluid amount to
43              decide whether to deposit or dissolve the sediment to current cell
44          }
45          end for;
46          if there is water in the cell and there is lower neighbor
47          {
48              remove all water;
49          }
50          end if;
51          if there is no lower neighbor (local minima)
52          {
53              deposit part of the sediments;
54          }
55          end if;
56      }
57      end for;
58  }
59  end while;

```

The reverse erosion algorithm requires more computation. For each cell, we have to recalculate the amount of water that needs to be moved from its neighboring cells.

Implementation & Experimental Results

Experiments have shown that implementing this reverse erosion algorithm in C++ code results in the algorithm taking *5 times* longer than the normal erosion algorithm. However, this is the only way to port the algorithm into a GPU program. With the GPU's powerful computational capabilities, it is still expected to be faster than the normal erosion program which is implemented in CPU.

6.3.5 Memory Requirement

Memory requirement is one of the major problems in GPGPU programming. The current development platform's graphics hardware is an NVIDIA GeForce 7900 GS card which has only 256 MB memory. This is inadequate for our needs.

The erosion shading program needs two textures as inputs and returns two textures as outputs. Therefore, at least four textures are required. In order for the simulation to have higher accuracy, data has to be stored in 32 bit floating point format. However, if this exceeds graphics hardware memory capacity, the 16 bit floating point data format can be considered instead.

For example, we need to run the erosion simulation for a 1Kx1K terrain using 32 bit (4 bytes) floating point data format. Here is how much memory is required to store the textures which are computed:

$$\text{Memory required} = \text{Terrain Size} \cdot \text{num of bit/channel} \cdot 4 \text{ channels} \cdot 4 \text{ textures} \quad (21)$$

Therefore, memory required in this case: $1\text{K} \cdot 1\text{K} \cdot 4 \cdot 4 \cdot 4 = 64 \text{ MB}$.

Implementation & Experimental Results

The following table shows how much memory is required in other cases:

Table 6: GPU Memory Requirement

	16 bit floating point per channel	32 bit floating point per channel
1Kx1K Terrain	32 MB	64 MB
2Kx2K Terrain	128 MB	<u>256 MB (out of video memory)</u>

Table 6 shows that 256 MB video memory is needed to simulate erosion on 2Kx2K terrains when using 32 bit floating point/channel data format. This is the maximum amount of memory that the GPU has and therefore it is not possible to run the simulation because the GPU still needs memory for other processes such as storing variables, temporary memory for computation etc.

In short, the only way to run the erosion simulation on 2Kx2K terrains using GPU is to use the 16-bit floating point per channel data format. This means that the accuracy is reduced since a 16-bit floating point data format cannot represent as many values as a 32-bit floating point data format [25, 68].

6.3.6 GPU shading program

There are many simulation steps in the erosion model. At each step, the shader including three passes will be called. Figure 33 illustrates the order of calling these passes in the GPU shading program. Each pass includes one vertex shader and one pixel shader. Only one vertex shader was used for these three passes and three pixel shaders were written separately. Note that it is possible to write the whole program in one pass, but that will make it slower. Details of the passes are as follows:

Implementation & Experimental Results

- ❖ Add-Rain Pass: Rain is distributed first in the simulation. In this pass, the fluid amount is simply added to each cell. This job can easily be done using a for loop in C++ code. However, to speed the whole program, the rain distribution program was also written in shading language. This pass is only called once.
- ❖ PreComputation Pass: in the reverse erosion algorithm, the total neighbor height difference needs to be calculated. If this were implemented in only one pass, the same calculation job would be done many times. PreComputation pass was written to prevent this in order to speed the whole program.
- ❖ Erosion Pass: this is the main portion of the erosion model. This erosion pass reads the two textures from last simulation steps and uses the erosion algorithm to update and return two new output textures. These new textures are feed back to the next simulation step.

Figure 33: Shading Program

6.3.6.1 Control Parameters

To let the user have more control in the program, a setting text file is used to save control parameters. Because there are so many physically-based parameters and it is also hard to

Implementation & Experimental Results

modify them all without understand their meanings, the setting text file is designed to only provide three parameters which mainly affect the erosion results:

- SedimentCapacity
- RainDensity
- ErosionStep

While ErosionStep is similar to the simulation step physically-based parameter, SedimentCapacity and RainDensity parameters are scaled to a range of [0..1] so that it is easier to control.

6.3.7 GPU version results

The GPU variation of the eroded terrain is quite similar to the CPU version. However, in 2Kx2K terrain erosion simulation, because it is not possible to use 32 bit floating point data format, the simulation was not as accurate. Figure 34 shows the difference between two versions of the eroded terrains.

Implementation & Experimental Results

Figure 34: Comparison between 2Kx2K eroded terrains from CPU and GPU
Above: CPU, below: GPU

Implementation & Experimental Results

The eroded terrain from CPU is slightly smoother than the one from GPU, but the speed advantage provided from GPU simulation is more preferable.

