University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Diploma Thesis

Modeling of Erosion Impacts on the Terrain

Pilsen 2012 Věra Skorkovská

Declaration of Authorship

I hereby declare that this diploma thesis is completely my own work and that I used only the cited sources.

Pilsen, May 14, 2012

Věra Skorkovská

Acknowledgements

I wish to express my gratitude to all the people who supported me during the realization of this thesis. First and foremost, I would like to thank to my supervisor Prof. Dr. Ing. Ivana Kolingerová for her valued advices and never-ending patience. This thesis would not have been possible without Doc. Ing. Bedřich Beneš, PhD., whose experience in this field of research was very helpful. My thanks also go to my family and my friends whose support was very important to me during my whole studies.

Abstract

Erosion-based terrain modeling has been an important part of computer graphics for more than twenty years but many problems still remain unsolved. Many solutions are only capable of working with 2.5D terrain, these solutions do not allow formations such as caves or overhangs. Other group of solutions supports the fully 3D terrain but these methods are usually very memory consuming and not capable of running with real-time response. This thesis proposes a solution to hydraulic erosion; the method is capable of working with fully 3D terrain and representing fully 3D water effects. The terrain is stored as a triangular mesh which allows to adaptively change its resolution based on the topology of the scene. The use of the triangle mesh data structure introduces some new problems, e. g., the irregularity of the mesh has to be taken into account during the calculations. The main drawback of this solution are the inconsistencies of the mesh that can be created during the erosion simulation. The solution to this problem is suggested in the work, however, it is not yet successfully implemented, due to the numerical imprecision issues.

This work has been supported by the Ministry of Education, Youth and Sports, project LH11006 Interactive Geometrical Models for Simulation of Natural Phenomena and Group Behavior.

Contents

List of Figures

1 Introduction

In the field of computer graphics, we often find ourselves in the need of visually plausible models of terrain. Creating such a terrain using a modeling software would be very time-consuming and the results may not be as good as we would need. Many terrain generation methods have been introduced since the dawn of the research in this area of computer graphics.

Most of the modern approaches to terrain generation are using erosion processes to create a realistic scene. In nature, erosion has the main influence on how the terrain changes with the passing years. Hydraulic erosion is causing the biggest alterations on terrain and because of that it has been in the spotlight of the terrain modeling research. The impacts of weathering and wind erosion are not as significant but can be essential for certain types of terrain.

The methods for erosion-driven terrain modeling which are physically based can be as well used in simulation of erosion impact, such as simulation of impacts of rainfall, floods or simulation of terrain evolution over the years.

1.1 Aim of the Thesis

The aim of this thesis is to summarize the state of the art of erosion based terrain modeling and propose and implement a solution which will address some of the pending problems of this area of the computer graphics.

Many algorithms have been presented in this field of computer graphics. Most of the algorithms are solving the hydraulic erosion as it has the greatest influence on the terrain alterations. Many are working with a height field data structure which allows the creation of fast interactive solutions but its main disadvantage lies in the impossibility of simulating a fully 3D scene. Other methods are addressing this problem by proposing a way to simulate a fully 3D terrain using a uniform voxel grid but these methods are far from being interactive and are very memory consuming. Our goal is to design a solution to hydraulic erosion problem that would work with fully 3D terrains and would be spatially adaptive, resulting in lower memory consumption.

1.2 Proposed Solution

This work has been supported by the Ministry of Education, Youth and Sports, project LH11006. The solution was proposed by Prof. Dr. Ing. Ivana Kolingerová, Ing. Bedřich Beneš, PhD., and me, Bc. Věra Skorkovská during a business trip to the Purdue University in West Lafayette, U.S.A., which took place in November of 2011 and was financed by the project LH11006.

The solution is based on Smoothed-particle hydrodynamics (SPH) fluid simulation interacting with objects represented by surface triangle meshes. The advantage of the SPH simulation is that the particles are localized only in the regions with fluids and so we can limit the computation to this locations. Similarly, the triangle mesh resolution can be adjusted according to the complexity of the scene, allowing higher resolution in the regions with great details and lower resolution in the homogeneous regions. The SPH particles will simulate the fluid movements and interact with the mesh when collision occurs. Neither SPH nor the triangle mesh gives us the information about the spatial distribution of the elements, for that reason, auxiliary data structures will be used to speed up the algorithm.

1.3 Text Structure

In Chapter 2 the state of the art of erosion techniques will be described, in Chapter 3 the fluid simulation and 3D hydraulic erosion methods will be presented in more detail. In Chapter 4 a summary of the data structures commonly used in erosion simulation will be given. Chapter 5 will describe the methods and data structures used in the proposed solution, while Chapter 6 will offer a detailed description of the algorithm. In Chapter 7 the results will be presented and Chapter 8 will conclude the work.

2 Erosion

Erosion-based terrain modeling has been an important chapter of computer graphics for more than two decades but there is still a lot of work to be done. The methods used in erosion based terrain modeling have changed very much since the beginning of the research.

The first attempts to create artificial terrains tried to generate plausible terrains straight away but the terrains generated this way usually looked too "sharp" and unrealistic. As the computer performance improved, new approaches to terrain modeling appeared. Most of the new methods were based on erosion. The authors realized that the best way to imitate a real terrain is to take an artificial terrain as a base (e.g., a terrain created using a fractal geometry [Man82]) and simulate the erosion processes that alter it. Erosion as it takes place in the nature is a very complex process and so it is not easy to simulate. Even with the modern powerful computers it is not possible to create an erosion simulation that would be both physically exact and at the same time running in real-time. Every developer then has to choose between these two options and decide where are his preferences.

Erosion is a process happening in the nature by which the material such as sand and rocks is taken from its original location and then moved and deposited in other location by the means of water, wind or other natural forces. Erosion processes can be subdivided into three main categories: weathering, hydraulic erosion and wind erosion. Hydraulic erosion is leaving the most significant mark in the landscape and that is the reason why most of the algorithms are addressing it. Hydraulic erosion is not only the erosion caused by rain and flowing or still water, erosion caused by glaciers and avalanches also fall into this category. Wind erosion is not causing such big changes generally but in certain landscapes such as the desert sceneries its effect can be essential. In the rest of this chapter these processes will be described in more detail and the current state of the art in each of the categories will be summarized.

2.1 Hydraulic Erosion

Hydraulic erosion methods can be subdivided into two categories. The first category contains physically inspired methods - methods, which take inspiration in natural processes but are not trying to simulate them exactly. Their main purpose is to mimic the erosion impacts with as little computational effort as possible. The other category incorporates physically based methods. These methods are based on hydrodynamics but usually introduce some simplifications in order to speed up the simulation.

2.1.1 Physically Inspired Solutions

The first attempts to create a hydraulic erosion simulation were limited by the computational force of the computers at the time. The methods were designed so that the results ”look good” without the need of simulating the physical erosion processes.

Musgrave et al. [MKM89] proposed a simple hydraulic erosion algorithm simulating rain effects on the terrain. The method consists of depositing water (rain) on vertices of the height field. The water erodes the terrain and moves the sediment to lower locations. The implementation is done by associating the volume of the water and the amount of sediment with each of the vertices in the height field.

Chiba et al. [CMF98] introduced a method based on velocity fields of water flow. They are using particles to approximate the water volume, the erosion is calculated when a particle collides with the terrain. The algorithm is designed to simulate natural ridges and valleys.

Beneš and Forsbach [BF02] describe a model for hydraulic erosion caused by running water. The process consists of four independent steps that can be repeatedly applied to achieve the visual effect that is desired. At first, the water appears at some locations, simulating rain or water sources. Then the water erodes or dissolves the material and captures the sediment which is transported in the third phase. The final step representing the deposition process is affected by two factors. The water slows down and the sediment settles on the ground as the water flow is not strong enough to carry it anymore. The second factor affecting deposition is evaporation of the water which causes the excess of sediment in the particles.

2.1.2 Physically Based Solutions

Fluid Dynamics

The fluid dynamics is described by Navier-Stokes equations for incompressible Newtonian fluids, a set of partial differential equations that are supposed to hold throughout the fluid. The equations are as follows [Ach90]:

$$\rho\left(\frac{\partial \vec{v}}{\partial t} + \vec{v}\cdot\nabla\vec{v}\right) = -\nabla p + \mu\nabla^2\vec{v} + \rho\vec{f}, \tag{2.1}$$

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\vec{v}) = 0 \tag{2.2}$$

The symbol $\vec{v}\,[m\cdot s^{-1}]$ is used for the velocity of an infinitesimal element of mass at a point in space. The letter $p\,[Pa]$ stands for the pressure at that point, $\rho\,[kg\cdot m^{-3}]$ substitutes the density of the fluid and is assumed to be constant throughout the whole volume of the fluid, $\vec{f}\,[N\cdot m^{-3}]$ stands for external forces. The constant $\mu\,[Pa\cdot s]$ represents the viscosity of the fluid. [Dav11], [Hib10]

Equation (2.1) is called the *momentum equation* and describes the behavior of the fluid due to the forces acting on it. Equation (2.2) is called the *conservation of mass*. Real fluid can change its volume but the changes are so small that we are not able to visually perceive them. The volume changes have such a tiny effect on how the fluid moves that it is practically irrelevant in the field of computer animation. This is a very import fact leading to a simplification and treating the fluids as being incompressible.

Shallow Water Equations

Shallow Water equations are a simplification of Navier-Stokes equations for fluid dynamics. The Shallow Water model does not allow overlaps such as breaking waves or splashes - the water surface is stored as a height field resulting in only 2.5D water effects. Second simplification of the method comes from the fact that we are assuming the water to be shallow, allowing us to ignore the vertical component of the velocity of the water. The last assumption is that the horizontal component of the velocity is constant in the whole vertical column. These simplifications limit the use of the method but from the experience it is satisfactory for many simpler cases [KM90].

The Shallow Water model was used to create interactive hydraulic erosion simulation, e.g., in papers by Mei et al. [MDH07] and Beneš [Ben07].

3D Water Simulation

With the improvements in computational force of computers the effort to create a physically based simulation prevailed but even nowadays the simulations that work in fully 3D scenes are far from being interactive. More realistic results can be achieved with more precise simulation of the fluid physics and so many algorithms are working with Navier-Stokes equations that describe fluid dynamics. The matter of fully 3D fluid simulations is so vast that it is reasonable to dedicate a separate Chapter 3 to talk about it in more detail.

2.2 Wind Erosion

Wind erosion has a major influence in arid landscapes where the hydraulic erosion is nearly absent. Wind erosion consist of two main parts, abrasion of rocks and transferring the material particles. Abrasion happens when the wind carries material particles and hits the solid surface of a rock causing small pieces of material to fall off. Wind can also capture the particles and move them to other location creating such formations as sand dunes or wind-ripples.

Onoue and Nishita [ON00] proposed a method for modeling desert sceneries. They use a height field terrain model and divide the erosion process into two parts, saltation and creep. Saltation is the effect when wind grabs a particle and lifts it to move it, creep on the other hand is occurring when the wind is moving the particle on the surface. Equations describing the processes are proposed in the paper, resulting in the creation of sand dunes and wind-ripples. The two are created separately and then combined during rendering using bump-mapping technique. Their results can be seen in Figure 2.1.

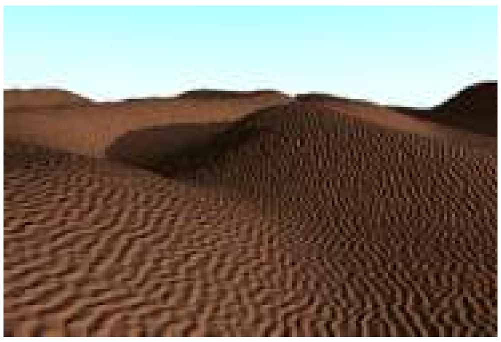


Figure 2.1: Desert scenery with wind-ripples. [ON00]

Beneš and Roa extended their algorithm by adding interaction with obstacles [BR04]. The material is accumulated on the windward side of the obstacles and on the leeward side the wind shadow appears. The intensity of the wind is reduced in the wind shadow according to the simplified geometry of the obstacle, causing the reduction of the wind-ripples. The accumulation on the windward side is done by extension of the saltation and creep algorithm - if the particle is moved and the final position is inside an obstacle, it is moved to the boundary. An example of generated scenery is shown in Figure 2.2.

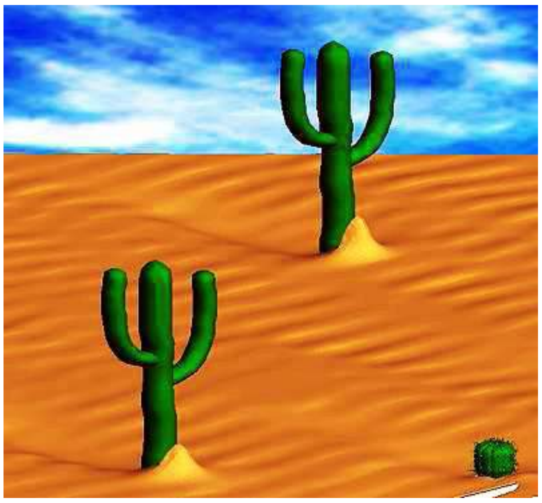


Figure 2.2: Material is accumulated on the windward side. [BR04]

Miao et al. [MMW01] introduced a method to simulate the initiation and evolution of wind blown sand ripples and dunes. Their model is capable of reproducing sand ripples and describe the reparation of a destroyed rippled surface. They also include an algorithm for sliding when the gradient of the sand slope is greater than the angle of repose.

Hatano and Hatano [HH01] published a method producing dune patterns as barchans and linear dunes from the initial random state. They were studying the efficiency of sand transport which turned out to be the most efficient for the linear transverse dune and least efficient when no pattern was formed.

2.3 Weathering

Weathering is the process of breaking down rocks, e.g., due to the contact with chemical substances, living organisms or due to the thermal changes. Weathering happens in place without the material transferring to other locations, so it is not an erosion process in the strict meaning of the term. But in the field of computer graphics it is commonly ranked as erosion as it is often coupled with other erosion processes.

Thermal weathering is the weathering process which is being simulated most often. Thermal weathering is caused by expanding and contracting the material due to the temperature changes. This process is most obvious in the deserts, where the temperatures between the day and night vary greatly. The material drops of the rocks and falls down to the ground, where it creates talus slopes with the critical angle defined by the material. When the angle of the talus exceeds the critical value, movement occurs and some material slides down.

One of the first algorithms on this matter was introduced by Musgrave et al. [MKM89]. They presented a new two-step approach to the synthesis of fractal terrain height fields with local control of fractal dimension. In the first pass, a fractal terrain is generated using a noise function to locally influence the smoothness and asymmetry of the terrain. Then in the second pass simplified global erosion is applied. At each time step they compare the difference in heights of each vertex and its neighbors and if the slope is greater than the critical talus angle, some of the material is moved to the lower neighbors. Using this simple algorithm the scene converges to stable slopes, many other researchers have taken it over and used it in their simulations.

Beneš and Arriaga in [BA05] present a method designed to generate table mountains. Their goal is a geologically inspired algorithm which can simulate visually plausible terrains. They are working with two types of material:

hard rock and soft eroded material (sand). The hard material erodes when exposed to moisture and thermal changes and the eroded parts are falling off and becoming the soft material. The motion of the soft material is simulated by a diffusion algorithm.

Dorsey et al. [DEJ$^+$99] presented a method for modeling and rendering of a weathered stone. They proposed a new voxel surface-aligned data structure that works as an intermediate between the stone and the surrounding erosion factors. They designed an algorithm to simulate the flow of water and dissolution and transportation of minerals that causes the surface erosion.

