Západočeská univerzita v Plzni
Fakulta aplikovaných věd

MODELOVÁNÍ VLIVU EROZE NA GEOMETRICKÉ OBJEKTY

Ing. Věra Skorkovská

disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: prof. Dr. Ing. Ivana Kolingerová

Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2019

University of West Bohemia
Faculty of Applied Sciences

MODELING OF EROSION IMPACT ON GEOMETRIC OBJECTS

Ing. Věra Skorkovská

doctoral thesis
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: prof. Dr. Ing. Ivana Kolingerová

Department: Department of Computer Science and Engineering

Pilsen 2019

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že jsem tuto práci zpracovala samostatně s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne Ing. Věra Skorkovská

Abstrakt

Simulace eroze je důležitým problémem v oblasti počítačové grafiky. Nejvýznamnějšími erozními procesy v přírodě jsou zvětrávání a hydraulická eroze. Mnoho metod se těmito problémy zabývá, ale většinou jsou tyto metody založeny na výškových mapách nebo volumetrických datech. Výškové mapy neumožňují simulaci složitých plně trojrozměrných scén, zatímco volumetrická data mají vysoké paměťové nároky. Tato disertační práce zkoumá výhody reprezentace erodovaných objektů pomocí trojúhelníkových sítí a navrhuje řešení problémů, které vznikají v důsledku použití této datové struktury. Trojúhelníkové sítě se ukázaly být vhodnou datovou strukturou pro použití při simulaci eroze díky jejich adaptivitě a možnosti modelovat složité konkávní prvky scény. Použití trojúhelníkových sítí však přináší nové problémy, například problém vzniku nekonzistence sítě v důsledku silné eroze nebo problém simulace složitých scén složených z více materiálů. Tato práce zkoumá zmíněné problémy a navrhuje jejich možná řešení.

Abstract

Erosion simulation is an important problem in the field of computer graphics. The most prominent erosion processes in nature are weathering and hydraulic erosion. Many methods address these problems but they are mostly based on height fields or volumetric data. Height fields do not allow the simulation of complex fully 3D scenes while the volumetric data have high memory requirements. This thesis explores the advantages of representing the eroded objects as triangular meshes and proposes solutions to problems that arise due to the use of this data structure. Triangular meshes prove to be an advantageous data structure for erosion simulations due to their adaptivity and the possibility to model complex concave features. However, the use of the triangular meshes brings new problems to the erosion simulation, such as the problem of creation of an inconsistency in the mesh due to heavy erosion or the problem of simulation of complex scenes composed of multiple materials. This thesis explores these problems and suggests possible solutions.

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 my supervisor prof. Dr. Ing. Ivana Kolingerová for her valued advice and never-ending patience. This thesis would not have been possible without prof. 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 during my studies.

Contents

Chapter 1

Introduction

Erosion is an important land-changing process and as such it deserves our attention. Erosion is mainly observed as a process that shapes the landscape over the years, but its effects can also be observed on smaller individual objects. As the erosion of the land surface is the most important, it will be given higher attention. However, the erosion algorithms can be generalized and applied to general objects as well.

In computer graphics, erosion is mainly seen as a means of creating visually plausible models of terrains for the use in various fields, such as animation or computer games. The existing erosion approaches usually fall into one of two groups: volumetric approaches or approaches working on a height field. The height field data structure is very straightforward and the algorithms built upon it can be very fast, but it cannot be used to model fully 3D scenes as a consequence of its simplicity. On the other hand, the volumetric solutions can represent any object with concave features and complex topology, but the methods are memory demanding and usually much slower than the methods based on height fields.

While some of the most recent erosion methods in the field of computer graphics focus on automated generation of large visually plausible complex landscapes ([Cor+16; Cor+17]), there are still many open problems in smaller-scale erosion simulations as none of the existing methods allows for the simulation of erosion on small-scale but complex fully 3D objects that would have sufficient quality and at the same time would not be very computationally expensive.