6.3.7.1 Statistics Performance

Converting the erosion model from CPU code to GPU code provides a great advantage in terms of speed for the whole program. In general, erosion simulation running in GPU can be 6 to 9 times faster than running in CPU. The following table shows the approximate time taken to finish the simulation with 1Kx1K and 2Kx2K terrains in CPU and GPU:

Table 7: Time Comparison between CPU and GPU version

	1Kx1K Terrain		2Kx2K Terrain	
	CPU	GPU	CPU	GPU
50 steps	1 min 7 s	<i>11.2 s</i>	4 min 50 s	<i>26.8 s</i>
100 steps	1 min 22 s	<i>17.2 s</i>	5 min 42 s	<i>44.3 s</i>

6.4 Optimization

The tremendous speed increase achieved from GPGPU does not just come from the powerful computational capabilities of the GPU; the resulting program has also been optimized in many ways by the conversion process:

6.4.1 Save data in a smaller type

Normally, the D3DFMT_A32B32G32R32F floating point data format is used to ensure accuracy of data. However, using this data type, the simulation would not be able to run on 2Kx2K terrains. By solving some limitations on the D3DFMT_A16B16G16R16F format, such as preventing the data value becomes unrepresentable in 16 bit floating point

Implementation & Experimental Results

format, this data structure can be used to reduce memory requirements and even allows us to simulate the erosion process on 2Kx2K terrains..

6.4.2 Stop transferring data in between GPU and CPU memory

In previous versions of GPU programming, after each simulation steps, we had to use `GetRenderTargetData()`, a function of DirectX 9.0, to retrieve data from the GPU. This is potentially a waste of processing time as this step will keep the CPU busy. However, with four render target data, two of them can be set as input data and the other two as output. (One render target cannot be set as both an input and an output; otherwise weird results will be achieved) This helps to save a lot of time.

6.4.3 Move neighbor texture coordinate calculation to Vertex Shader

The erosion algorithm needs to access neighboring cell values. Previously, Pixel Shader was used to calculate the coordinates to access the texture. As all of the pixel would be included in the calculation, this can contribute a lot of unnecessary computational. Since texture coordinates can be interpolated between vertex shader and pixel shader (interpolation process), this calculation was moved to the Vertex Shader to save computational time. The following code shows how vertex shader was used to calculate neighboring texture coordinates.

```

60  VS_OUTPUT VSMain( float4 inPosition:POSITION, float2 inTexCoord0:TEXCOORD0)
61  {
62      VS_OUTPUT Out;
63      Out.Position = mul( inPosition, gWorldViewProjection );
64      float2 ox = float2(1/gSize,0.0f);
65      float2 oy = float2(0.0f,1/gSize);
66
67      Out.TC[0] = inTexCoord0 - ox - oy;
68      Out.TC[1] = inTexCoord0 - oy;
69      Out.TC[2] = inTexCoord0 + ox - oy;
70      Out.TC[3] = inTexCoord0 - ox;
71      Out.TC[4] = inTexCoord0 + ox;
72      Out.TC[5] = inTexCoord0 - ox + oy;
73      Out.TC[6] = inTexCoord0 + oy;
74      Out.TC[7] = inTexCoord0 + ox + oy;
75      Out.TC[8] = inTexCoord0;
76
77      return Out;
78  };
79

```

6.4.4 Reduce the algorithm's complexity

Previously there were two cases for the fluid to move from the current position to neighbor positions. If the fluid velocity is big enough, it will be moved to only one neighbor based on its velocity direction. If the velocity is small, the fluid will be moved to all lower neighbors. After trying with many parameters to find a good result, the big velocity case was found not very necessary in GPU code although it can be faster in CPU code. Removing the big velocity case reduced many much calculations. Simulating only small velocity case seems simpler but more physically realistic. Many papers also proposed this simple implementation.

6.5 Channel Erosion

The developed erosion algorithm can create ridges, valleys on the mountain slopes and deposit sediment at the bottom of the mountain. However, the algorithm implemented in GPU is not able to simulate channel erosion which deepens and carves long gullies into the terrain plain. Figure 35 shows one example of erosion with long channels.

Implementation & Experimental Results

So far, to the best of our knowledge, there are also no works in the literature which address this issue of creating long gullies via erosion. Most works describe more physically-based methods and simulate how water moves over the terrain. As water is not moving in one direction, but splitting itself into many directions, it is not easy to create long deep gullies when using physically-based methods. This has been proved by experiment, not matter how the parameters are changed, the long deep gullies using physically based methods are not achieved.

Figure 35: Erosion with Deep and Long Gullies

A simple algorithm is proposed and written in C++ code which focuses primarily on creating ridges and gullies via the erosion process without deposition. This algorithm only moves the water in one direction which is the lowest neighbor's direction, but not splitting to any other directions. The result has been found to be quite acceptable. Figure 36, Figure 37 are the two steps used currently to create long channel erosion and Figure 38 is a closer look of channel erosion result. In step 1, the simple algorithm can be used to creating the deep and long gullies on the terrains. Our physically hydraulic erosion method will be used in step 2 to further enhance the result.