Another method that works with 3D objects was proposed by Jones et al. [JFBB10]. They presented an algorithm for spheroidal and cavernous weathering of rocks with concave surfaces which allows the user to control the durability of the material and by doing so affecting the resulting scene. The algorithm is built on a voxel grid and the erosion is calculated through the mean curvature estimation. The method is not physically accurate but produces visually plausible results which can be used in computer animation or games (see Figure 2.3).

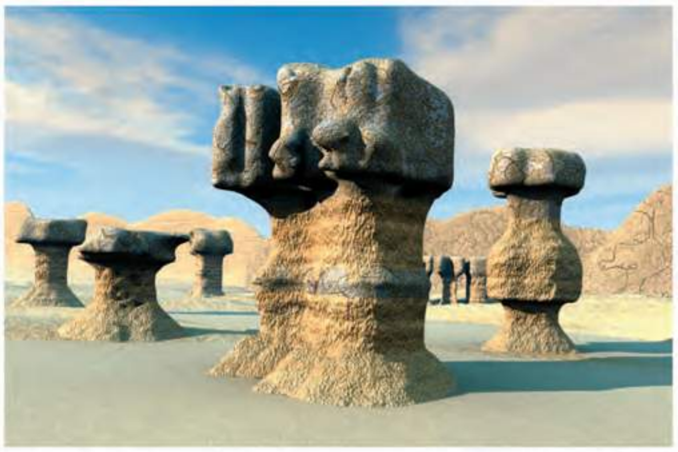


Figure 2.3: A scene containing several weathered rock shapes created using the weathering simulation. [JFBB10]

3 3D Fluid Simulation

This chapter describes some of the simplifications of Navier-Stokes equations (see Section 2.1.2 for more details) that are commonly used in computer graphics. For a more detailed description of the use of fluid simulation in computer graphics the reader can refer to [Bri08].

There are two approaches to solving the Navier-Stokes equations in computer graphics, the Langrangian approach representing the fluid with particles and the Eulerian approach dividing the volume and tracking fluid quantities at fixed points in the space.

3.1 Eulerian Approach

The Eulerian approach looks at the volume of the fluid and tracks the fluid quantities at fixed points. As the fluid moves, it goes through these fixed points, causing the tracked quantities to change. This approach usually divides the space into a uniform grid, making the necessary computations such as pressure gradients somewhat easier. This approach leads to more accurate results (compared to the Lagrangian approach) but the algorithms are computationally expensive. Another disadvantage of this viewpoint is the uniform space division. If we are trying to represent a vast nonuniform landscape, we will still have to create the grid even in the uninteresting regions where no changes are happening, with the same resolution as in the regions with the most detail. That will result in huge amounts of data and the resulting algorithms will be very memory consuming.

This approach is used in [BTHB06] to create a fully 3D simulation of hydraulic erosion. Their solution requires a model of the environment as a regular grid. Each voxel is classified into one of the following classes - FULL, the voxel is full of water, it can contain dissolved material; EMPTY, an empty voxel containing only air; MAT, a voxel containing material. A voxel can change its state from FULL to MAT by depositing the material and from MAT to FULL by erosion. The authors present solutions for both cohesive and cohesionless material capable of fully 3D water effects. The main disadvantage of their application is that it is not capable of running interactively.

3D Eulerian approach is used in a paper by Wojtan et al. [WCMT07] to simulate a corrosion and erosion of solid objects. They store the surface as a level set [1] and simulate the erosion by advecting it inward and the deposition by advecting the level set outward.

3.2 Lagrangian Approach

On the other hand, the Lagrangian approach represents the fluid as a particle system. The fluid volume is treated as a set of separate particles, each of them has its own position, velocity and other data. This representation makes some things much simpler, e.g., the *mass conservation* condition (Equation (2.2)) is automatically satisfied, provided the particles do not disappear. It also addresses the disadvantage of the Eulerian approach with nonuniform scenes as the calculations are done only in the regions where the fluid is present. Generally the particle based methods are less accurate than the grid based due to difficulties in dealing with spatial derivatives on an unstructured particle cloud but are much faster which allows their use in real-time applications. [BS09]

3.2.1 Smoothed-Particle Hydrodynamics

Smoothed-particle hydrodynamics (SPH) is an approximative numeric solution to Navier-Stokes equations for fluid dynamics (2.1), (2.2). It was developed in 1977 by Gingold and Monaghan [GM77] and independently by Lucy [Luc77]. Initially it was designed for use in astrophysics but later it found its way into many other fields of research, such as ballistics, oceanography and, more recently, computer animation.

It represents the fluid with a set of independent particles and thus it ranks among the methods of Langrangian viewpoint. The particles have a *smoothing radius*, a distance over which their quantities are *smoothed* by a kernel function. Put in another way, the particle's attributes can be calculated only with the use of all the neighboring particles within the distance defined by smoothing radius. The contributions of each particle are weighted by their distance and their density, the weights are given by the

[1] Level-Set Method is a numerical algorithm for approximating the dynamics of moving curves and surfaces [Phi99]

kernel function we use. There is a wide variety of kernel functions to use, including Gaussian function and the cubic spline. A detailed description of the SPH method can be found, e.g., in [Mon92] or [DG96].

Solenthaler et al. [SSP07] propose a unified particle model. They use SPH particles for both solid and fluid materials and distinguish between these two types only by changing the attribute values of the particles. The proposed model is then used to simulate a variety of fluid-solid interaction processes, such as melting and solidification.

Adams et al. [APKG07] proposed an adaptive sampling algorithm for particle-based fluid simulation. They introduce a sampling condition allowing the change of the density of the particles so that they focus the computations in the complex regions and reduce the number of particles inside the fluid or near flat surfaces.

SPH method is often used in fluid simulation ([KW06], [MCG03]) but Krištof and Beneš in *Hydraulic Erosion Using Smoothed Particle Hydrodynamics* [KBKS09] were the first ones to try to couple SPH with erosion. They represent the water flow with SPH particles that erode the terrain and transport the sediment. They define a donor-acceptor scheme that describes the advection between SPH particles and with it they simulate the diffusion of the sediment and settling in the direction of the gravitation. For the terrain they use a height field data structure which limits the use of this algorithm to 2.5D terrains. Boundary particles are used as a means of communication between the terrain and the particles. First the SPH particles interact with the boundary particles and exchange the sediment and after that the terrain height is adjusted according to the change of sediment in the boundary particles.

Their solution effectively couples SPH with erosion. The resulting method works only on 2.5D terrains but the fluid simulation is fully 3D, allowing such formations as breaking waves or splashes.

3.3 Semi-Lagrangian Approach

Both the Eulerian and Lagrangian approaches have their strengths and weaknesses. Hybrid Semi-Lagrangian methods try to combine the two approaches to reduce their individual drawbacks.

Foster and Fedkiw used this combined approach in [FF01] for modeling and animating of liquids. They simulate the fluid with particles but they use an implicit surface called a level set to track the motion. The fluid system they designed is capable of interacting with graphics primitives such as parametric curves and moving polygons. Fedkiw et al. [FSJ01] use a similar approach to create a visual simulation of smoke.

Andrysco et al. [ABB08] use their modified Semi-Lagrangian approach to simulate the interaction of fluids with permeable materials. In their work they propose equations which allow the simulation of permeable, porous and absorbent materials.

4 Data Structures for Erosion-Based Terrain Modeling

In computer graphics, terrains are usually represented as triangle meshes but this data structure can cause many problems in erosion simulations. This chapter will describe the terrain representations which are the most common in erosion-based terrain modeling.

4.1 Height Map

Height map is the most common data structure used for terrain modeling. It can be described as a two-dimensional array, in which each element has its own properties. Every element carries information about the height at the element but it can contain many other data describing the state or the characteristics of the material. The most important simplification of the structure is that it considers the whole column to be made of the same material with the same properties.

This data structure is very often used in simulations that do not require great precision but need to work interactively. The simplicity of the data structure allows very fast and efficient solutions and its memory requirements are significantly better than those of the remaining data structures used in terrain modeling. Considering that each element stores n bytes of data and we are representing the terrain as a grid of 1024 x 1024 elements, we will need only n MB to store the terrain [BF01]. The main disadvantage of height fields is that their usage is limited to simulation of 2.5D effects as the data structure does not support concave structures such as caves or overhangs.

This data structure is used, e. g., in [MKM89], [ON00] and [BR04].

4.1.1 Layered Height Map

Beneš and Forsbach in *Layered Data Representation for Visual Simulation of Terrain Erosion* [BF01] suggested an improvement of the basic height map data structure. They took inspiration in real geological measurements and extended the height map by adding layers. In nature, the terrain usually consist of several layers of materials with different characteristics (see Figure 4.1). The authors integrate this idea by changing the height map structure - each element is now consisting of a one-dimensional array of elements which contain the same information as was stored in the height map (height of the layer and the information about the material). They take advantage of the fact that the material layers are usually very thick so we can measure the height of the layer and store all the information at once instead of describing every voxel and thus saving a lot of data. The memory requirements are similar to the ones of the height field data structure just with the difference that for each element in the 2D array we now store k elements representing k layers. To store the terrain with 1024 x 1024 elements $k \cdot n$ MB will be needed.

This data structure is used, e. g., in [BF02] and [MDH07].

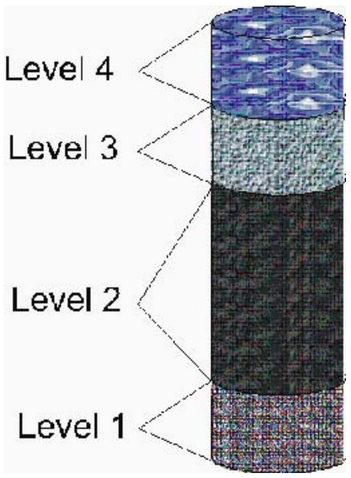


Figure 4.1: Typical structure obtained by geological core sample. [BF01]

4.2 Volume Grid

Volume grids are together with the height maps the most commonly used data structures in terrain modeling. The volume representation divides the volume of the scene into voxels, 3D cubes of the same size. Each voxel then contains the same information as the elements in the previous types of data structures only with the exception of height parameter which is now pointless as the position of the voxel is given by its coordinates in the 3D array and the size of the voxel is identical for the whole grid.

The techniques using voxel grids have the highest precision and give the best results. Voxel grids are capable of describing any 3D structures and so they are giving us the means of modeling features that cannot be modeled by the previous approaches. The main drawback of this data structure is that it has huge memory requirements. To model a scene with resolution 1024 x 1024 x 1024 using a voxel technique we will need n GB of data ([BF01]). The methods which work with voxel grids give precise results but are not capable of running with a real-time response so they cannot be used if an interactive application is desired.

This data structure is used, e. g., in [JFBB10] and [DEJ$^+$99].

4.3 Triangle Mesh

Another data structure which can be used for terrain modeling is a triangle mesh. It represents an object as a set of vertices connected by edges. Three vertices connected by edges represent a triangle face, the set of all faces represents the surface of the object. One of the advantages of this structure compared to the previous methods is that it does not have to be transformed in order to render it as this data structure is very often used in computer graphics and the rendering pipeline is optimized to work with it. Another advantage of the triangle mesh is that its resolution can change throughout the scene, allowing the creation of a very detailed scene in important areas while the flat unimportant regions are modeled with lower resolution.

But its use for simulation of erosion has many disadvantages as well. When eroding an object, its surface changes. In the previous methods this could be simulated just by changing the height of the element of a height map or by changing the properties of a voxel. When working with triangle meshes, we have to actually move the vertices to simulate the surface change. The triangle mesh can only store the information about the neighborhood of the given vertex but it does not provide any information about the relation of the given vertex and the rest of the mesh. By changing the position of a vertex an inconsistency can be created as the vertex penetrates the mesh. The problem of fixing mesh geometry is well-known but a general solution to this problem is to the best of my knowledge not known.

Purchart in [Pur09] uses an adaptive triangle mesh to simulate a deformable sandy terrain but the solution is designed to work only with the 2.5D terrains, formations such as caves or overhangs are forbidden.

4.4 Tetrahedral Mesh

Tetrahedral mesh is a volume variation of the triangle mesh data structure. The model is not represented only by its surface, the whole volume is stored as a set of tetrahedra. It looses the rendering advantage of the triangle mesh as we have to extract the surface in order to render the tetrahedronized model. But in some applications the volume approach can be more desirable than the surface approach of the triangle meshes as it can give us additional information about the topology of the object.

Tetrahedron data structure is used in [TJ10] for weathering and erosion of 3D terrains. Their mesh is based on Delaunay deformable models (DDM). In each iteration of the algorithm, the vertices of the mesh are moved to their new location to simulate erosion and a new Delaunay triangulation (DT) is constructed. By moving the vertices and building the new triangulation the information about the material in each of the vertices is lost, to restore it, a backward advection scheme is used: for each cell of the DT the mean offset of its vertices is found and applied to the circumcenter of the cell, the material that occupied this location in the previous frame is assigned to the cell. The authors demonstrate a use of the data structure in weathering and erosion simulation whose objective is a creation of visually plausible scene for the use in computer animations. The main drawback of their approach is that it is not capable of running interactively as the creation of the DT in each step of the algorithm is very computationally expensive.

5 Our Solution

In this chapter the methods and structures used in the proposed solution will be described. At first, our approach to hydraulic erosion will be described, followed by the description of the SPH library and the data structures used in the implementation.

5.1 Hydraulic Erosion

Terrain modeling has been an important part of computer graphics for many years now but many problems still remain unsolved. At present there are many solutions available. Most of the solutions try to achieve realistic looks of the terrain by imitating the influence of natural erosion processes. Some of the solutions are only useful in very specific cases as they are designed to generate very specific scenes or formations.

Hydraulic erosion has the greatest influence on the terrain appearance so most of the researchers are dealing with it in their simulations. Most of the available hydraulic erosion solutions can be classified into one of the following groups: slow, precise algorithms or fast, approximate ones. The algorithms ranking in the first group usually solve the Navier-Stokes equations using the Euler approach which results in precise solutions, however, these solutions are usually very memory consuming and the simulations are not capable of running with a real-time response. The other group incorporates methods which do not try to simulate the erosion processes as accurately as possible but try to produce realistic-looking terrains while simplifying the calculations to achieve simulations with real-time response. These solutions usually work only with 2.5D terrains and water effects, some of them are using a Lagrangian approach to simulate a realistic behavior of fluids including the 3D effects such as breaking waves or splashes.

The only solutions capable of working with both fully 3D terrains and fully 3D water are based on the Euler viewpoint. These solutions give accurate results but they are very computationally expensive and very memory consuming. We propose a solution to hydraulic erosion which uses a Lagrangian approach to solving fluid dynamics equations. Our solution supports fully 3D water effects and requires less memory than the Euler based solutions.

With similar intentions (3D effects and low memory consumption) we decided to represent the terrain with a triangle mesh, allowing us to increase the resolution in detailed regions of the scene and decrease it in flat homogeneous ones. The solution we propose is capable of reproducing a fully 3D scene, with both 3D water and terrain effects, while reducing the memory requirements.

5.2 SPH

As mentioned before, we decided to use a Lagrangian approach to solving fluid dynamics equations. The advantage of this approach is that it uses particles to describe the fluid volume and thus limits the computations only to the locations where the fluid is present. SPH method is often used in fluid simulations but Krištof et al. in [KBKS09] were the first ones to try to use it in erosion simulation. We took inspiration in their work and decided to represent the fluid in a similar way using the SPH method. The erosion is calculated when the SPH particles collide with the terrain model, the eroded sediment is then associated with the particles.