Our main interest lies in the simulation of hydraulic erosion and weathering, as they have the biggest influence on the landscape evolution. Our goal is to propose a spatially adaptive solution that would be capable of working with fully 3D objects while maintaining real-time characteristics.

Triangular mesh data structure fits our requirements as it can represent any object and its resolution can be adaptively adjusted according to the local complexity of the eroded object. The adaptivity of triangular meshes and the possibility to model complex concave features are the main advantages of the data structure in relation to erosion simulation. The triangular mesh data structure allows to model an object adaptively with varying density of vertices based on the complexity of the object, leading to lower memory requirements of the scene when compared to the volumetric

approaches, while it keeps the ability to model fully 3D scenes with complex features. Also, as triangular meshes are by far the most commonly used object representation in computer graphics, it is reasonable to design methods using this structure.

However, the use of triangular mesh data structures introduces other challenges. It may be more difficult to estimate the object properties such as curvature on irregular data structures. Also, when the triangular mesh is being deformed during the simulation, it has to be deformed in such a manner that does not damage its integrity or alternatively, a mesh repair algorithm needs to be applied to restore the correct topology.

Another problem that arises due to the use of triangular mesh data representation is the problem of erosion of non-homogenous objects composed of several different materials. Real-life erosion scenes are usually formed of multiple materials and so a reliable means of material definition suitable for triangular meshes is needed.

1.1 Aim of the Thesis

Our aim is to propose a complex erosion simulation approach that will be capable of simulating hydraulic erosion and weathering on both complex terrains, containing features such as caves, tunnels or overhangs, and also general objects, such as statues or other man-made objects. Our focus is on the creation of a unified solution that could be used for both scientific simulations, such as the evolution of landscape or river systems, or for creation of visually plausible eroded terrains or general objects - just by exchanging the underlying erosion simulation equations.

As the triangular mesh data structure seems to be a valid candidate for the data representation structure for the proposed erosion approach due to its adaptivity and capability of representing any complex features, a secondary aim of the thesis is to prove the soundness of the data structure choice and to propose possible solutions to the problems that are caused by the use of the data structure.

The objective of the thesis is to propose solutions to the following problems:

- Hydraulic erosion of complex concave terrains, including terrains with caves, tunnels or overhangs
- Hydraulic erosion of general objects, such as statues or other man-made objects
- Weathering simulation based on similar principles as the hydraulic erosion simulation
- Correct handling of topology changes in the triangular mesh, such as merging or splitting
- Erosion simulation of multiple material scenes

1.2 Thesis Structure

The thesis is divided into two main parts. The first part of the thesis explains the background of the problem and summarizes the state-of-the-art solutions that focus on erosion simulation and connected areas of research. The second part builds up on the existing methods and presents our contributions to the problem of erosion simulation.

The structure of the thesis is as follows: Chapters 2 and 3 describe the state-of-the-art methods used in erosion simulation in computer graphics. Chapter 4 summarizes the data structures that are the most common in erosion simulations. Chapter 5 introduces the problems of mesh repair and describes the existing solutions and their strengths and weaknesses. Chapter 6 lists the most common approaches to the definition of multiple materials.

Our contributions start with Chapter 7 that shortly introduces the proposed methods which are then discussed in further detail in the following Chapters. Chapter 8 describes our solution to the hydraulic erosion problem based on the combination of terrains represented by triangular meshes with fluid represented as a particle system. Chapter 9 extends the solution to create a unified approach for simulation of both hydraulic erosion and weathering. Chapter 10 addresses the problem of repair of intersections that can be created during the erosion simulation. Chapter 11 presents several possible approaches for simulation of multiple materials. Chapter 12 takes the erosion principles and applies them to 3D head models to simulate aging. Finally, Chapter 13 summarizes our contributions and suggests possible avenues for the continuation of our research and Chapter 14 concludes the thesis.