Implementation & Experimental Results

Figure 36: Step 1 - Creating the deep and long gullies on the terrains

Figure 37: Step 2 - Running erosion simulation after that

Figure 38: A closer look of the channel erosion result.

6.6 Tool to create erosion on any terrain

The erosion implementation returns a height map which can be viewed using any renderer. In order to easily create the erosion on any terrain and have better controls over the erosion algorithm, an Erosion Tool which includes a renderer was implemented. How water erodes the terrain after each simulation step can thus be visualized. Figure 39 shows the interface of this erosion tool.

Implementation & Experimental Results

Figure 39: GPU-based erosion tool

The menu in the tool includes:

- ❖ Toggle full screen: allows the user to toggle the application between full screen mode and normal mode.
- ❖ Toggle wireframe (W): allows the user see the wireframe of the terrain in the erosion process to have better control.
- ❖ Toggle ORIGINAL (O): allows the user to compare between original terrain and eroded terrain after each simulation step.
- ❖ Toggle TEXTURE (T): allows the user to compare between the terrain which is rendered in color mode and texture mode.
- ❖ Reset (R): to reset the erosion process. The original terrain will be loaded.

Implementation & Experimental Results

- ❖ Start (SPACE): to start the erosion process. This command is also used to stop the simulation.
- ❖ Step forward ->: to run only one step of the erosion simulation.
- ❖ Terrain Height: to scale the terrain at different heights.
- ❖ Open File: allows the user to load another terrain from an image file.
- ❖ Erosion Type: to select between standard erosion or channel erosion.
- ❖ Sediment Capacity: sediment capacity is one of the main parameters in erosion simulation. This sediment capacity based on 0.0f-1.0f scale will allow the user to have better control over the erosion process.
- ❖ Rain Density: similar to sediment capacity, rain density is also based on 0.0f-1.0f scale.
- ❖ Erosion Step: is the maximum number of steps which the erosion simulation will take after which it will stop automatically.

Figure 40, Figure 41 and Figure 42 are a comparison between the original terrain, the eroded terrain in color mode and the eroded terrain in texture mode of the tool. With this tool, it is very easy for the terrain designer to simulate erosion on various terrains.

6.7 Other Graphic Cards Testing

The developed GPU erosion algorithm works well with the graphics card GeForce 7900 GS in the development platform. However, because of different graphics card architectures, it may deliver different results with other graphics cards. This section shows the comparison of visual results and time performance between GeForce 7900 GS and 8600 GTS, 8800 GTX graphics card.

Implementation & Experimental Results

Figure 40: Original Terrain in Color Mode

Figure 41: Eroded Terrain in Color Mode

Implementation & Experimental Results

Figure 42: Eroded Terrain in Texture Mode

6.7.1 Testing on GeForce 8600 GTS

The following table shows the specifications of 8600 GTS graphics card.

Table 8: GeForce 8600 GTS specification

Memory	256MB
Memory Interface	128-bit
Memory Bandwidth (GB/sec)	32
Fill Rate (Billion pixels/sec)	10.8
Stream Processors	32
Core clock (MHz)	675
Shader Clock (MHz)	1450

Tested with 2Kx2K terrain, the performance of the GeForce 8600 GTS exhibit some differences with that of the 7900 GS graphics card. We are unsure as to the exact reason why these discrepancies exist, however the following points are a list of the prominent differences for readers who wish to investigate further into this issue:

Implementation & Experimental Results

- ❖ When running the simulation with 2Kx2K terrain, 16 bit floating point must be used to save data as there is not enough memory. Other graphics programs must also be closed. Although 8600 GTS has the same video memory amount 256 MB, there is no problem running the simulation using 32 bit floating point data type. The reason is unclear as we do not know the exact structure of the graphics card 8600 GTS.
- ❖ The current GPU code produces good results for the 7900 GS card. However, the results do not seem to stand for the 8600 GTS. The problem was found to be in the neighbor texture coordinates calculation. Vertex shader is currently used to calculate coordinate of the neighbors. By right, this should work for every card. However, when texture coordinate calculation is moved to pixel shader, the 8600 GTS produces correct result.
- ❖ Although when processing a 2Kx2K terrain, the 8600 GTS is able to use 32 bit floating point data types, it only produces correct result when another graphics program is running. It is not clear why when another graphics application is not running, completely different results will be produced. If this application is closed, parameters need to be changed in order to achieve a similar result. The following figures illustrate the difference.

Figure 43: Erosion result if another graphics application is not concurrently used.

Figure 44: Erosion result if another graphics application is concurrently used.

6.7.2 Testing on GeForce 8800 GTX

The following table shows the specifications of 8800 GTX graphics card.

Table 9: GeForce 8600 GTS specification

Memory	768MB
Memory Interface	384-bit
Memory Bandwidth (GB/sec)	86.4
Fill Rate (Billion pixels/sec)	36.8
Stream Processors	128
Core clock (MHz)	575
Shader Clock (MHz)	1350

The visual results of erosion simulation on the GeForce 8800 GTX do not differ much from that of the results of the erosion simulation on the 7900 GS graphics card. Obviously, since the 8800 GTX graphics card is more powerful, the erosion simulation on this card runs faster. The next section will give details of the time statistics performance differences between the three graphics cards.