The SPH method is well-known in the field of fluid simulation and there are libraries available. We decided to use an existing implementation of SPH instead of implementing it. There are several complex open source libraries dedicated to fluid modeling and simulation such as *OpenFLUID* (available from `http://www.umr-lisah.fr/openfluid/`), *OpenFoam* (http://www.openfoam.com/), *SPHear* (`http://sphear.sourceforge.net/`) or *GADGET* (`http://www.mpa-garching.mpg.de/gadget/`). These libraries are very complex and offer much stronger tools than what we need in our simulation at the moment. We chose to implement our solution using C++, so we had to discard a lot of libraries due to the programming language they use, e.g., *Spheral++* (`http://sourceforge.net/projects/spheral/`) is using Python, *CFD Fluid Collection* (`http://cfdcollection.sourceforge.net/`) is written in C#.

In the end we had narrowed our choice to two libraries, both written in C++ and simple enough for our needs. The first one was *GPU SPH Simulation* (`http://code.google.com/p/gpusphsim/`). This SPH implementation was developed as a part of a master thesis with no further information if the author is still improving it and it was possible that it could contain errors that the author did not discover during the testing phase. So we decided to use the other library *Fluids v.2* (`http://www.rchoetzlein.com/eng/graphics/`

`fluids.htm`) in our implementation. This library is also propagated with *bullet* (`http://code.google.com/p/bullet/`), a free 3D game multiphysics library. One of its advantages is that it has both the CPU and GPU version so if we decide to implement our solution on GPU in the future, we will not have to replace the library or implement it ourselves, we will just use the other implementation of the SPH method. An example of a scene generated using the unmodified library *Fluids v.2* is shown in Figure 5.1.

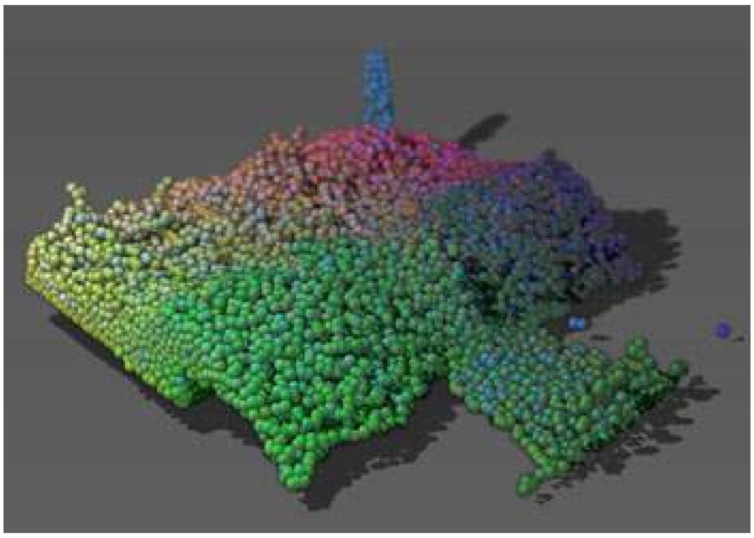


Figure 5.1: An example of SPH fluid simulation. [Hoe09]

5.3 Data Structure

As we want to model a fully 3D terrain, we cannot use the height map data structure. The voxel grids are often used to model fully 3D terrains but they are very memory consuming. We wanted our solution to have adaptive resolution based on the level of detail of the modeled scene, so we were choosing between triangle and tetrahedral meshes. We are only simulating surface erosion processes and for that we do not need any information about the inner parts of the modeled objects. For that reason we have decided to use the surface triangle meshes as they are simpler to implement and maintain and their use should result in faster simulation.

The advantage of the triangle mesh data structure is that its resolution can be adaptively changed according to the characteristics of the scene. The mesh can be more detailed in important regions. We can take advantage of this in erosion simulation - if we can estimate in which region the erosion will be occurring, we can prepare more detailed mesh in that locations and get more detailed erosion simulation, while the rest of the scene can remain unchanged.

The implementation can work with 3D models stored in .obj format. The source code managing the import was taken from *OpenGL OBJ Viewer* (`http://www.dhpoware.com/demos/glObjViewer.html`) and adjusted for our needs.

The 3D models used in the simulation cannot contain holes and have to be *closed*. Closed model is such a model which does not have any faces that could be accessed from both sides. In other words, the model has to be "waterproof"; if we would fill the model with water, the water cannot be leaking from it anywhere. This condition is important when working with terrains as terrains are often modeled only by the upper surface and not the whole volume (see Figure 5.2). The required way of modeling the terrain is shown in Figure 5.3.

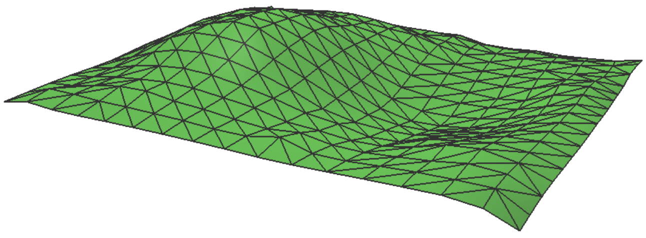


Figure 5.2: An example of a simple terrain modeled only by its surface.

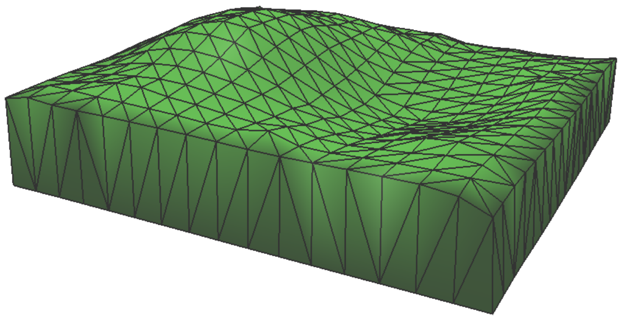


Figure 5.3: An example of a simple terrain modeled as a closed model.

5.4 Auxiliary Data Structures

Neither the SPH particles nor the triangle mesh data structure give us an implicit information about the spatial division of the elements in the scene. The particles are interacting only with other particles that are within its radius, the same applies for the fluid-solid collision. Without any auxiliary data structure, all of the particles or faces of the triangle mesh would have to be searched to find all the elements necessary for the given calculation. For that reason, two separate auxiliary data structures are used to speed up the simulation, one stores the spatial division of the particles and the other the spatial division of the faces of the mesh.

Both the SPH particles and the faces of the triangle mesh can change their position during the simulation, because of that we have to reinitiate the structures every iteration of the algorithm. After the initialization of the structures, we only have to search a small number of particles or faces to calculate the fluid forces or fluid-solid collisions. With the use of the auxiliary data structures, the algorithm can speed up significantly.

5.4.1 Particles

The *Fluids v.2* uses a uniform grid spatial division to speed up the calculations of the inter-particle forces. The size of a cell is given as double the smoothing radius of a particle. The grid resolution is then calculated according to the size of the whole scene.

Each particle is located in exactly one cell. Because of that it is not necessary to keep a list of all particles that belong to the cell in the cell itself. Instead of that it is possible to store only one particle index in the cell. The particle on the indexed position then has a pointer to the next particle within the grid cell and so on. The advantage of this approach lies in the fact that we do not have to know the number of the particles that will end up in each cell of the grid in advance.

When I was working with the library I discovered a bug in this part of the code. In the original code the grid was by mistake created with a wrong parameter, using the grid size instead of the grid resolution. Because of that, the grid was created with much greater resolution than what was needed, which was especially obvious when working with large scenes. After

I replaced the parameter, the grid was created with correct resolution and the algorithm sped up significantly.

5.4.2 Triangle Mesh

A similar data structure as the one described in Section 5.4.1 is used for storing the spatial information about the faces of the triangle mesh. A uniform grid is created with the same resolution as the particle grid, the size of a cell is twice the smoothing radius of a particle (see Figure 5.4).

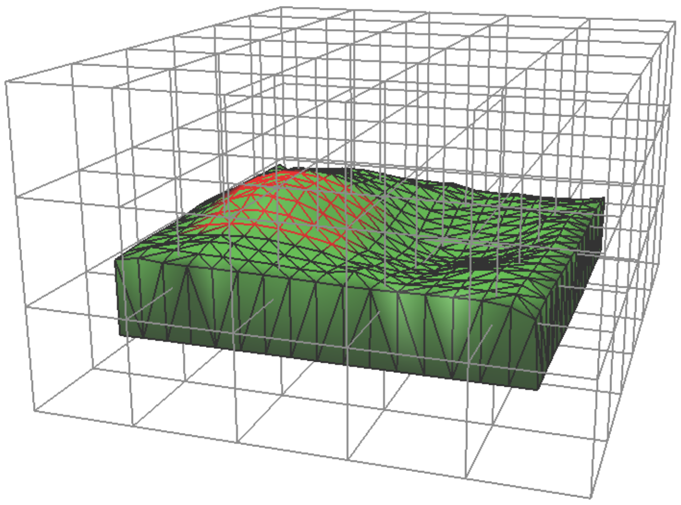


Figure 5.4: The grid storing the spatial information about the mesh.

Unlike the particles, generally a face of the mesh can belong to more than one cell, in that case its index has to be stored in all of the cells where the triangle is partly present. To decide in which cells to store the face, the bounding box of the face is found and its coordinates are used to calculate the minimum and maximum indices in the grid. The face is then stored in all the cells with indices between these values. An example of a grid cell is shown in Figure 5.5, faces that belong to the cell are highlighted in red color.

To calculate fluid-solid interactions, all the faces within the particle radius have to be localized. For a particle located near a cell boundary, the particle would also interact with faces from adjacent cells, not only from the cell the particle is in. If all the cells within the reach of the particle would be searched for collisions, the contribution of one face would be calculated more

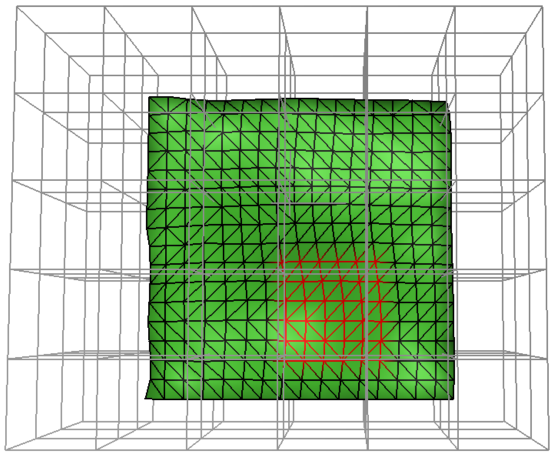


Figure 5.5: An example of spatial division, the red triangles belong to the same cell. Triangles belong to the cell if they are located inside it.

times. This could happen if the face was stored in more than one of the searched cells, leading to unexpected behavior of the particles. To eliminate this problem, I extend the bounding box of the face by the radius of the particle. Each cell of the grid then contains all the faces that are located inside it and in addition all the faces that are within a particle reach from it. The same scene as in Figure 5.5 is shown in Figure 5.6, this time the triangles that are not located inside the cell but are within the particle radius are included as well.

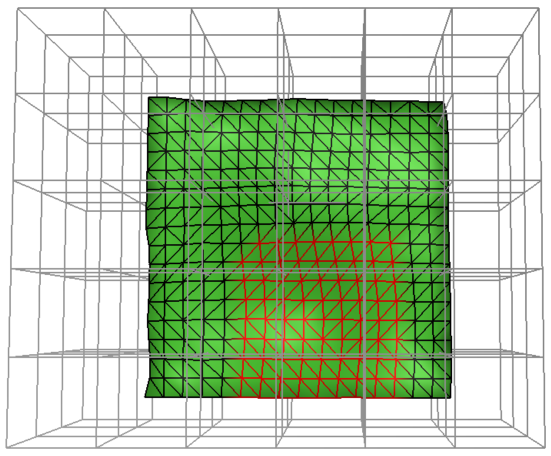


Figure 5.6: An example of spatial division, the red triangles belong to the cell. Triangles belong to the cell if they lie within the specified distance.

6 Proposed Algorithm

The proposed algorithm consists of the following steps:

1. Fluid particles appear in the scene.
2. The fluid forces are calculated.
3. The particles are moved and the fluid-terrain interactions are calculated.
4. The erosion/deposition sediment exchange between the particles and the terrain is calculated.
5. The mesh is updated according to the amount of sediment in the vertices of the mesh.

In the following text each of the steps will be described in more detail.

6.1 Water Sources

At the beginning of each iteration, new particles can appear in the scene, simulating water sources. The user can influence the position where new particles will appear. He can as well set a rate how often should the particles be generated.

One of the parameters of the simulation is the maximum number of particles that are allowed in the scene. If new particles are being generated every iteration, at some point this maximum will be reached. Some particles have to be deleted to make space for new ones. The particles to delete are chosen according to the *age* parameter, that way the oldest particles are removed from the scene as they have probably already moved to less important parts of the scene.

6.2 Fluid Forces

The library *Fluids v.2* is handling the calculations of the inter-particle forces. The calculations take advantage of the data structure introduced in Section 5.4.1. Without this structure, all particles would have to be searched in order to find all the particles within the reach of a given particle. Using this structure, maximum of eight cells have to be searched - in case that the particle is located in the corner of the cell, the adjacent cells located within the particle radius have to be searched as well.

6.3 Fluid-Terrain Interaction

The motion of the particles is calculated based on the fluid forces from the previous step. The original library used a simple bounding box to limit the region where the particles are allowed. In the proposed solution the particles need to interact with the triangle mesh representing the terrain, so I had to modify the code for the particles to correctly interact with the triangle faces.

The motion of the particle is influenced by external forces - by gravity and by all the faces within the reach of its radius. I use the data structure introduced in Section 5.4.2 so that I would not have to go through all the faces of the mesh to find the faces influencing the particle. The particle position is transformed into grid coordinates and then I only have to check for interactions with the faces stored in the grid cell where the particle is located.

For each face f in the cell the interaction with particle p is checked in the following manner:

- At first, a line going through the particle position is constructed with the direction of a normal vector of the face f. This line is then checked for intersections with the face f.
- If the line intersects the face f, the distance between the face f and the particle p is calculated.
- If the distance between the face f and the particle p is smaller than the particle radius, the face f will influence motion of the particle p.

The contribution of the face is then calculated using a penalty-force method [Ama06]:

$$f = k^s d\vec{n} + k^d(\vec{v} \cdot \vec{n})\vec{n}, \tag{6.1}$$

where k^s is the penalty force stiffness, k^d is the damping coefficient, d is the penetrated distance, $\vec{v}$ is the velocity of the particle and $\vec{n}$ is the normal vector of the face.

6.4 Erosion and Deposition

When the fluid-solid interaction occurs, the particle erodes the surface or deposits the sediment on it. Erosion is calculated using the method introduced in [WCMT07]. Erosion is caused by the fluid flowing parallel to the solid boundary. The force applied to the solid boundary is called the shear stress. To apply this force to a solid, Wojtan et al. give the solid object non-Newtonian characteristics via a power-law model:

$$\tau = K\theta^m, \tag{6.2}$$

where τ is the shear stress, $K = 1$ is a constant, θ is the shear rate and m is the power-law index, a constant determined by the material. If enough shear stress is applied to the solid, it will deform like a fluid, so the solid is treated as a *shear-thinning* fluid. A typical value for shear-thinning fluids is $m = 0.5$. The shear rate θ can be approximated from the fluid velocity relative to the surface:

$$\theta = \frac{\vec{v}_{rel}}{l}, \tag{6.3}$$

where $\vec{v}_{rel}$ is the relative fluid velocity and l is the distance between the particle and the face in the triangle mesh.