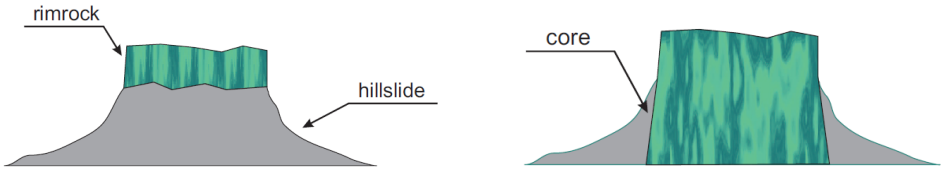
Chapter 2

Erosion

Erosion is a process happening in nature by which material such as sand and rocks is taken from its original location, transported and deposited at another location by the means of water, wind or other natural forces. Erosion is a long-term process that has a huge impact on the evolution of the landscape over the years. It can be observed in large-scale scenarios as well as on smaller individual objects, such as rocks and stones. Man-made objects, e.g., buildings, roads or statues, are also affected by the erosion processes.

Erosion is usually studied in the context of terrain alteration where it is the most noticeable. In computer graphics, erosion simulation has a long history as a means of generating visually plausible artificial terrains. The first attempts to create artificial terrains tried to generate the terrains straight away but the generated terrains usually looked too sharp and unrealistic. As the performance of the computers improved, new approaches to the terrain modeling appeared. Most of the improved methods were based on erosion. The approaches simulate the natural processes that form the terrain in the real world. A generated artificial terrain (e.g., a terrain created using a fractal geometry [Man82]) is taken as a base and altered using an erosion simulation. However, erosion as it takes place in the nature is a very complex process and 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.

Erosion processes can be subdivided into three main categories: weathering, wind erosion and hydraulic erosion. Hydraulic erosion is leaving the most significant mark on the landscape and as such it draws the biggest attention of the researchers. 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 generally not causing so substantial changes but in certain landscapes, such as desert sceneries, its effects can be essential.



(a) Table mountain shape (b) Inner structure of a table mountain

Figure 2.1: Generation of table mountains [BA05]

2.1 Weathering

Weathering is a process of breaking down rocks, e.g., due to the contact with chemical substances, living organisms or due to thermal changes. Weathering happens in place without 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 most common weathering process. It is caused by expansion and contraction of the material due to the temperature changes. This process is the most distinctive in the deserts, where the temperatures vary greatly between day and night. The eroded material drops off of the rocks and falls down to the ground, where it creates talus slopes with a critical angle defined by the material. When the angle of the talus exceeds a critical value, movement occurs and the material slides down to restore the state of equilibrium.

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. In the second pass, a 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 adopted this approach and used it in their simulations.

Beneš and Arriaga presented a method designed to generate table mountains in [BA05]. The goal of the work is a geologically inspired algorithm which can simulate visually plausible terrains. They work with two types of material: hard rock and soft eroded material (sand). The hard material is eroded when exposed to moisture and thermal changes and the eroded parts are falling off and becoming the soft material (Figure 2.1). The motion of the soft material is simulated by a diffusion algorithm to achieve an equilibrium state and the characteristic hillsides of table mountains.

Dorsey et al. [Dor+99] presented a method for modeling and rendering of weathered stones. They proposed a novel voxel surface-aligned data structure called *slab* that works as an intermediate between the stone and the surrounding erosion factors.

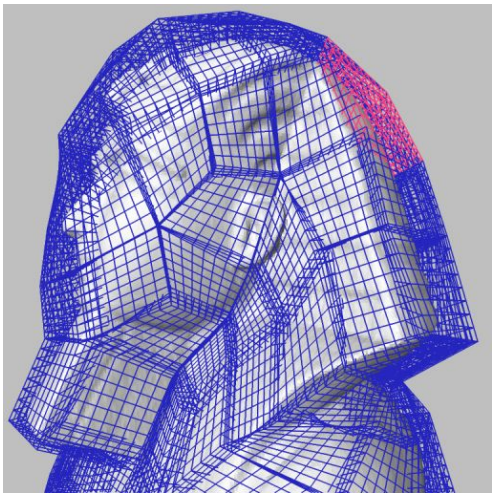


Figure 2.2: Slab data structure [Dor+99]