Implementation & Experimental Results

6.7.3 Statistics performance between various GTX graphics cards

The following table shows the approximate time taken to finish the simulation with 1Kx1K and 2Kx2K terrains on 7900 GS, 8600 GTS and 8800 GTX.

Table 10: Time Comparison among 7900 GS, 8600 GTS and 8800 GTX graphics card

	1K x 1K terrain			2K x 2K terrain		
	7900 GS	8600 GTS	8800 GTX	7900 GS	8600 GTS	8800 GTX
50 steps	<i>11.2 s</i>	<i>5.8 s</i>	<i>4.2 s</i>	<i>26.8 s</i>	<i>12.6 s</i>	<i>9.1 s</i>
100 steps	<i>17.2 s</i>	<i>10.6 s</i>	<i>7.1 s</i>	<i>44.3 s</i>	<i>24. 2 s</i>	<i>15.5s</i>

As shown, the more powerful the graphics card, the faster the erosion simulation is.

6.8 Performance comparison with commercial erosion tool

The commercial software which was used for comparisons with our erosion algorithm is World Machine [11]. World Machine has a graph-based user interface which is easy to create erosion effects. Figure 45 shows the interface of the erosion tool of World Machine.

Implementation & Experimental Results

Figure 45: World Machine erosion tool

There are also some parameters present to control the erosion effect, for example, erosion base duration, rock hardness, sediment carry amount, filter strength. One thing to note is that, although the names are different from ours, our erosion simulation model has similar parameters to match those found in World Machine. Figure 46 shows an example of the visual results of the erosion process simulated on a 1Kx1K terrain by World Machine.

Implementation & Experimental Results

Figure 46: World Machine erosion result

Figure 47, Figure 48 show the time taken for the World Machine to create erosion on a 1Kx1K terrain and 2Kx2K terrain respectively. The platform used to test the erosion model of World Machine was also a Pentium D 3.2Ghz dual core.

Figure 47: Time taken to simulate erosion on 1Kx1K terrain in World Machine

Implementation & Experimental Results

Figure 48: Time taken to simulate erosion on 2Kx2K terrain in World Machine

Table 11 shows the time comparisons between the erosion simulation on the 3 graphics cards 7900 GS, 8600 GTS and 8800 GTX and the erosion model of the commercial software World Machine.

Table 11: Time Comparison between erosion algorithm on GPUs and World Machine

	1K x 1K terrain				2K x 2K terrain			
	7900 GS	8600 GTS	8800 GTX	World Machine	7900 GS	8600 GTS	8800 GTX	World Machine
50 step	11.2 s	5.8 s	4.2 s	12.50 s	26.8 s	12.6 s	9.1 s	118.81s
100 steps	17.2 s	10.6 s	7.1 s		44.3 s	24. 2 s	15.5s	

As World Machine is not using a physically-based erosion model, it is not fair to make any comparison based on the number of simulation steps. However, using 100 simulation steps, the time taken for GPU is still faster than the erosion simulation time in World

Implementation & Experimental Results

Machine. Specifically, it is more than 2 time faster for the 7900 GS, 5 times faster for the 8600 GTS and about 8 times faster for the 8800 GTX.

6.9 Conclusion

In this chapter, we have presented the erosion implementation on both CPU and GPU. The platform which was used to develop and test the erosion model is a Pentium D 3.2Ghz dual core equipped with NVIDIA GeForce 7900 GS graphics card. Due to programming constraints, only the more computationally costly reverse erosion algorithm can be implemented on the GPU. Despite this, the implementation on GPU still provides greater advantage in terms of speed as compared to the CPU implementation. In general, erosion simulation in GPU runs 6 to 9 times faster with the 7900 GS graphics card than in CPU. The result will be 10 to 20 times faster and 18 to 25 times faster in the case of the 8600 GTS and the 8800 GTX graphics card respectively. In comparison with the erosion tool of commercial software World Machine, the GPU-based erosion algorithm is still slightly more than 2 times faster for the 7900 GS, 5 times faster for the 8600 GTS and about 8 times faster for the 8800 GTX. This, in general, has shown the tremendous advantage in computational power of the GPU over the CPU.

CHAPTER 7

7 Conclusion and Future Work

This chapter concludes this report by reviewing the proposed method and conceptual models involved. Future works and suggestion for improvements are also included in this chapter.

7.1 Research and Development Results

Visual simulation of natural scenery has always been a popular research field in computer graphics. There have been two main terrain synthesis works developed: fractal terrain modeling and physical hydraulic erosion. Initially, fractal terrains were credible imitations of natural mountain terrain. However, closer examination reveals an unnatural characteristic in these surfaces. In general, fractal terrains have no global erosion features inherently due to isotropy. These features are the result of interaction between the terrain and running water on its surface. Erosion can result in the reallocation of sediment on the terrain surface and modification of terrain relief.