The erosion rate can be calculated using the equation by Partheniades [Par65].

$$\varepsilon = k(\tau - \tau_c)^a, \tag{6.4}$$

where ε is the erosion rate, k is the erosion constant, τ is the shear stress, τ_c is the critical shear stress and a is a constant, which is often set to be 1 as we can assign any value to the constant k. The critical shear stress is a threshold value that needs to be overcome for erosion to take place. For higher values the object will keep its shape more easily, for lower values even slow fluids will cause an erosion of the surface. This principle could be used in the future to integrate different materials of the model, assigning different values of critical shear stress to different parts of the scene. The deposition occurs when the particle gets close enough to the terrain but the shear stress is not strong enough to cause erosion. In that case part of the sediment carried by the particle is deposited on the terrain.

The calculated erosion or deposition rate is uniformly divided among the three vertices of the triangle. Each vertex has an associated parameter storing the amount of the sediment exchanged during the current iteration of the algorithm. This parameter was introduced to speed up the simulation - we sum all the contributions from all the particles in it and only update the positions of the vertices once at the end of the iteration.

6.5 Mesh Subdivision

The erosion or deposition process affects the face located within the radius of the particle and all its neighbors. If the resolution of the mesh representing the terrain is not high enough, the erosion or deposition can affect even the regions of the mesh where the fluid is not present. To reduce this problem, the mesh resolution can be adaptively increased. The distances between the triangle vertices and the particle are calculated in order to decide whether the triangle needs to be divided.

The triangle division follows this algorithm: if at least one of the distances between the vertices and the particle is greater than a specified maximum distance d, tesselate the triangle and locate the new triangle which is closest to the particle. Compute the new distances between the vertices and the particle and repeat this process until all the distances are shorter than the limit distance d. When the tesselation is finished, the erosion or deposition is applied to the vertices of the new triangle, all the vertices are now within the distance d from the particle.

The way the triangles are tesselated has a big impact on the resulting scene. I have implemented and tested several possible tesselation schemes to find out which one is giving the best results. I will label each of the schemes as *M-N scheme*, where M defines how many triangles will be created from the original one and N defines into how many triangles the neighbors will be divided.

6.5.1 3-0 Tesselation Scheme

The 3-0 tesselation scheme adds one point inside the triangle, creating three new triangles. The inserted point can be, e.g., the point closest to the particle or the center of the triangle. The neighboring triangles are not affected. Figure 6.1 shows the results of the tesselation scheme applied to one triangle of the mesh. The inserted points are highlighted with a small dot.

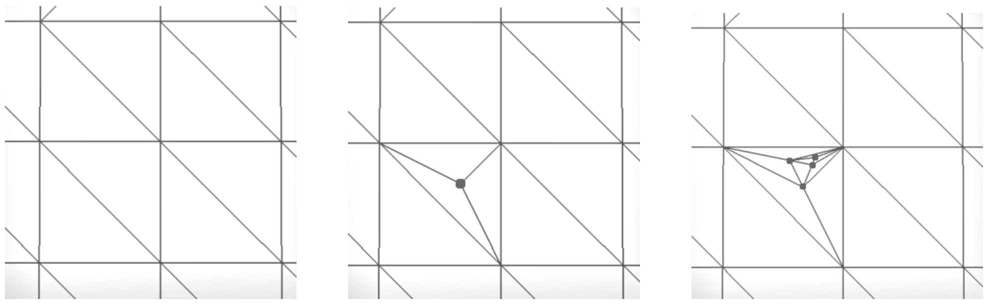

(a) Original mesh. (b) Simple tesselation. (c) Recursive tesselation.

Figure 6.1: 3-0 tesselation scheme.

This scheme tends to create long narrow triangles, which are exactly the types of triangles we are trying to avoid. The scene created with this scheme is depicted in Figure 6.2. This scheme is unsuitable for our purposes.

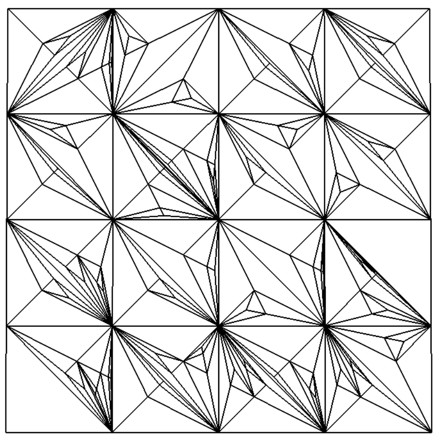


Figure 6.2: An example of a scene created with 3-0 tesselation scheme.

6.5.2 4-2 Tesselation Scheme

The 4-2 tesselation scheme adds three new points in the middle of the edges of the triangle. Four new triangles are then created by connecting these three new vertices with edges. The neighboring triangles have to be tesselated as well to maintain the mesh consistency, the tesselation is done by simply connecting the new vertex in the middle of the edge with the opposite vertex. Figure 6.3 shows the results of the tesselation scheme applied to one triangle of the mesh. The inserted points are highlighted with a small dot.

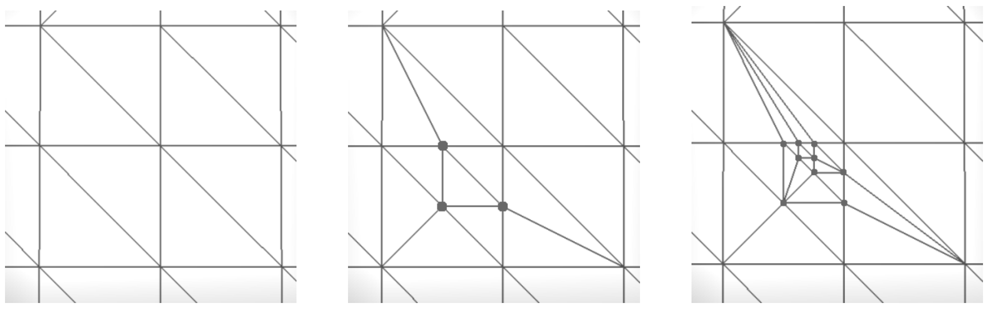
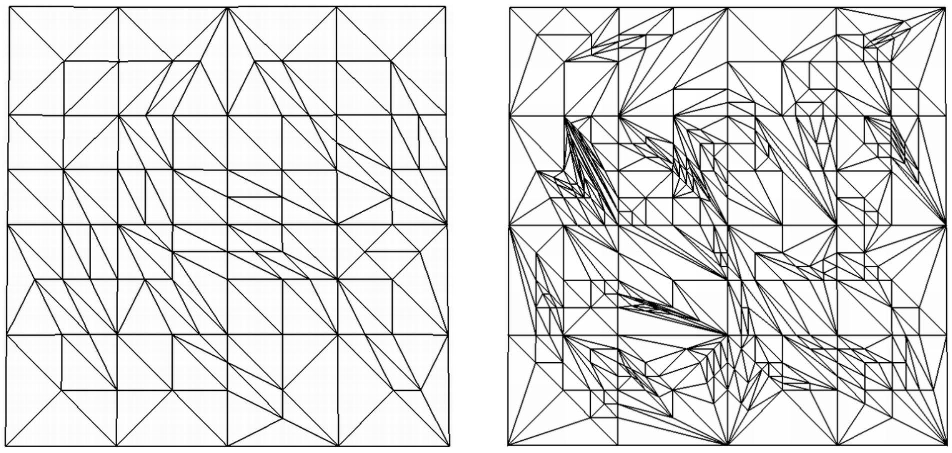

(a) Original mesh. (b) Simple tesselation. (c) Recursive tesselation.

Figure 6.3: 4-2 tesselation scheme.

This scheme creates triangles that are more even that the ones created with the 3-0 scheme. The looks of the resulting scene will differ greatly according to the value of the maximum distance parameter d (already mentioned in Section 6.5). For d approximately half the size of the triangle edges the scheme results in the creation of an acceptable triangle mesh (see Figure 6.4a). For smaller d the generated triangles become thinner and less acceptable (see Figure 6.4b). Nevertheless, this scheme gives the best results from all the tested tesselation schemes and it is used in the implemented solution. The user has to pay attention to the parameter d and choose its value reasonably according to the level of detail of the input mesh.

6.5.3 4-4 Tesselation Scheme

The 4-4 tesselation scheme divides the input triangle in the same manner as the 4-2 scheme. The difference between these two lies in the way how they handle the neighbor tesselation. The 4-2 scheme cuts the neighboring triangle in half, which may lead to long triangles. The idea was to add another point

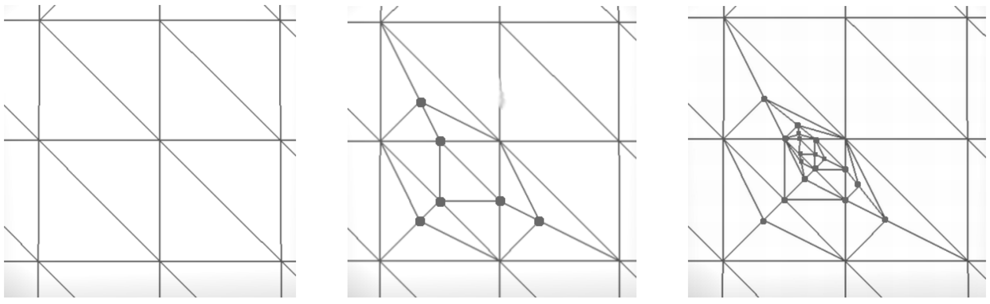


(a) Tesselation for d approx. half the size of the triangle.

(b) Tesselation for smaller d.

Figure 6.4: An example of a scene created with 4-2 tesselation scheme.

inside the neighboring triangle and thus divide it into four new ones. The results of this tesselation scheme when applied to one triangle of the mesh can be seen in Figure 6.5. The inserted points are highlighted with a small dot.



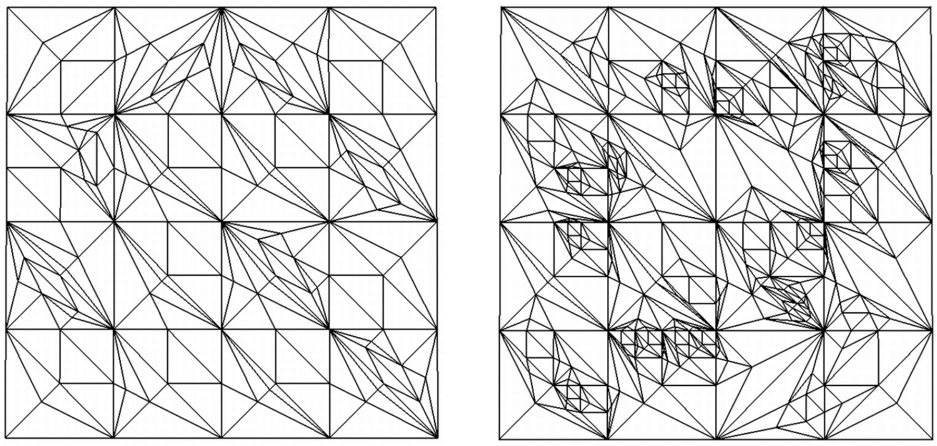
(a) Original mesh.

(b) Simple tesselation.

(c) Recursive tesselation.

Figure 6.5: 4-4 tesselation scheme.

Contrary to my assumption that this alteration will lead to better results, the tesselation scheme produces more uneven results even for larger values of the parameter d. I tested several possible positions for the point inserted inside the neighboring triangle such as the center of the triangle or the middle of the line segment connecting the point in the middle of the edge with the opposite vertex. I also tried to include a random factor to the position but none of these adjustments did cause a significant improvement in the resulting scene. Figure 6.6a captures the results for d approximately half of the triangle edges, the results for smaller d are shown in Figure 6.6b.


(a) Tesselation for d approx. half the size of the triangle.

(b) Tesselation for smaller d.

Figure 6.6: An example of a scene created with 4-4 tesselation scheme.

6.6 Mesh Modification

The erosion calculation consists of two separate steps. At first the interaction between the SPH particles and the triangle mesh is calculated following the algorithm described in Section 6.4, the amount of eroded or deposited sediment is stored for each vertex. In the second step the mesh is modified in each of the vertices that were influenced by the first step. The vertex is moved in the direction of the normal vector according to the amount of the associated sediment.

The vertex will be moved in the direction of the normal vector but it is not obvious how far should the vertex be moved. The vertex belongs to several triangles. In general these triangles can differ greatly in size and shape leading to various volume changes for the same vertex displacement. The situation is illustrated in Figure 6.7. The dashed line represents the original mesh, the dotted line represents the vertex displacement in the direction of the normal vector and the full line shows the modified mesh after the vertex relocation. The triangles of the original mesh in Figure 6.7a have smaller area than the triangles in Figure 6.7b, resulting in a smaller volume change for the same translation of the vertex.

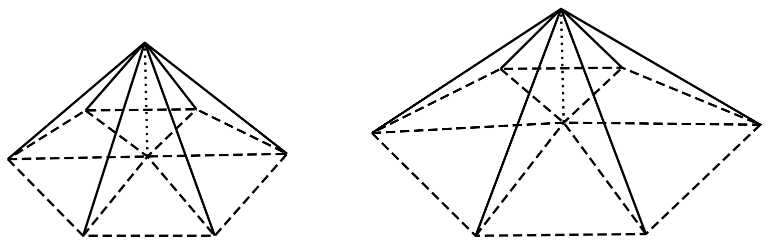

(a) Scene consisting of faces with smaller area. (b) Scene consisting of faces with larger area.

Figure 6.7: An example of volume changes for the same value of vertex displacement.

We need to control the volume change regardless of the local topology of the scene. In order to do that we need means to calculate the vertex displacement so that the volume change would have a unit size. The total volume change is a sum of contributions from all the affected triangles. The volume change for one triangle equals to the volume of the region bordered by the original and the new triangle, this area always forms a tetrahedron (see Figure 6.8).

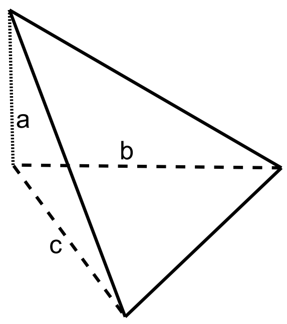


Figure 6.8: The volume change for one face has a form of a tetrahedron.

To derive the required equation for vertex displacement we have to start from the formula for the volume of a tetrahedron [Wik12]:

$$V = \frac{|(A - D) \cdot ((B - D) \times (C - D))|}{6}, \tag{6.5}$$

where A, B, C and D are the vertices of the tetrahedron. If the tetrahedron is translated so that the vertex D coincides with the origin of the coordinate system, the formula simplifies to:

$$V = \frac{|a \cdot (b \times c)|}{6}, \tag{6.6}$$

where $a = (a_1, a_2, a_3), b = (b_1, b_2, b_3), c = (c_1, c_2, c_3)$ represent three edges that meet at the vertex D (see Figure 6.8) and $a \cdot (b \times c)$ is a scalar triple product. The scalar triple product can be calculated using the determinant:

$$6V = |a \cdot (b \times c)| = \begin{vmatrix} a & b & c \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \tag{6.7}$$

By solving the determinant we obtain the following relation:

$$6V = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2 - a_1b_3c_2 - a_2b_1c_3 - a_3b_2c_1. \tag{6.8}$$

Using it we can calculate the volume as a sum of tetrahedron volumes:

$$V_{total} = \sum_{0}^{n} V_i, \tag{6.9}$$

where n is the number of triangles containing the given displaced vertex and V_i are the volumes of individual tetrahedra. The V_{total} is the total volume change for the the vertex displaced $|a|$ units in the direction of the vertex normal. If the vertex is displaced only $\frac{|a|}{x}$ units instead, the volume change for one tetrahedron will be:

$$\begin{aligned} 6V_1 = \begin{vmatrix} \frac{a_1}{x} & b_1 & c_1 \\ \frac{a_2}{x} & b_2 & c_2 \\ \frac{a_3}{x} & b_3 & c_3 \end{vmatrix} &= \frac{a_1b_2c_3}{x} + \frac{a_2b_3c_1}{x} + \frac{a_3b_1c_2}{x} - \frac{a_1b_3c_2}{x} - \frac{a_2b_1c_3}{x} - \frac{a_3b_2c_1}{x} \\ &= \frac{a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2 - a_1b_3c_2 - a_2b_1c_3 - a_3b_2c_1}{x} \end{aligned} \tag{6.10}$$

By comparison of equations 6.8 and 6.10 it is obvious that $V_1 = \frac{V}{x}$. A volume change of a unit size will be obtained for $x = V$. Using these ideas the solution to the given problem is very simple. The vertex is moved one unit in the direction of the normal vector and the volume change V is calculated. The new displacement distance is then obtained as $\frac{1}{V}$, this displacement will result in a volume change of unit size. The final displacement is calculated by multiplying it with the amount of sediment carried by the particle.