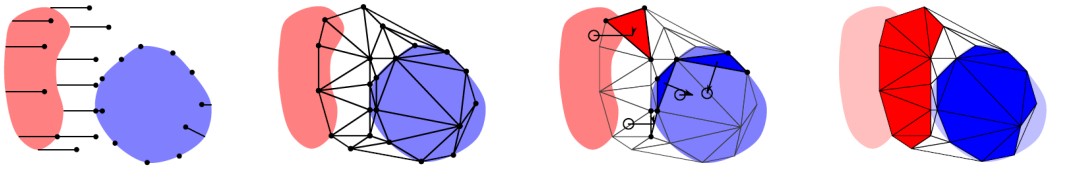
The slabs constrain the erosion computation to the regions adjacent to the surface of the object and allow the sampling of the object beneath the surface. Each slab is a trilinear volume defined by its eight corners, neighboring slabs share the corresponding four corners (Figure 2.2). They designed the algorithm to simulate the flow of water and dissolution and transportation of minerals that causes the surface erosion.

Beardall et al. [Bea+07] proposed a method for generating sandstone structures called *goblins* using a voxel-based simulation of spheroidal weathering. They approximate the effects of spheroidal weathering by using bubbles centered in the voxels. Spheroidal weathering is simulated for voxels on or near the surface by computing the ratio of air to stone within a fixed bubble-shaped volume around each voxel. The amount of erosion at each voxel is then calculated as a function of that ratio and the voxel's resistance to weathering.

A similar method that works with 3D objects was proposed by Jones et al. [Jon+10]. 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).

Tychonievich and Jones proposed a weathering method using a tetrahedral mesh data structure in [TJ10]. The 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 (Figure 2.4a) and a new Delaunay triangulation (DT) is constructed (Figure 2.4b). 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

Figure 2.3: Various weathered rocks created by spheroidal and cavernous weathering [Jon+10]



(a) Initial models with vertices offsets (b) DT of the offseted vertices (c) Material correlation (d) Resulting models

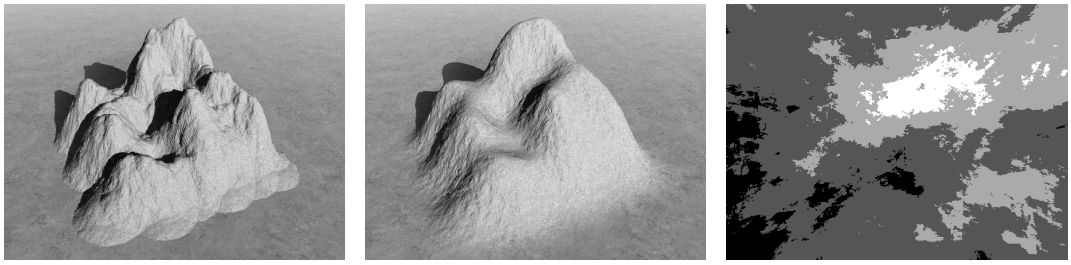
Figure 2.4: Material restoration in tetrahedra-based erosion simulation [TJ10]

(Figure 2.4c). The newly created DT serves as a base for the next iteration of the algorithm (Figure 2.4d). The authors demonstrate the use of the data structure in weathering and erosion simulation whose objective is the creation of visually plausible scenes 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.

Warszawski and Nikiel proposed an erosion method for terrains with hardness layer in [WN14]. In their approach, the erosion force is applied uniformly across the entire model with local distribution of varying terrain sensitivity. They use a two-layered model; the first layer represents the information about the height of the terrain stored as a height map, while the second layer stores the hardness of the terrain. They use synthetic data to demonstrate the capability of their method to drive the erosion speed based on the terrain hardness (see Figure 2.5).