Physically-based hydraulic erosion methods have been proposed to solve this problem. The methods not only create erosion ridges, canyons and gullies but can add fractal details to the terrain as well. Although the results from most of the current hydraulic erosion algorithms are realistic, they are only tested on small or low resolution terrains. The existing works in the literature have mainly been focused on creating realistic erosion shapes, but not on the speed of the simulation. The latest erosion paper in 2005 tested its proposed algorithm on a small terrain with only one mountain in 256x256 grid cells. Due

Conclusion and Future Work

to the computationally expensive nature of that proposed method, it runs, when tested on a 2.4 GHz Pentium IV PC, at only 4 frames per second.

Simulating current hydraulic erosion method on larger terrains, such as 2048x2048 grid cells, takes an even longer period of time. As a result, many applications currently have to generate terrain models in advance and store them at high resolution. This might prove a problem in platforms with limited storage space. Also, there is very little flexibility for generating terrains with realistic looking erosion features for applications.

Recently, graphics hardware has evolved into an indispensable tool in improving the speed of computation. Although there are many limitations, such as the GPU's unusual programming model and the difficulties in applying GPU processing to non-graphics tasks, GPUs are still an ideal platform for computationally challenging tasks in many application domains. The biggest motivation to use the GPU is its tremendous computational power and inexpensive, but limited, memory bandwidth. The computational capabilities of GPUs have compounded at an average yearly rate of 1.7 (pixels/second), compare to a yearly rate of roughly 1.4 for CPU performance. Modern GPUs now include deeply programmable processing units that support vectorized floating-point operations for programmable pixel, vertex and geometry engines. This is a great motivation for us to develop an erosion algorithm using the General Purpose Computation on Graphics Processing Unit (GPGPU) for faster terrain generation.

In this thesis, we have presented a novel erosion algorithm which can be implemented on both CPU and GPU. Our method is based on a Newtonian physics approach which is implemented on a two-dimensional data structure which stores height fields, water amount, and dissolved sediment and water velocities. The approach has been optimized in order to be implemented in a shading language. The program can simulate large size

terrains, i.e. 2Kx2K terrains on a graphics card with limited amounts of memory, such as the GeForce 7900 GS. To the best of our knowledge, there are no works in academic literature which have been able to develop an algorithm to simulate erosion on a large scale terrain to be used in real graphics applications. The platform which was used to test our model is equipped with a Pentium D 3.2 GHz dual core and an NVIDIA GeForce 7900 GS graphics card. Due to programming constraints, the erosion algorithm must be implemented on GPU reversely even though this implementation is computationally more taxing. Despite this, the GPU-based erosion program still provides greater advantage in term of speed and efficiency. In general, compared with the CPU implementation, the erosion simulation in GPU runs six to nine times faster for the case of the 7900 GS graphics card. This performance boost is even more significant if more powerful graphics cards are used. With the 8600 GTS and the 8800 GTX graphics cards, it is ten to twenty times faster and eighteen to twenty-five times faster respectively.

The erosion algorithm implemented on GPU has also been shown to provide faster simulation times than the erosion tool of commercial software World Machine. In large-scale terrain erosion simulation, the GPU-based erosion implementation is more than 2 times faster on the 7900 GS graphics card, 5 times faster on the 8600 GTS graphics card and about 8 times faster on the 8800 GTX graphics card. This demonstrates the tremendous advantage in computational power of the GPU over the CPU.

7.2 Published Paper

The following paper has been published by ACM in the proceedings of Graphite 2007, 5th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, sponsored by SIGGRAPH, ACM Special Interest Group on Computer Graphics and Interactive Techniques:

- ❖ A. Nguyen Hoang, S. Alexei, and A. Parimal, "Physically based hydraulic erosion simulation on graphics processing unit," in Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia Perth, Australia: ACM, 2007.

7.3 Seminar Presentation

The work in this project also had been chosen for presentation at GameScape 2007, a showcase of Nanyang Technological University (NTU) gameLAB's current Research and Development work. It was held on 13 December 2007 in NTU.

7.4 Future work

Although the GPU erosion implementation shows good results with fast computational times, there are still places for further improvement.

7.4.1 *Revisit the State of the art in the erosion field*

There are a few papers which have just been published in the same hydraulic erosion field. More importantly, Stava and Rianto have already cited the author's work in their state of the art section. For further improvement, the author can continue to analyze and research these papers:

- ❖ M. Xing, P. Decaudin, H. Bao-Gang, "Fast Hydraulic Erosion Simulation and Visualization on GPU", in Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, Oct. 29 – Nov 2, pp. 47 – 56, 2007. [69]
- ❖ O. Stava, B. Benes, M. Brisbin, and J. Krivanek, "Interactive Terrain Modeling using Hydraulic Erosion", in Proceedings of ACM SIGGRAPH/ Eurographics Symposium on Computer Animation, pp. 30-39, 2008.
- ❖ O. Stava, B. Benes, M. Brisbin, and J. Krivanek, "Interactive Hydraulic Erosion on the GPU", in ShaderX7 – Advanced Rendering Techniques, 2008.