6.7 Mesh Consistency

While relocating the vertices of the mesh to simulate the erosion process, an inconsistency can be created in the mesh. An example of an inconsistency is captured in Figure 6.9. The relocated vertex cut through another face of the mesh, resulting in intersection of the faces from different parts of the mesh. The dashed line represent the boundary where the two regions intersect.

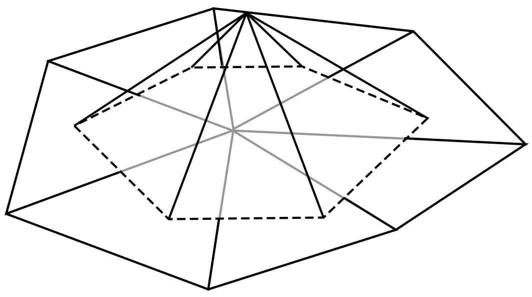


Figure 6.9: An example of a mesh inconsistency.

6.7.1 Inconsistency Detection

For a simple scene the inconsistency can be detected easily. If an inconsistency was created, the vertex had to move through one of the faces during the mesh modification step. That means that the line connecting the initial and the final position of the vertex has to intersect with one of the faces of the mesh.

For a more complex scene with concave features the detection algorithm is more complicated. The inconsistency can be created even if the vertex trajectory did not intersect with any faces. Figure 6.10a shows a valid scene with concave features, the point marks the vertex where the erosion will be applied. The resulting scene after erosion is shown in Figure 6.10b. The vertex trajectory did not intersect any faces but the inconsistency was created. To detect this type of inconsistency, instead of checking the vertex trajectory, we have to check for intersections with all the faces to which the relocated vertex belongs. In this thesis I work only with the type of inconsistencies that can be created in a simple scene.

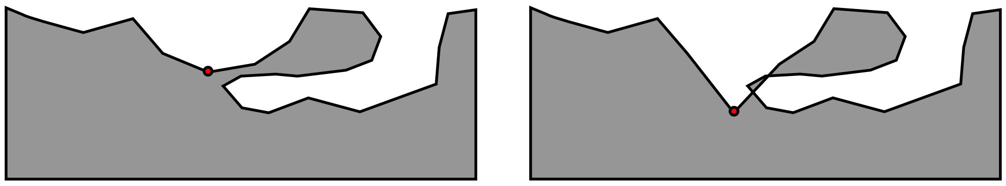

(a) Original scene. (b) Scene with an inconsistency.

Figure 6.10: A simple 2D example of an inconsistency in the scene with concave features .

6.7.2 Finding the Boundary of the Inconsistency

When the inconsistency is detected, we have to eliminate it and fix the mesh. In order to do it, first we have to find the boundary of the inconsistency - the polygonal chain where two parts of the mesh intersect. An example of the inconsistency boundary is represented by the dashed line in Figure 6.9.

Two planes which are not parallel meet in a single line l. If triangles lying in these planes intersect, their intersection is a part of the line l. The four possible ways of intersection of the two triangles are presented in Figure 6.11. The parameters t_1 and t_2 determine the points of intersection of the line l and the edges of the triangle T_1, the parameters t_3 and t_4 determine the points of intersection of the line l and the edges of the triangle T_2.

During the inconsistency detection phase I have found two intersecting faces which are a part of the inconsistency. The line segment where they intersect forms the first segment in the polygonal chain representing the boundary of the inconsistency. To find the next segment we have to inspect the possible ways of triangle intersection (Figure 6.11). The parameter t_1 or t_3 in figures represents the entry point of the line segment and t_2 or t_4 represents the exit point; the intersection line segment is always the segment between the second and the third parameter on the line as this is the region where both

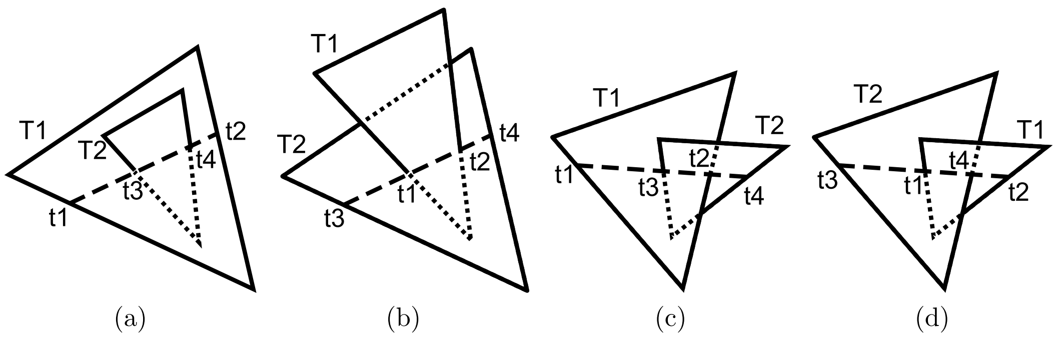


Figure 6.11: Four ways two triangles can intersect.

triangles intersect the line l. If the line segment ends with a point defined by the parameter t_2, it means that the processing of the triangle T_1 was finished and we need to proceed to its neighbor through the edge containing the point defined by t_2. Similarly, if the segment ends with a point defined by t_4, the processing of the triangle T_2 is finished and we proceed to its neighbor. Following this simple algorithm we can find the boundary of the inconsistency. For each point in the boundary we also store the indices of the triangles whose intersection formed the given line segment, we will use this information later for fixing the mesh.

The situation is more complicated when the line of intersection l contains an edge or a vertex of one of the triangles. If the line l contains an edge of one of the triangles, it is not clear to which neighbor we should move when the triangle is processed. A simplified example of the boundary is shown in Figure 6.12 - only one side of the intersected mesh is shown. Let us assume that the correct boundary is marked with the red dotted line. The line segment between points 1 and 2 (segment 12) can be found using the algorithm described above. For segment 23 we encounter a problem - the segment ends in a vertex of the triangle F and we do not know in which of the neighbors the boundary continues. Or, as in this case, the boundary continues through a triangle which is not a neighbor of the current triangle at all. For singular cases such as this, we have to store all the possibilities and find out where the boundary goes on. For segment 23 the boundary can continue either in triangle E or G. By testing these triangles we find out that they only meet with the boundary at one point, which is already included in it. The boundary can continue through the edge containing the intersection point, so we have other triangles for the list of possible intersections with the boundary - through E we continue to B and through G to D. The triangle

D meets with the boundary in segment 34 and so on. The searching ends when we arrive back to the first point of the inconsistency boundary.

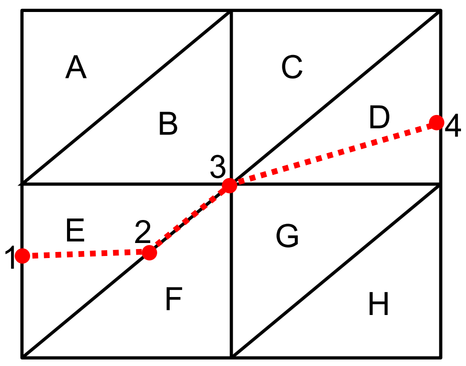


Figure 6.12: An example of the boundary of the inconsistency.

6.7.3 Fixing the Inconsistency

After we find the boundary of the inconsistency, we have to fix the mesh to restore it to the consistent state. An example of the boundary is shown in Figure 6.13, the red dashed line represents the boundary and the red dots mark the points where the boundary leaves one triangle and enters another.

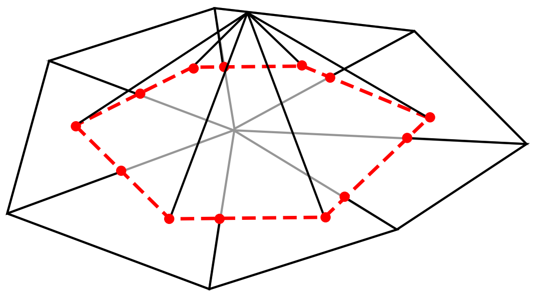
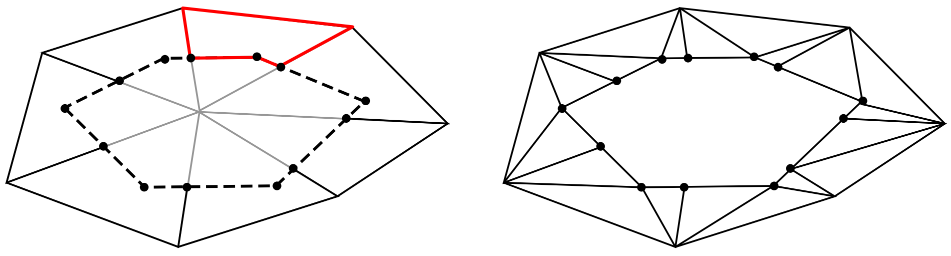


Figure 6.13: An example of a mesh inconsistency, the red dashed line represents the boundary of the inconsistency.

For each of the involved triangles we have to find a polygonal region (Figure 6.14a), where the mesh has to be repaired and triangulated using an ear cutting algorithm (the ear cutting algorithm is described, e.g., in [Ebe08]). An example of a repaired mesh is shown in Figure 6.14b.



(a) One of the polygonal regions that needs to be triangulated.

(b) Fixed mesh inconsistency.

Figure 6.14: An example of a fixed mesh inconsistency.

6.7.4 Issues Due to the Numerical Imprecisions

When I was implementing this part of the algorithm, I encountered problems caused by numerical imprecisions. When looking for the boundary of the inconsistency, it is very important to determine whether the two triangles intersect through a general line or whether they meet at one of the edges or vertices. The numerical imprecisions of floating point calculations can result in an incorrect answer to this question which will lead to wrong detection of the boundary. I was unable to eliminate this problem in my implementation which results in unreliable behavior of the application when dealing with mesh inconsistencies.

7 Results

This chapter will present some of the results of the proposed solution. In erosion-based modeling it is generally difficult to confirm the correctness of the implemented algorithm. To confirm it, the results would have to be based on real data and compared to a sequence of images capturing the erosion of the same scene in the real world. Unfortunately, erosion is a very long term process and thus it is almost impossible to obtain the real data for testing purposes. Most authors then validate their algorithms only visually, evaluating if the resulting scenes look visually plausible. The correctness of the algorithm can be as well tested by comparing the results with results of a similar published method.

7.1 SPH Implementation

The SPH implementation used in this work was taken over from [Hoe09] but it had to be altered to interact with the triangle meshes. To test if the modified algorithm behave correctly the *dam break* test was performed. It is a test commonly used to determine if the fluid particle system behaves correctly. In the beginning of the test, the particles are aligned in one half of the volume of the scene. Then the simulation starts and the particles move due to gravity and inter-particle forces. The fluid flows towards the opposite wall where it splashes and eventually the fluid stabilizes itself in a stable position. The results of the test can be seen in Figure 7.1

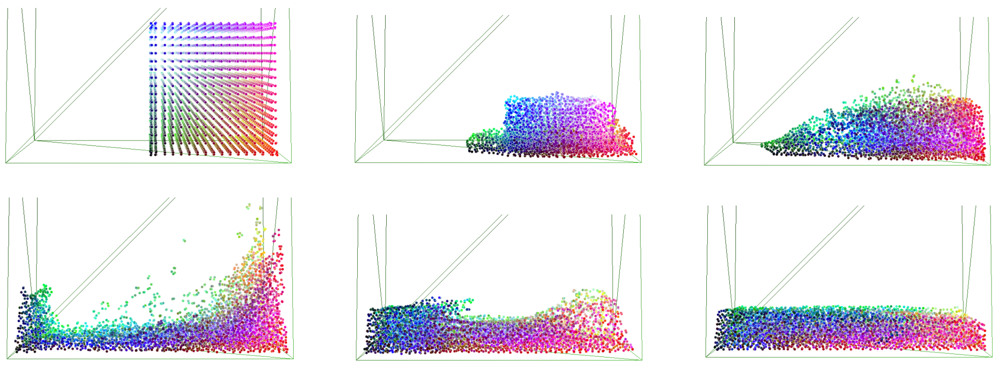
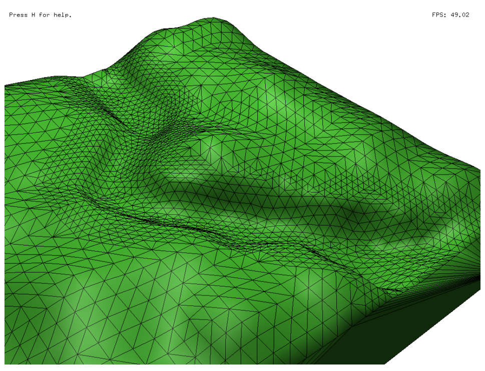


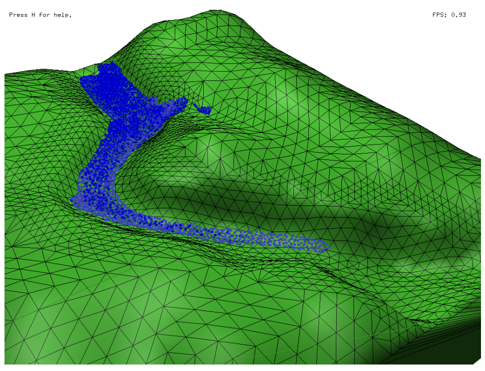
Figure 7.1: Dam break test results. (in lines left to right, top to bottom)

7.2 Simple River

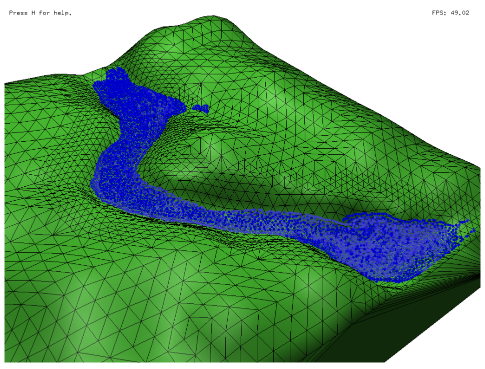
To test the behavior of the algorithm in a more realistic situation, a simple scene containing a river bed was created in Blender [1]. The fluid is generated from a source in the upper part of the scene and flows inside the river bed. The fluid interacts with the terrain and erodes its surface. The terrain was created with non-uniform resolution, the river bed where the erosion takes place is more detailed than the rest of the scene. Figure 7.2 shows the results of the simulation. Figure 7.2a captures the original scene, Figures 7.2b and 7.2c show the fluid motion during the simulation. The resulting scene can be seen in Figure 7.2d; the red color of the mesh denotes the regions where the terrain was eroded. The fluid motion is captured in Figure A.2.