2.2 Wind Erosion

Wind erosion has major influence in arid landscapes where the hydraulic erosion is nearly absent. Wind erosion consists of two main parts, abrasion of rocks and transferring of the material particles. Abrasion happens when the wind carries the



(a) Height-field layer (b) Model after 100 steps (c) Hardness-field layer

Figure 2.5: Terrain model with hardness layer [WN14]

Figure 2.6: Desert scenery with wind-ripples [ON00]

material particles and hits the solid surface of a rock, causing small pieces of the material to fall off. The transfer of material takes place when the wind captures the particles and moves them to a different location, creating formations such 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 the wind grabs a particle and lifts it to move it, while creep is occurring when the wind moves the particle on the surface. Equations describing the creation of the sand dunes and wind-ripples are proposed in the paper. The two sand structures are created separately and then combined during rendering using bump-mapping technique. Their results can be seen in Figure 2.6.

Beneš and Roa extended their algorithm by adding an interaction with obstacles in [BR04]. The material is accumulated on the windward side of the obstacles and the wind shadow appears on the leeward side. The intensity of the wind is reduced in the wind shadow based on the simplified geometry of the obstacle, causing a reduction of the wind-ripples. The accumulation on the windward side is done by an 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 a generated scenery is shown in Figure 2.7.

Miao et al. [MMW01] introduced a method to simulate the initiation and evolution of the wind blown sand ripples and dunes. Their model is capable of reproducing

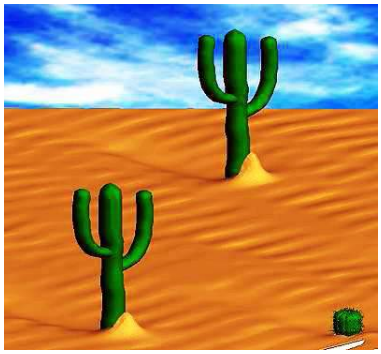


Figure 2.7: Material is accumulated on the windward side of the obstacles [BR04]

sand ripples and describe a repair of the 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 such as barchans and linear dunes from the initial random state. They studied 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.

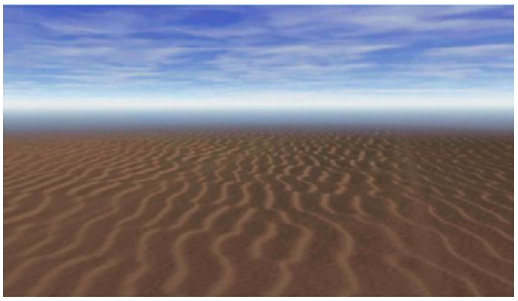
More recently, Wang and Hu proposed a physically and procedurally based method for the simulation of sand movement and sand ripple evolution in [WH12]. The method is based on the *physics of blown sand*, i.e., the mechanical behavior of individual sand grains, and defines a set of rules to create realistic desert scenes. A simplified vegetation and wind field model is used to speed up the simulation. The method is implemented on the GPU to achieve real-time simulations. Several examples of desert scenes generated by the method are shown in Figure 2.8.

Most recent papers from this area of research shift from the graphical view point to more physically based approaches. Abdikerem et al. [Abd+14] propose three kinds of models with different complex air flow fields for the purpose of simulating the sand ripple formation process. Wei et al. demonstrate in their paper [Wei+16] that the standard sine-shaped model for sand surface electromagnetic scattering does not correctly describe natural sand ripples and propose a discrete model for sand ripples generation.

2.3 Hydraulic Erosion

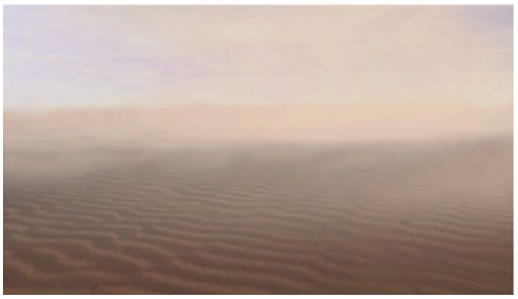
Hydraulic erosion is the erosion caused by still or flowing water, but also the erosion caused by glaciers or avalanches. It has the most noticeable impact on the evolution of the landscape. The erosion caused by rain can smooth the rocks and mountains while the streams and rivers can cause the creation of river beds, valleys or canyons.