- ❖ S. Rianto, L. Li, and B. Hartley, "Fluid Dynamic Simulation for Cutting in Virtual Environment." [70]

7.4.2 *New shading CUDA language*

The current GPU-based erosion algorithm was developed in the graphics shading language HLSL. However, currently there is a new GPGPU technology named CUDA (Compute Unified Device Architecture), developed by NVIDIA, which allows for easier GPU programming without the knowledge of graphics APIs. CUDA works with the new GeForce 8 Series, which features the G8X GPUs. An example of such a card is the 8800 GTX graphics card. By opening up the architecture, CUDA provides developers both with the low-level, deterministic, and the high-level API for repeatable access to hardware which is necessary to develop essential high-level programming tools such as compilers, debuggers, math libraries, and application platforms.

The followings are several advantages of CUDA over traditional GPGPU using graphics APIs:

- ❖ It uses the standard C language, simple extensions.
- ❖ Scattered writes – data can be written to arbitrary addresses in memory.
- ❖ Shared memory – CUDA exposes a fast shared memory region that can be shared amongst threads.
- ❖ Faster downloads and readbacks to and from the GPU
- ❖ Full support for integer and bitwise operations

With CUDA, developers can use the C programming language to code algorithms for execution on the GPU. As a trend in GPGPU, the erosion algorithm can be ported to the CUDA language to take advantage of features on future graphics cards for better performance.

7.4.3 *Simulate erosion in caves*

One of the weaknesses of the current erosion algorithm is the inability to simulate erosion in caves such as shown in the Delicate Arch in Figure 49. The challenge lies in coming up with the data structure needed to save the terrain height data and other erosion simulation data such as fluid amount, sediment amount, etc. Voxel or layered-data structure cannot be used for erosion simulation in this case. As a future work, the erosion algorithm can be improved upon to handle such phenomena.

Figure 49: Delicate Arch, located at Arches National Park in Moab, Utah. The LaSalle Mountains are seen in the background.

7.4.4 *High resolution terrain generation*

The objective of this project is to propose a fast procedural technique, which utilizes the powerful computational capabilities of graphics hardware, in order to allow applications to ship with lower resolution height map data, such as United States Geological Survey (USGS) Digital Elevation Model (DEM) data, and perform erosion at load time, thus generating a highly detailed high-resolution terrain from the height map data.

An erosion algorithm which meets the objective stated above has been proposed in this report. However, the problem of generating a high resolution terrain from low resolution

terrain data has not been addressed yet. As a future work, methods to interpolate the terrain data from small resolution such as 256x256 to higher resolutions such as 2048x2048 or 4096x4096 will be considered and developed to meet the requirement of the proposed erosion algorithm.

7.4.5 Texture Generation at loading time

The current erosion algorithm changes the terrain height map. It can be further extended for terrain texture generation at application load time. Figure 50 on the next page shows an example of mountains with erosion in our current game application. This image is the result of applying the sample erosion texture shown in Figure 51 on the mountain cliff at load time. As we can see, the image does not look realistic due to simple texture generation. As a future work, our current erosion algorithm can be used in conjunction with other texture generation methods to generate more realistic textures for the current game's terrain.

Figure 50: Texture is used in current game

Figure 51: Texture applied on mountain cliff

REFERENCE

- [1] A. Nguyen Hoang, S. Alexei, and A. Parimal, "Physically based hydraulic erosion simulation on graphics processing unit," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia* Perth, Australia: ACM, 2007.
- [2] O. Deussen and B. Lintermann, *Digital Design of Nature: Computer Generated Plants and Organics*: Springer, 2005.
- [3] D. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, "Texture and Modeling: A Procedural Approach," *Morgan Kaufmann, San Francisco*, 2003.
- [4] B. B. Mandelbrot and J. A. Wheeler, *The Fractal Geometry of Nature* vol. 51, 1983.
- [5] R. F. Voss, "Random fractal forgeries," *Fundamental Algorithms for Computer Graphics*, vol. 835, 1985.
- [6] B. J. Skinner and S. C. Porter, *The Dynamic Earth: An Introduction to Physical Geology*: Wiley, 1995.
- [7] F. K. Musgrave, C. E. Kolb, and R. S. Mace, "The synthesis and rendering of eroded fractal terrains," in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, 1989, pp. 41-50.
- [8] B. Neidhold, M. Wacker, and O. Deussen, "Interactive physically based Fluid and Erosion Simulation," in *Proceedings of Eurographics Workshop on Natural Phenomena*, 2005, pp. 25–32.

- [9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Eurographics 2005, State of the Art Reports*, pp. 21–51, 2007.
- [10] D. S. Ebert, "Procedural Techniques and Real-Time Graphics " 2002.
- [11] S. Schmitt, "World Machine," <http://world-machine.com/>, 2006.
- [12] NVIDIA Corporation, "SUPERCOMPUTING 2006 Tutorial on GPGPU," 2006.
- [13] NVIDIA Corporation, "SUPERCOMPUTING 2007 Tutorial: High Performance Computing with CUDA." vol. 2007, 2007.