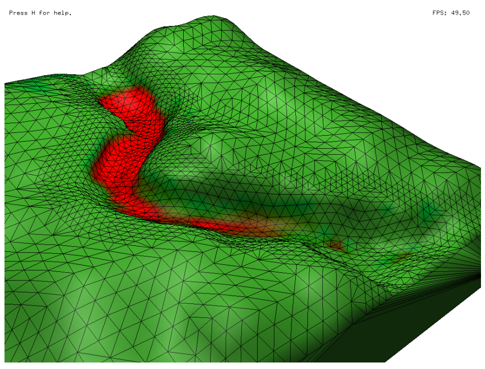
(a) Original scene.



(b) Fluid eroding the terrain.



(c) Fluid eroding the terrain.



(d) The resulting scene. The red color marks eroded regions.

Figure 7.2: Simple scene containing a river bed.

[1] Blender is an open source, cross platform suite of tools for 3D modeling. `www.blender.org`.

The scene is very simple but shows the capability of the fluid to erode the underlying terrain. Better results would be obtained with a more detailed model of the terrain. One of the problems causing the unrealistic look of the simulation is the question of the scale. The Navier-Stokes equations (see Section 2.1.2) are scale sensitive, the actual simulation scale of the fluid used in the implementation is around $1cm^3$ up to tens cm^3 [Hoe09]. On the other hand, the terrain is representing a large-scale scene, such as the landscape with the river bed. This inconsistency in scales can lead to less realistic behavior of the fluid.

7.3 Lake

A simple scenery with a lake was created to compare the method with an existing solution by Krištof et al. [KBKS09]. The authors of the reference method demonstrate its correctness with a simulation of a lake being filled by water which erodes it. Their results are shown in Figure 7.3.

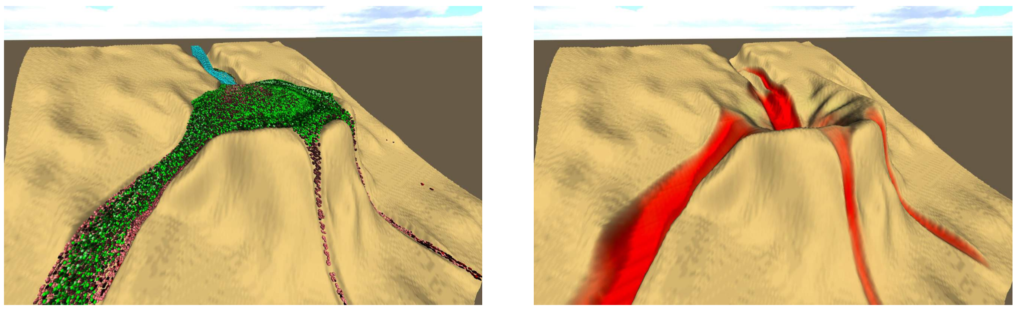

(a) The fluid erodes the terrain. (b) The red color marks the eroded regions.

Figure 7.3: An example of a lake being filled by water which erodes it, the results presented in [KBKS09].

A simplified scenery was designed in Blender in order to compare the proposed solution with the results of [KBKS09]. The scene contains a lake which is being filled by water. The water eventually fills the lake and starts to flow away, eroding the surface. The scene is captured in Figure 7.4. The process of filling the lake with water is shown in Figure A.1. The final eroded scene is shown in Figure 7.5. Figures 7.5a and 7.5c show the original terrain from two different viewpoints, Figures 7.5b and 7.5d show the same scene after the erosion is applied, the regions eroded by the fluid are marked with red color.

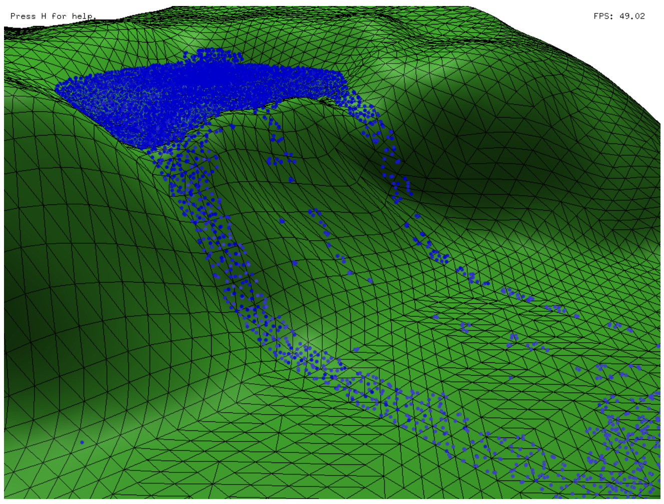
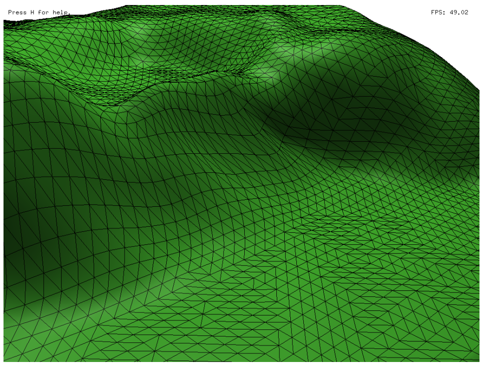


Figure 7.4: An example of a lake being filled with water. The water flows over the boundaries and erodes the surface.

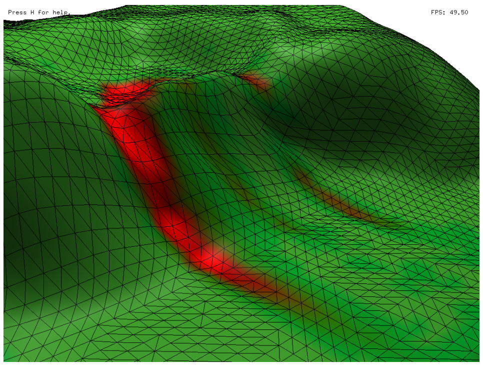
Both of the scenes generated by our method and the reference method described in [KBKS09] succeed in the creation of a terrain eroded by flowing water. The results of the reference method are more visually plausible, due to the larger resolution of the scene. The authors use 90 000 particles on a height field of an unspecified size; we created the scene in Figure 7.5 using approximately 12 000 particles.

The same scene from Figure 7.4 was used to test the adaptive tesselation of the mesh. At first, the model was tesselated in Blender in order to obtain more detailed mesh in the regions where the erosion will take place, no further tesselation was applied during the simulation. The tesselated mesh can be seen in Figure 7.6a. The implemented adaptive tesselation which was described in Section 6.5.2 was used in the scene captured in Figure 7.6b. The disadvantage of the adaptive tesselation lies in the fact that it can create long uneven triangle faces which cause problems in the erosion calculations and later during the rendering phase. The mesh subdivision which was created using Blender is more even and the calculations on it are more stable, leading to better results.

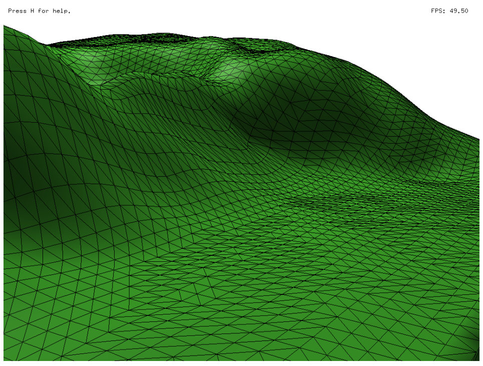
To improve the implemented tesselation scheme, a validation step could be added in future work. After the mesh is subdivided, the new faces would be tested to see if their shape fulfills the desired criteria and if not, the region of the mesh would be retriangulated. The problem of this approach is that generally the reparation of the mesh can be very computationally expensive.



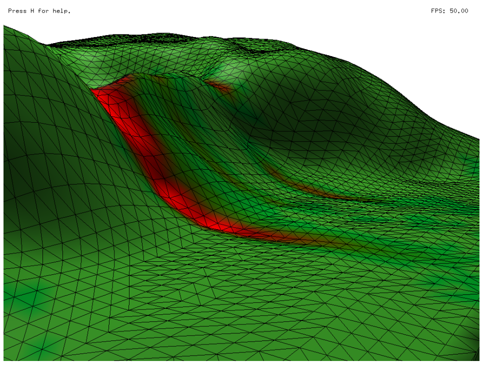
(a) The original terrain.



(b) The red color marks the eroded regions.

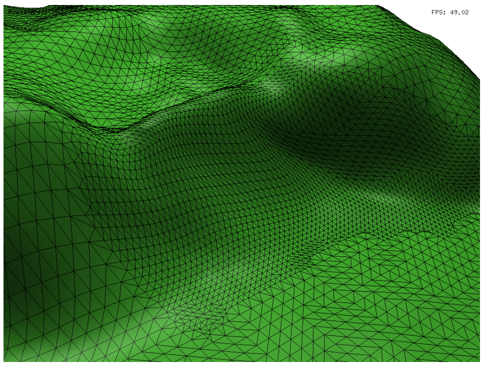


(c) The original terrain.

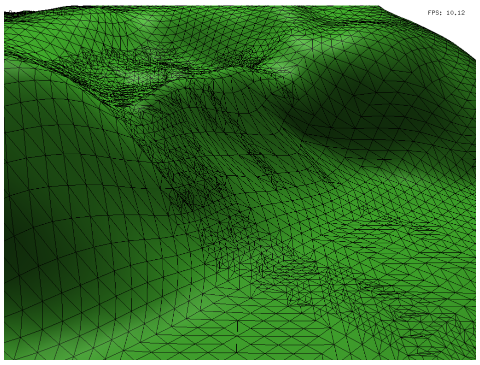


(d) The red color marks the eroded regions.

Figure 7.5: An example of a lake being filled with water. The water flows over the boundaries and erodes the surface.



(a) The mesh has higher resolution in regions where the erosion takes place.



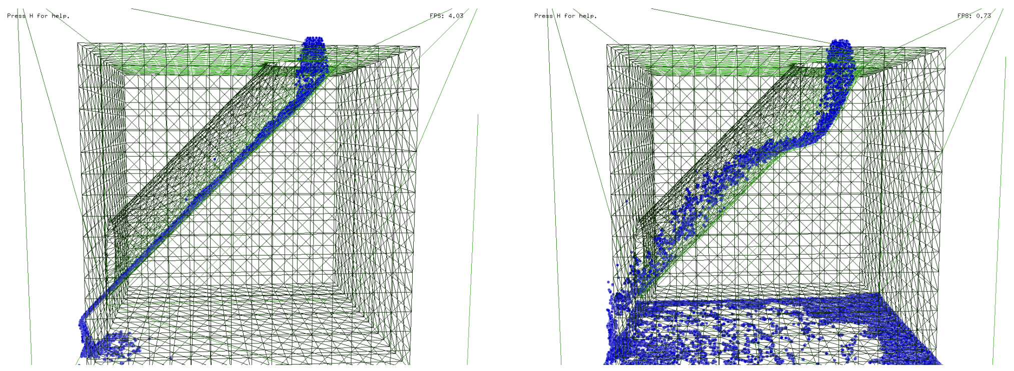
(b) The implemented adaptive tesselation.

Figure 7.6: Different approaches to mesh subdivision.

Some of the edges of the triangle would be switched during the reparation process, possibly leading to the change of the volume of the model but these changes would be minor and probably would not be visually perceptible.

7.4 Tube

A simple scene was modeled to demonstrate the capability of the algorithm to work with 3D models with concave features. The scene consists of a cube and a tube which is going through it. The fluid pours through the hole and erodes the bottom and the sides of the tube (see Figure 7.7a). The fluid force is the strongest in the region where the fluid collides with the mesh for the first time, resulting in a more noticeable erosion (see Figure 7.7b). The fluid flow is affected by the change of the topology of the mesh. The changes of the fluid flow are captured in Figure A.3.



(a) Water pours through a tube.

(b) The erosion is strongest in the region where the fluid collides with the terrain for the first time.

Figure 7.7: Water pours through a tube eroding its sides and bottom.

7.5 Computational Requirements

Computational requirements were measured for the scenes presented in the previous text. The application was written in C++ and the simulations were performed on Intel Core 2 Duo at 2GHz with operating system Windows 7 32-bit.

Table 7.1 contains information about the scene; the *Particles* column stores the number of fluid particles in the scene, the *Vertices* and *Faces* columns store the number of vertices or faces in the scene and the *Grid Cells* column stores the number of cells in the grid auxiliary data structures described in Sections 5.4.1 and 5.4.2. The size of a grid cell is constant and thus the number of the cells also describes the size of the scene.

The results are summarized in Table 7.2. It contains information about the time of computation for various scenes [2]. The columns *Particles* and *Faces* store the time necessary to assign all the particles or faces of the scene to the appropriate cell of the grid auxiliary data structure. The column *Press* and *Force* store the time of the computation of the inter-particle forces. The column *Advance* shows the time that was needed for the computation of the erosion and the subsequent modification of the mesh. The last column, *Frame*, contains the total time necessary for one iteration of the algorithm.

Table 7.1: The topology of the presented scenes.

Scene	Particles	Vertices	Faces	Grid Cells
River	12125	2126	4244	36162
Lake	12528	4538	9068	75768
Detailed Lake	12096	6065	12122	75768
Tube	12528	2356	4706	68921

Table 7.2: Computational requirements of the presented scenes. (time in [ms])

Scene	Particles	Faces	Press	Force	Advance	Frame
River	17	16	935	104	822	1 885
Lake	27	29	1 086	108	1 658	2 905
Detailed Lake	26	36	970	100	3 167	4 316
Tube	24	17	266	44	342	727

The time response was measured for scenes containing approximately the same number of particles. As can be seen in Table 7.2, the size of the scene influences the time necessary to assign the faces and particles to the appropriate cell of the grid auxiliary data structure, the time complexity

[2]The simulations were performed on an Intel Core 2 Duo at 2GHz

of this problem is linear. The distribution of the particles in the scene influences the time necessary to calculate the inter-particle forces and the fluid-solid interactions. The *River* and *Tube* scenes contain approximately the same number of particles and faces but the *Tube* scene runs more than twice faster. This is caused by the fact that the particles in the *River* scene are gathered in the river bed leading to more frequent interactions and thus slower calculations.

The authors of the reference method [KBKS09] performed their simulations on Intel Quad Core Q6600 at 2.4 GHz equipped with NVIDIA 8800 GT, their results can be seen in Figure 7.3. Their simulation of 90 000 particles was generated at a rate of one frame per 0.9 sec. Our simulation of a similar lake scene showed in Figure 7.5 is using approximately 12 000 particles; the simulation took 2.9 sec per frame. The slower runtime of our simulation is caused by more complicated calculations in the step of mesh modification. The simulation also needs to be optimized to achieve better performance.

The bottleneck of the algorithm is the calculation of the inter-particle forces and the computation of the erosion and the subsequent modification of the mesh. In future work, especially the erosion calculation and the mesh modification steps should be optimized to speed up the simulation. The algorithm could be as well implemented on GPU, using the CUDA framework.

8 Conclusion

A novel solution to the erosion-based terrain modeling was proposed in this thesis. Our solution is capable of representing both fully 3D terrains and fully 3D water effects. Other available methods working with fully 3D scenes are usually based on voxel techniques which lead to enormous memory requirements. Our solution is using a triangle mesh data structure for terrain representation. The advantage of this data structure is that its resolution can be adapted to the topology of the scene; the detailed parts of the scene are modeled with higher resolution than the flat homogeneous parts. In addition to that, several tesselation schemes which allow an adaptive division of the mesh during the simulation were implemented and tested. The user can affect the size of the triangles created during the tesselation, the smaller the newly created triangles are, the worse their shape characteristics are.