Hydraulic erosion methods can be subdivided into two categories: *physically inspired methods* and *physically based methods*.



(a) Ripple marks in desert

(b) A series of tranverse dunes



(c) Aeolian sand flow in desert

(d) Aeolian sand flow on dunes

Figure 2.8: Example desert scenes [WH12]

2.3.1 Physically Inspired Solutions

The physically inspired methods take inspiration in natural processes but do not try to simulate them exactly. Their main purpose is to mimic the erosion impacts with as little computational effort as possible, so that the results looked good without the need to simulate the exact physical erosion processes.

Musgrave et al. [MKM89] proposed a simple hydraulic erosion algorithm simulating rain effects on the terrain. The method starts by the deposition of water (rain) on the 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 the water flow. They use particles to approximate the water volume, the erosion is evaluated 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 desired visual effect. 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

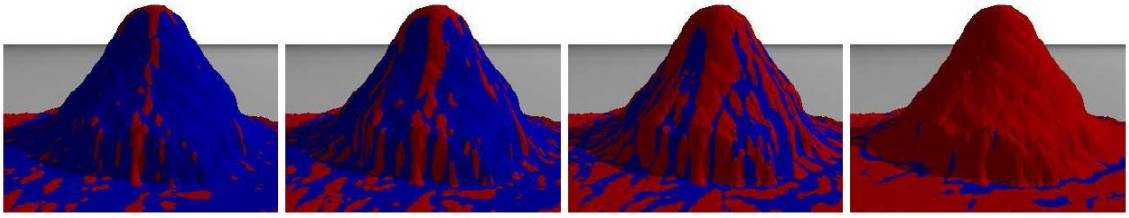


Figure 2.9: Rain simulation [BF02]

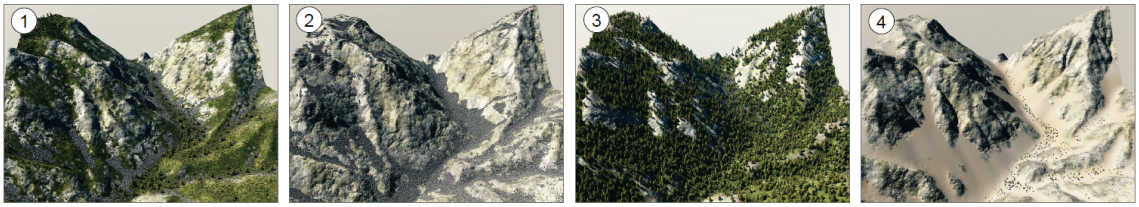


Figure 2.10: An example of generated landscapes: (1) default parameters, (2) a high altitude rocky region, (3) a dense forest on a fertile ground, and (4) a sand-filled desert with drought-adapted vegetation [Cor+17]

which causes the excess sediment in the particles to be deposited. An example of simulation of fictive rain on the martial volcano Olympus Mons is captured in Figure 2.9.

Cordonnier et al. [Cor+17] used a simple erosion method similar to the method proposed by Musgrave et al. [MKM89] to create a complex framework for interactive authoring of large-scale terrains. Their algorithm works on a layered height field and allows the simulation of the mutual interaction between vegetation and erosion processes. Their solution offers a great deal of user control; the user can influence the scene and the simulation parameters during the modeling process. Figure 2.10 shows an example of a landscape generated using their approach.

2.3.2 Physically Based Solutions

The physically based methods use hydrodynamics in order to simulate the erosion processes more exactly. However, even methods from this category usually introduce some simplifications of the fluid dynamics in order to speed up the simulation.

Fluid Dynamics A real fluid can change its volume but the changes are so small that it is not possible 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 allowing us to treat the fluids as being incompressible.