- [14] I. Buck, K. Fatahalian, and P. Hanrahan, "GPUBench: Evaluating GPU performance for numerical and scientific applications," *2004 ACM Workshop on General-Purpose Computing on Graphics Processors*, 2004.
- [15] INTEL, "Intel Processor List," <http://www.intel.com/products/desktop/-processors/index.htm?iid=process+desktop>, 2007.
- [16] M. Ekman, F. Warg, and J. Nilsson, "An in-depth look at computer performance growth," *ACM SIGGRAPH Computer Architecture News*, vol. 33, pp. 144-147, 2005.
- [17] D. Luebke, M. Harris, J. Krüger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, and A. Lefohn, "GPGPU: general purpose computation on graphics hardware," *Proceedings of the conference on SIGGRAPH 2004 course notes*, 2004.
- [18] R. Fernando and NVIDIA Corporation., *GPU Gems: programming techniques, tips, and tricks for real-time graphics*. Boston, MA: Addison-Wesley, 2004.
- [19] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 703-712, 2002.

- [20] J. Owens, "Streaming architectures and technology trends," *International Conference on Computer Graphics and Interactive Techniques*, 2005.
- [21] M. Pharr and R. Fernando, *GPU Gems 2: programming techniques for high-performance graphics and general-purpose computation*. Upper Saddle River, NJ: Addison-Wesley, 2005.
- [22] A. Lefohn, J. Kniss, and J. Owens, "Implementing efficient parallel data structures on GPUs," *GPU Gems*, vol. 2, pp. 521–545, 2005.
- [23] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 777-786, 2004.
- [24] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: a system for programming graphics hardware in a C-like language," *International Conference on Computer Graphics and Interactive Techniques*, pp. 896-907, July 2003.
- [25] K. Gray, *Microsoft DirectX 9 Programmable Graphics Pipeline*: Microsoft Press Redmond, Wash, 2003.
- [26] W. F. Engel, *Programming vertex and pixel shaders*, 1st ed. Hingham, Mass.: Charles River Media, 2004.
- [27] J. Kessenich, D. Baldwin, and R. Rost, "The OpenGL Shading Language version 1.10.59," Go online to <http://www.opengl.org/documentation/oglsl.html>, 2004.
- [28] M. McCool, S. Du Toit, T. Poppa, B. Chan, and K. Moule, "Shader algebra," *International Conference on Computer Graphics and Interactive Techniques*, pp. 787-795, 2004.
- [29] A. Bleiweiss and A. Preetham, "Ashli—Advanced Shading Language Interface," *Real-Time Shading Course, Siggraph*, 2003.

- [30] D. D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*: Prentice Hall Professional Technical Reference, 2003.
- [31] E. K. Koh and D. D. Hearn, "Fast Generation and Surface Structuring Methods for Terrain and Other Natural Phenomena," *Comput Graph Forum*, vol. 11, pp. 169-180, 1992.
- [32] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Communications of the ACM*, vol. 25, pp. 371-384, 1982.
- [33] G. S. P. Miller, "The definition and rendering of terrain maps," *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 39-48, 1986.
- [34] J. P. Lewis, "Generalized Stochastic Subdivision," *ACM Transactions on Graphics*, vol. 6, pp. 167-190, 1987.
- [35] R. Szeliski and D. Terzopoulos, "From splines to fractals," in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, 1989, pp. 51-60.
- [36] N. Sala, S. Metzeltin, and M. Sala, "Applications of Mathematics in The Real World: Territory And Landscape," *University of Italian Switzerland*, 2002.
- [37] A. D. Kelley, M. C. Malin, and G. M. Nielson, "Terrain simulation using a model of stream erosion," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, 1988, pp. 263-268.
- [38] K. Nagashima, "Computer generation of eroded valley and mountain terrains," *The Visual Computer*, vol. 13, pp. 456-464, 1998.
- [39] F. Belhadj and P. Audibert, "Modeling landscapes with ridges and rivers: bottom up approach," in *Proceedings of the 3rd international conference on Computer*

- graphics and interactive techniques in Australasia and South East Asia*, 2005, pp. 447-450.
- [40] P. Roudier, B. Peroche, and M. Perrin, "Landscapes Synthesis Achieved through Erosion and Deposition Process Simulation," *Computer Graphics Forum*, vol. 12, pp. 375-383, 1993.
- [41] N. Chiba, K. Muraoka, K. Yaegashi, and M. Miura, "Terrain simulation based on the recursive refinement of ridge-lines," *Proc. CAD/Graphics' 91*, pp. 19-24, 1991.
- [42] N. Chiba, K. Muraoka, and K. Fujita, "An erosion model based on velocity fields for the visual simulation of mountain scenery," *The Journal of Visualization and Computer Animation*, vol. 9, pp. 185-194, 1998.
- [43] B. Benes and R. Forsbach, "Visual Simulation of Hydraulic Erosion," *Journal of WSCG*, vol. 1, pp. 79-86, 2002.
- [44] J. Olsen, "Realtime Procedural Terrain Generation," *Published Oct*, 2004.