The results of erosion are shown and compared to results presented in [KBKS09]. We are using a similar approach to the erosion as the authors of the paper but their solution works only with terrains represented with a height field. The method most similar to our solution was published in [TJ10]. The authors of the paper are using a tetrahedral structure to represent 3D objects; they generate a new mesh every iteration of the method but in the process they lose the continuity of the mesh. Our solution keeps the same mesh during the whole simulation but inconsistencies can be created in the mesh. A solution to the mesh inconsistencies was proposed in this thesis, unfortunately it was not successfully implemented due to problems with numerical imprecisions of floating point calculations and for that reason our results could not be compared with the results published in [TJ10].

The work on this method will continue within my doctoral studies. There are many avenues for future research. At the first place, the implementation of the solution of the mesh inconsistencies have to be finished, in order to allow simulations of more complex scenes. Another important future work is the extension of the algorithm to support the scenes consisting of different materials. Last but not least, the implementation could be optimized and rewritten on GPU to speed up the simulation.

Bibliography

[ABB08] N. Andrysco, B. Beneš, and M. Brisbin. Permeable and absorbent materials in fluid simulations. ACM Siggraph/Eurographics Symposium on Computer Animation, Posters and Demos, 2008.

[Ach90] D. J. Acheson. *Elementary Fluid Dynamics*. Oxford University Press, 1990.

[Ama06] T. Amada. Real-time particle-based fluid simulation with rigid body interaction. In M. Dickheiser, editor, *Game Programming Gems 6*, pages 189–205. Charles River Media, 2006.

[APKG07] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3), July 2007.

[BA05] B. Beneš and X. Arriaga. Table mountains by virtual erosion. In Pierre Poulin and Eric Galin, editors, *Proceedings of the Eurographics Workshop on Natural Phenomena, NPH 2005*, pages 33–39. Eurographics Association, 2005.

[Ben07] B. Beneš. Real-time erosion using shallow water simulation. In *VRIPHYS*, pages 43–50. Eurographics Association, 2007.

[BF01] B. Beneš and Rafael. Forsbach. Layered data representation for visual simulation of terrain erosion. In *Proceedings of the 17th Spring conference on Computer graphics*, SCCG '01, pages 80–, Washington, DC, USA, 2001. IEEE Computer Society.

[BF02] B. Beneš and R. Forsbach. Visual simulation of hydraulic erosion. *Journal of WSCG*, pages 79–86, 2002.

[BR04] B. Beneš and T. Roa. Simulating desert scenery. In *WSCG (Short Papers)*, pages 17–22, 2004.

[Bri08] R. Bridson. *Fluid Simulation For Computer Graphics*. Ak Peters Series. A K Peters, 2008.

[BS09] C. Braley and A. Sandu. Fluid simulation for computer graphics: A tutorial in grid based and particle based methods. *Computer*, 2009.

[BTHB06] B. Beneš, V. Těšínský, J. Hornyš, and S. K. Bhatia. Hydraulic erosion. *Computer Animation and Virtual Worlds*, 17(2):99–108, 2006.

[CMF98] 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*, 9(4):185–194, 1998.

[Dav11] C. M. Davenport. Incompressible navier-stokes equations reduce to bernoulli's law. `http://home.comcast.net/~cmdaven/navier.htm`, 2011. [Online; Accessed 06/05/2012].

[DEJ+99] 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*, SIGGRAPH '99, pages 225–234, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[DG96] M. Desbrun and M. Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *In Computer Animation and Simulation '96 (Proceedings of EG Workshop on Animation and Simulation*, pages 61–76. Springer-Verlag, 1996.

[Ebe08] D. Eberly. Triangulation by ear clipping. http://www.geometrictools.com/Documentation/Triangulation ByEarClipping.pdf, Geometric Tools, LLC, 2008. [Online; Accessed 02/05/2012].

[FF01] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 23–30, New York, NY, USA, 2001. ACM.

[FSJ01] R. Fedkiw, J. Stam, and H. W. Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA, 2001. ACM.

[GM77] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, November 1977.

[HH01] Y. Hatano and N. Hatano. Dune morphology and sand transport. *Forma*, 16(1):65–75, 2001.

[Hib10] A. Hibbs. Navier-stokes equation. University of Warwick, Department of Physics, `http://www2.warwick.ac.uk/fac/sci/physics/pendulum/navierstokes/`, 2010. [Online; Accessed 06/05/2012].

[Hoe09] R. Hoetzlein. Fluids v.2 - a fast, open source, fluid simulator. `http://www.rchoetzlein.com/eng/graphics/fluids.htm`, 2009. Online; Accessed: 30/04/2012.

[JFBB10] M. D. Jones, M. Farley, J. Butler, and M. Beardall. Directable weathering of concave rock using curvature estimation. *IEEE Transactions on Visualization and Computer Graphics*, 16:81–94, 2010.

[KBKS09] P. Krištof, B. Beneš, J. Křivánek, and O. St'ava. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum (Proceedings of Eurographics 2009)*, 28(2):219–228, 2009.

[KM90] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 49–57, New York, NY, USA, 1990. ACM.

[KW06] P. Kipfer and R. Westermann. Realistic and interactive simulation of rivers. In *Proceedings of Graphics Interface 2006*, GI '06, pages 41–48, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[Luc77] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, December 1977.

[Man82] B. B. Mandelbrot. *The Fractal Geometry of Nature.* W.H. Freeman, San Francisco, 1982.

[MCG03] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[MDH07] X. Mei, P. Decaudin, and B. Hu. Fast hydraulic erosion simulation and visualization on gpu. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 47–56, Washington, DC, USA, 2007. IEEE Computer Society.

[MKM89] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *SIGGRAPH Comput. Graph.*, 23(3):41–50, July 1989.

[MMW01] T. Miao, Q. Mu, and S. Wu. Computer simulation of aeolian sand ripples and dunes. *Physics Letters A*, 288(1):16 – 22, 2001.

[Mon92] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574, 1992.

[ON00] K. Onoue and T. Nishita. A method for modeling and rendering dunes with wind-ripples. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, PG '00, pages 427–, Washington, DC, USA, 2000. IEEE Computer Society.

[Par65] E. Partheniades. Erosion and deposition of cohesive soils. *Journal of Hydraulics Division of the American Society of Agricultural Engineers 91*, pages 105—-139, 1965.

[Phi99] C. L. Phillips. The Level-Set Method. *The MIT Undergraduate Journal of Mathematics*, 1:155–164, 1999.

[Pur09] V. Purchart. Modelování písčitého terénu pro virtuální realitu. Master's thesis, University of West Bohemia, Pilsen, Czech Republic, 2009.

[SSP07] B. Solenthaler, J. Schläfli, and R. Pajarola. A unified particle model for fluid–solid interactions: Research articles. *Comput. Animat. Virtual Worlds*, 18(1):69–82, February 2007.

[TJ10] L. A. Tychonievich and M. D. Jones. Delaunay deformable mesh for the weathering and erosion of 3d terrain. *Vis. Comput.*, 26(12):1485–1495, December 2010.

[WCMT07] C. Wojtan, M. Carlson, P. J. Mucha, and G. Turk. Animating corrosion and erosion. In *Proceedings of the Eurographics Workshop on Natural Phenomena, NPH 2007*, pages 15–22. Eurographics Association, 2007.

[Wik12] Wikipedia. Tetrahedron — wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Tetrahedron&oldid=489166298`, 2012. [Online; Accessed 02/05/2012].

A Other Results

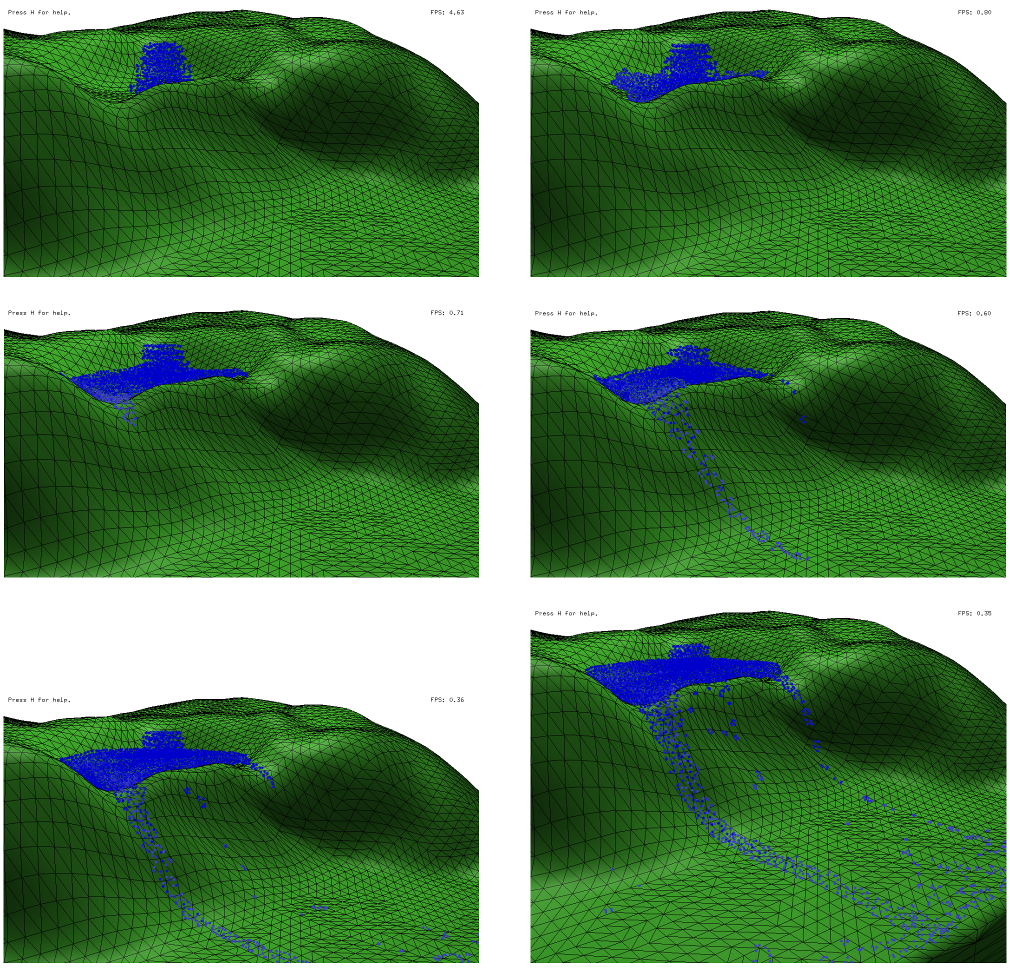


Figure A.1: An example of a lake being filled with water. The water flows over the boundaries and erodes the surface.

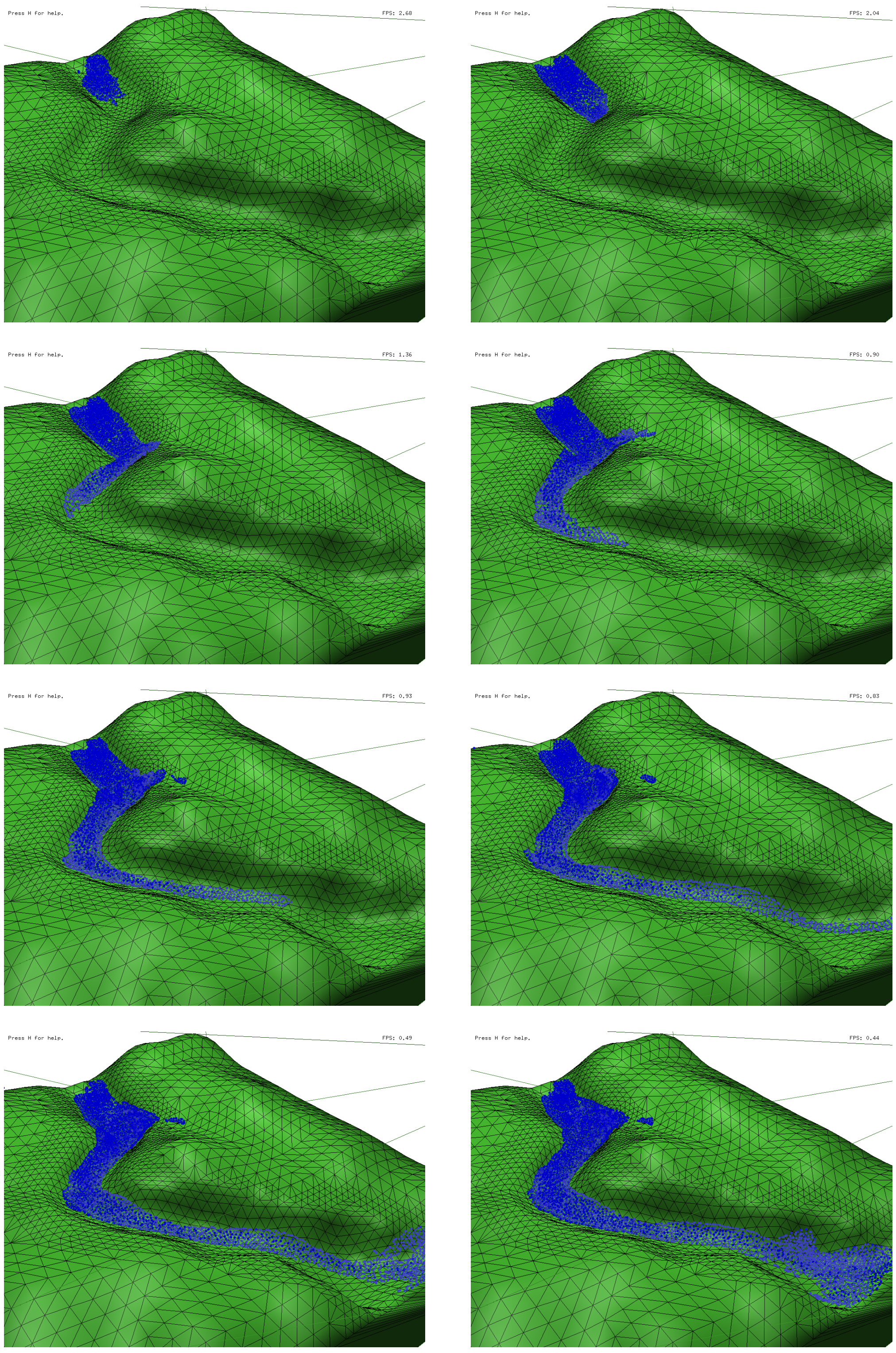


Figure A.2: A simple scene containing a river bed.