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

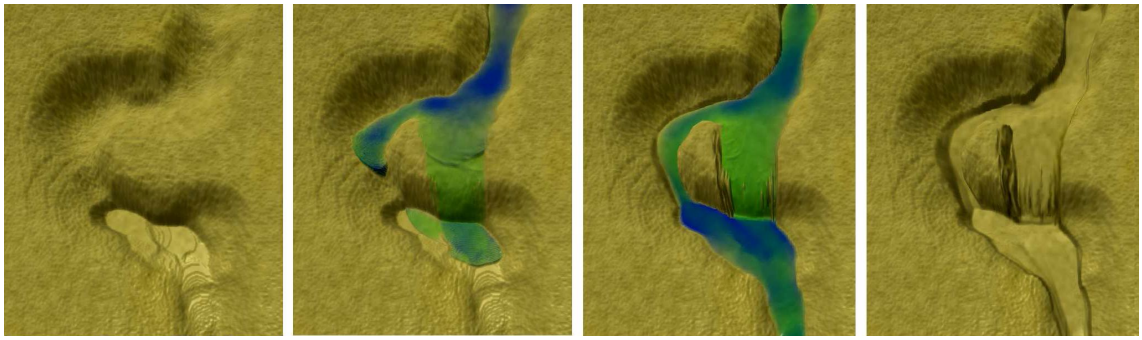


Figure 2.11: Water flow eroding a river bed [MDH07]

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

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

The symbol $\mathbf{v}$ $[m\cdot s^{-1}]$ is used for the velocity of an infinitesimal element of mass at a point in the 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. $\mathbf{f}$ $[N\cdot m^{-3}]$ stands for the external forces. The constant μ $[Pa\cdot s]$ represents the viscosity of the fluid. 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*. [Dav11], [Hib10]

Shallow Water Simulation Shallow water simulation is a simplification of Navier-Stokes equations for the 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 the water is assumed 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 it is satisfactory for many simpler cases [KM90].

Beneš used the shallow water model to create an interactive hydraulic erosion simulation in [Ben07]. He proposed a method for the simulation of erosion and deposition of the sediment. The deposition takes place when the water evaporates or the dissolved material in the water exceeds the critical level. Mei et al. implemented a similar method for rapidly moving water on the GPU in [MDH07]. Their results can be seen in Figure 2.11.

Šťava et al. took inspiration in the two aforementioned approaches ([Ben07; MDH07]) and proposed a method for interactive physics-based terrain modeling [Sta+08]. They represent the terrain as a layered height field and provide a user with a set of tools that can be used to control the evolution of the terrain. The

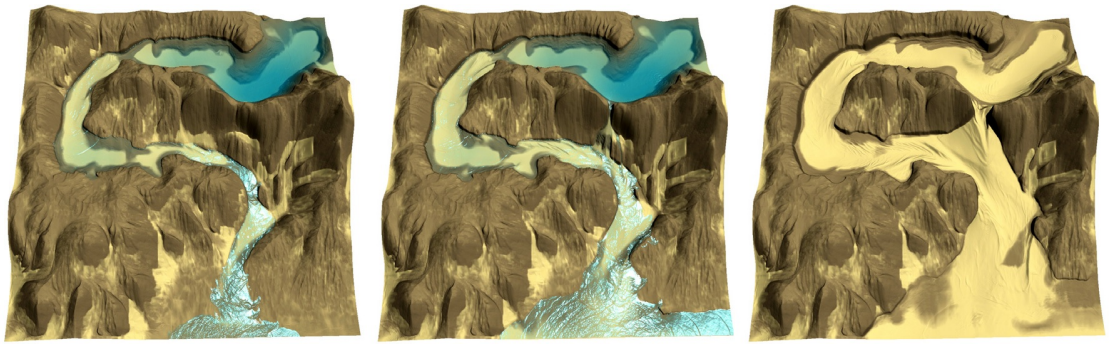


Figure 2.12: Water flow causing a meander break [Sta+08]

algorithm is implemented on the GPU which allows the algorithm to run at interactive rates for terrains with resolution of 2048 x 1024 and four layers of material (on an off-the-shelf computer at the time). An example of a scene generated by their approach is shown in Figure 2.12.