- [45] B. Benes and R. Forsbach, "Layered data representation for visual simulation of terrain erosion," *Spring Conference on Computer Graphics*, pp. 80-86, 2001.
- [46] M. Kass and G. Miller, "Rapid, stable fluid dynamics for computer graphics," *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 49-57, 1990.
- [47] J. Stam, "Stable fluids," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 121-128.
- [48] T. T. Wong, W. S. Luk, and P. A. Heng, "Sampling with Hammersley and Halton points," *Journal of Graphics Tools*, vol. 2, pp. 9-24, 1997.
- [49] B. Benes, Arriaga, "Table Mountains by Virtual Erosion," in *Proceedings of Eurographics Workshop on Natural Phenomena*, 2005, pp. 33-40.

- [50] B. Benes, V. Tesinsky, J. Hornys, and S. Bhatia, "Hydraulic erosion," *Computer Animation and Virtual Worlds* 17 (2), pp. 99–108, 2006.
- [51] B. Benes, "Hydraulic Erosion: A Survey," 2006.
- [52] F. F. S. M. C. S. John Brosz, "Terrain synthesis by-example," *GRAPP 2006: Proceedings of the First International Conference on Computer Graphics Theory and Applications*, pp. 122-133, February 25-28 2006.
- [53] J. P. Lewis, "Texture synthesis for digital painting," *ACM SIGGRAPH Computer Graphics*, vol. 18, pp. 245-252, 1984.
- [54] K. Perlin and L. Velho, "Live paint: painting with procedural multiscale textures," *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 153-160, 1995.
- [55] Howard Zhou, Jie Sun, Greg Turk, and J. M. Rehg, "Terrain Synthesis from Digital Elevation Models," *IEEE Transactions on Visualization and Computer Graphics : Accepted for future publication*, vol. 13, 2007.
- [56] N. Foster and R. Fedkiw, "Practical animation of liquids," *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 23-30, 2001.
- [57] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graphical Models and Image Processing*, vol. 58, pp. 471-483, 1996.
- [58] J. Fo'Brien, J. K. Hodgins, T. Georgia Institute of, and G. Visualization and Usability Center, *Dynamic Simulation of Splashing Fluids*: Graphics, Visualization and Usability Center, College of Computing, Georgia Institute of Technology, 1994.
- [59] N. Foster and D. Metaxas, "Controlling fluid animation," *Computer Graphics International*, vol. 97, pp. 178-188, 1997.

- [60] J. Dorsey, A. Edelman, H. W. Jensen, J. Legakis, and H. K. Pedersen, "Modeling and rendering of weathered stone," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 1999, pp. 225-234.
- [61] N. Ozawa and I. Fujishiro, "A morphological approach to volume synthesis of weathered stones," *Volume Graphics workshop'99*, pp. 207-220, 2000.
- [62] M. Carlson, P. J. Mucha, and G. Turk, "Rigid fluid: animating the interplay between rigid bodies and fluid," *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 377-384, 2004.
- [63] J. Stam, "Real-time fluid dynamics for games," *Proceedings of the Game Developer Conference*, vol. 18, 2003.
- [64] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 154-159, 2003.
- [65] N. Holmberg and B. C. Wünsche, "Efficient modeling and rendering of turbulent water over natural terrain," *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and Southe East Asia*, pp. 15-22, 2004.
- [66] E. W. Weisstein, "Hammersley Point Set," <http://mathworld.wolfram.com/HammersleyPointSet.html> ed: From MathWorld-- A Wolfram Web Resource. .
- [67] P. H. Baxendale and S. V. Lototsky, *Stochastic differential equations : theory and applications*. Singapore: World Scientific, 2007.
- [68] F. D. Luna, *Introduction to 3D Game Programming with DirectX 9*: Wordware Publishing, 2003.

- [69] X. Mei, P. Decaudin, and B. G. Hu, "Fast Hydraulic Erosion Simulation and Visualization on GPU," 2007, pp. 47-56.
- [70] S. Rianto, L. Li, and B. Hartley, "Fluid Dynamic Simulation for Cutting in Virtual Environment," *The 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) 2008, Plzen , Czech Republic*, pp. 119-125, 2008.

APPENDIX A: Hydraulic Erosion Process

Water distribution over the terrain

Water movement from high positions to lower positions

Water moves down and erodes the higher portions of the terrain

Water evaporates and leaves the deposition mostly on the bottom portions of the terrain

APPENDIX B: Comparison of 1Kx1K Original and Eroded Terrain (CPU version)

Original Terrain

Eroded Terrain after 50 steps of simulation

APPENDIX C: Comparison of Original and Eroded Terrain with Deposition (CPU)

Original Top View

Eroded Terrain with Deposition

APPENDIX D: Comparison of 1Kx1K Eroded Terrain and 2Kx2K Eroded Terrain (CPU)

Eroded 1Kx1K Terrain

Eroded 2Kx2K Terrain

APPENDIX E: Comparison of 2Kx2K CPU Eroded Terrain and GPU Eroded Terrain

CPU version

GPU version