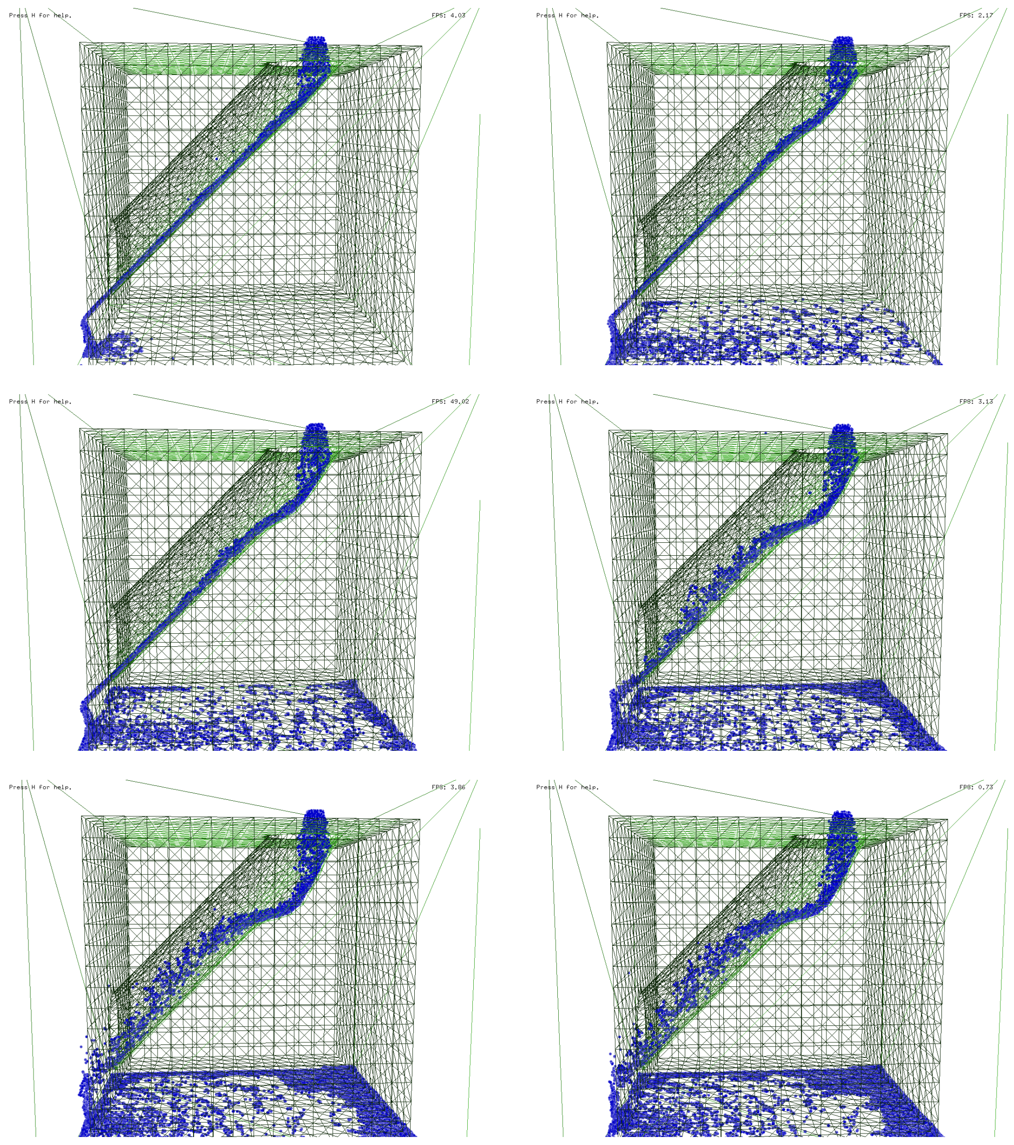


Figure A.3: Water pours through a tube eroding its sides and bottom.

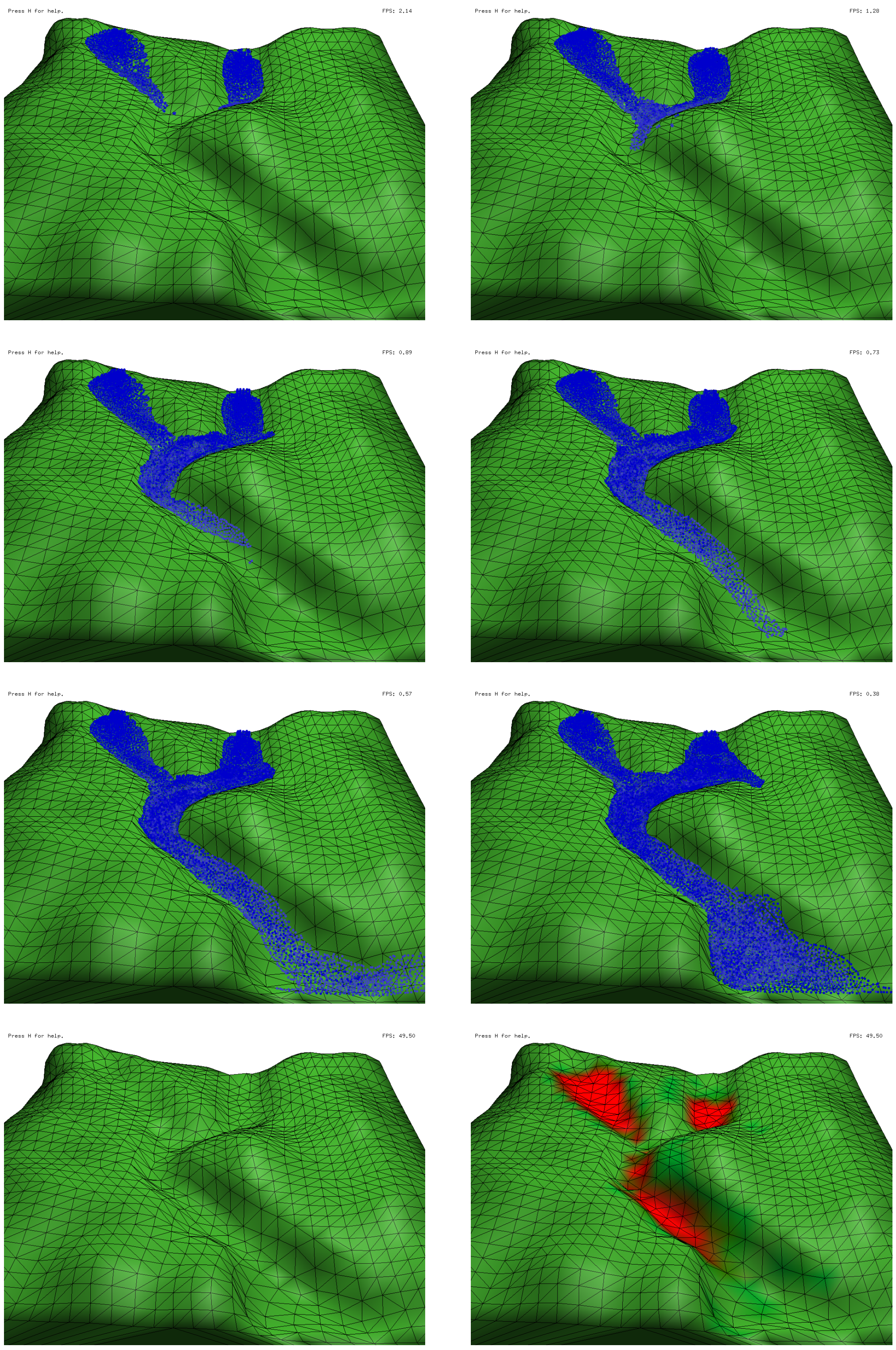


Figure A.4: An example of a scene with two water sources.

B User Manual

To run the program you need the following files:

- *fluids.exe*, the executable
- *glew32.dll* and *jpeg62.dll*, the necessary libraries
- folder *data* containing the 3D models in .obj format
- folder *img* for exporting images
- folder *export* for exporting models
- folder *settings* with settings files

The program can be started with a command: *fluids.exe [settings_file]*. The parameter *settings_file* is compulsory and contains the path to the settings file. An example of the content of the settings file can be as follows:

```
# Scene settings
----------------
# models (filename reverseWinding scaleFactor translation)
m data/cube.obj 0 30/30/30 0/0/0
m data/model.obj 1 30/30/30 0/0/0

# fluid source (rate position size)
f 10 0/0/10 10/10/10
f 5 -20/-20/10 3/3/3

# maximum number of particles
p 1000

# distance limit for tesselation
d 100.5

# material (sediment_change max_sediment_in_particle critical_shear)
s 0.02 1.0 0.25
```

The lines starting with a symbol # are comments, they do not affect the application. Line affecting the application have to start with *m, f, p, d* or *s.*

- Letter *m* - the file can contain more lines beginning with the letter m. The line has a following form *m [filename] [reverseWinding] [scaleFactor] [translation]* and contains the information about the 3D model. The first model in the file represents the bounding box of the scene; it will not be affected by the erosion or tesselation and its size should be adapted so that the whole scene would fit into it.

  filename - the path to the .obj model.

  reverseWinding - variable to decide if the winding of the triangles in the mesh should be reversed, 1 - reverse the winding, 0 keep the current winding.

  scaleFactor - the scale factor that should be applied to the model, the values of the vector are separated by /.

  translation - the translation vector that should be applied to the model, the values of the vector are separated by /.

- Letter *f* - the file can contain more lines beginning with the letter f. The line has a following form *f [rate] [position] [size]* and contains the information about fluid sources. The fluid source has to be located in a legal position, it cannot be located inside a model.

  rate - the rate of the fluid generation. For the value of rate equal to 10, new particles will be generated every 10 frames.

  position - the position of the water source, the values of the vector are separated by /.

  size - the size of the water source, the values of the vector are separated by /.

- Letter *p* - maximum number of the particles allowed in the scene.

- Letter *d* - the distance limit for tesselation. The triangle will be tesselated if the distance between its vertices and the interacting particle is larger than *d*.

- Letter *s* - the material constants. The line has a following form *s [sediment_change] [max_sediment] [critical_shear]*

  sediment_change - a constant influencing the erosion process. For bigger values the erosion will be more aggressive.

max_sediment - the maximum sediment that can be carried by the particle.

critical_shear - the value of critical shear, a constant used in erosion calculation. For bigger values, the force applied to the terrain has to be bigger for the erosion to take place.

After the application has started, the user can influence its behavior by pressing the following keys:

- *H* - shows help on the screen
- *space* - pause the simulation
- *escape* - ends the simulation
- *J* - turns on or off the stepping of the simulation. When on, the simulation will pause itself after each step
- *L* - switches between the wireframe and solid model
- *K* - shows or hides the wireframe when in solid mode
- *D* - changes the way of rendering particles
- *S* - changes the shading modes
- *N* - decreases the maximum number of particles in the scene and restarts the simulation
- *M* - increases the maximum number of particles in the scene and restarts the simulation
- *V* - turns on or off the capturing of all the rendered frames
- *B* - saves the current frame to the folder *img*
- *Q* - shows or hides the original model (without erosion)
- *T* - saves the model in .obj format
- *1* - measures the time of each iteration and writes the results in the console
- *2* - hides the bounding box of the scene

C Proceedings of SVK 2012: Modeling of Erosion Impacts on the Terrain

Modeling of Erosion Impacts on the Terrain

Věra Skorkovská[1]

1 Introduction

In the field of computer graphics, we often find ourselves in the need of visually plausible models of terrain. Creating such a terrain using a modeling software would be very time-consuming and the results may not be as good as we would need. For that reason many techniques solving this problem were developed; erosion-based terrain modeling is one of the possible approaches. It has been an important part of computer graphics for more than twenty years but many problems still remain unsolved. Many solutions are only capable of working with 2.5D terrain, these solutions do not allow formations such as caves or overhangs. Other group of solutions supports the fully 3D terrain but these methods are usually very memory consuming and not capable of running with real-time response.

Our solution is addressing the problem of hydraulic erosion, which has the greatest influence on the terrain alterations. The solution is representing the fluid as a particle system and the terrain is stored as a triangle mesh, leading to lower memory requirements while keeping the ability to simulate fully 3D phenomena.

2 Proposed Solution

The solution is based on Smoothed-particle hydrodynamics (SPH), an approximative numeric solution to equations describing the fluid dynamics. SPH represents the fluid as a set of particles that interact over a specified distance called the *smoothing radius*. The advantage of the SPH simulation is that the particles are localized only in the regions where the fluid is present and so we can limit the computation to this locations. For the implementation of SPH we use a library *Fluids v.2* by Hoetzlein (2009).

The terrain in erosion simulations is usually represented as a height field or a voxel grid. These regular data structures simplify the erosion calculations but they have significant drawbacks as well. A height field does not allow the creation of a fully 3D features of the terrain; voxel grids are capable of describing a fully 3D scene but they are very memory consuming. Our solution represents the terrain with a triangle mesh, which leads to lower memory requirements. Furthermore, the resolution of the triangle mesh can be adjusted according to the complexity of the scene, allowing higher resolution in the regions with great details and lower resolution in the homogeneous regions.

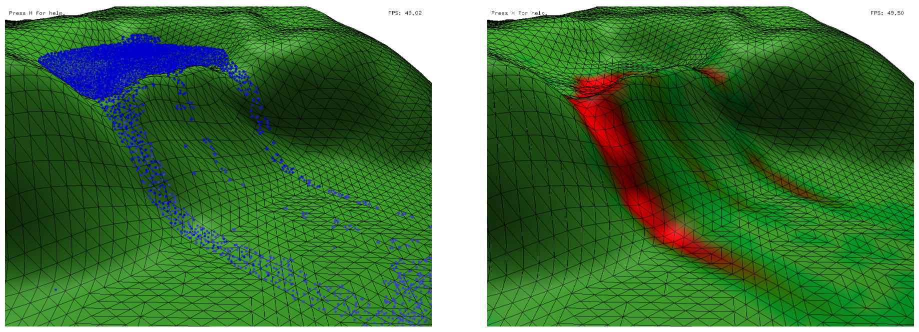
The SPH particles simulate the motion of the fluid and interact with the terrain when a collision occurs. On collision, the erosion or deposition amount is calculated using the equations published in Krištof et al. (2009). The erosion sediment exchange is calculated for all the particles in the scene and after that the triangle mesh is updated. During the modification of the mesh, an inconsistency can be created when two parts of the mesh intersect. The solution to this problem was suggested but it is not yet successfully implemented, due to the numerical imprecision issues.

[1] student of the master study programme Computer Science and Engineering, specialisation Computer Graphics and Computer Systems, e-mail: vskorkov@students.zcu.cz

3 Results

In erosion-based terrain modeling it is very complicated to confirm the correctness of the generated results. To confirm it, the simulated scene would have to based on real data and the results would have to be compared to a sequence of images capturing the process of erosion in the real world. Unfortunately, erosion is a very long term process and thus it is almost impossible to obtain the real data for testing purposes. Most authors then validate their algorithms only visually, evaluating if the resulting scenes look visually plausible.

The algorithm was tested on several scenarios, the results were visually acceptable. Better results could be achieved with more detailed terrain models. An example of a scene generated by the implementation is shown in Figure 1.



(a) The terrain is being eroded by flowing water.

(b) The resulting terrain. The red color marks the eroded regions.

Figure 1: An example of a lake being eroded by water.

4 Conclusion

A novel solution to the erosion-based terrain modeling was proposed which is capable of representing both fully 3D terrains and fully 3D water effects. The results of the erosion were shown and visually confirmed, however, it is impossible to prove their correctness as we do not possess any real data of a similar scene.

During the modification of the terrain, an inconsistency can be created in the mesh; the implementation of this subproblem was not successful due to the issues of numerical imprecisions. The solution to this subproblem will be a part of future work, along with the extension of the algorithm to support different materials and the optimization of the implementation.

Acknowledgement

This work has been supported by the Ministry of Education, Youth and Sports, project LH11006 Interactive Geometrical Models for Simulation of Natural Phenomena and Group Behavior.

References

Hoetzlein, R., 2009. Fluids v.2 - a fast, open source, fluid simulator. http://www.rchoetzlein.com/eng/graphics/fluids.htm, 2009. Online; Accessed: 30/04/2012.

Krištof, P. and Beneš, B. and Křivánek, J. and Štava, O., 2009. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum (Proceedings of Eurographics 2009)*, Vol. 28 No. 2, pp 219–228.