Later, Vanek et al. [Van+11] extended the method to be able to simulate erosion on large-scale terrains. They divide the terrain into tiles of different resolution based on the complexity of the terrain. Each of the tiles is stored as a mip-map texture and different levels of detail are then used during the simulation according to the dynamics of the scene evolution. Using this approach, they achieve a 50% speedup over non-adaptive computation.

Recently, Ren et al. [Ren+18] extended the standard shallow-water flow model to work on general triangular meshes. They use a feature-based friction model and derive formulations to represent a wide range of physical effects for real-world surface flow phenomena. Their method is capable of capturing surface flow effects on complex solid surfaces.

3D Water Simulation Methods presented in Section 2.3 mostly represent earlier and less realistic attempts to simulate hydraulic erosion. With the improvements in the computational force of computers the effort to create physically based simulations prevailed but even nowadays the simulations that work with fully 3D scenes are far from being interactive. More realistic results can be achieved with more precise simulation of the fluid physics and for that reason many algorithms are based on the Navier-Stokes equations. 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.

Chapter 3

3D Fluid Simulation

This chapter describes the two main approaches to solving the Navier-Stokes equations (see Section 2.3.2 for more details) that are commonly used in computer graphics: the *Lagrangian* and the *Eulerian* approach. For a more detailed description of the fluid simulation in computer graphics the reader can refer to [Bri08].

3.1 Eulerian Approach

Eulerian approach divides the volume of the scene and tracks the fluid quantities at fixed points in the space. 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 approach is the uniform space subdivision. The regular grid representing a vast nonuniform scene will have the same resolution in flat regions as in the most detailed regions. That will result in huge amounts of data and the resulting algorithms will be very memory consuming. More advanced data structures, such as an octree, can alleviate the problem of memory consumption at the cost of more complicated algorithms.

This approach is used by Beneš et al. [Ben+06] to create a fully 3D simulation of hydraulic erosion. Their solution requires a model of the environment represented 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. Figure 3.1 shows the results of their algorithm on an example of a river meander being eroded by a huge wave.

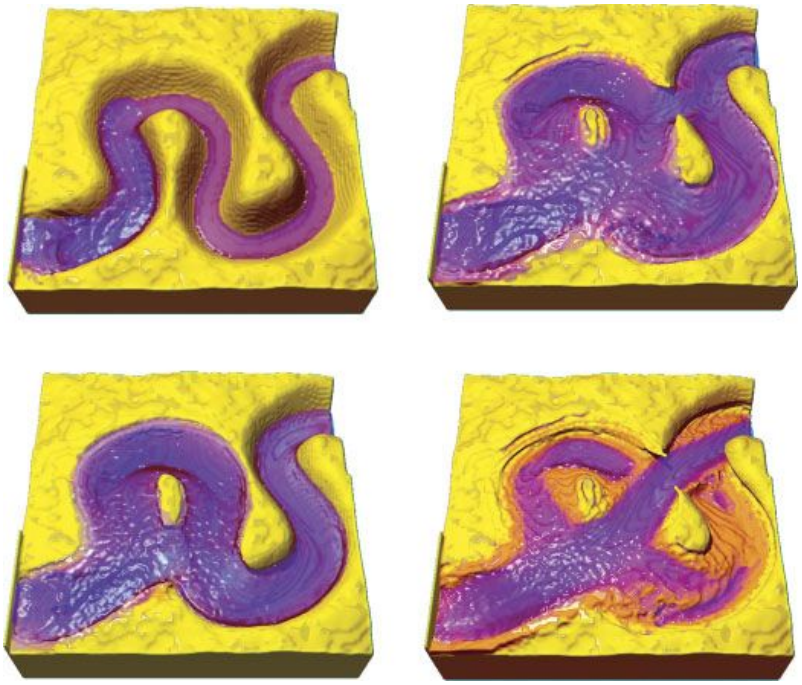


Figure 3.1: A sudden wave erodes a river meander [Ben+06]

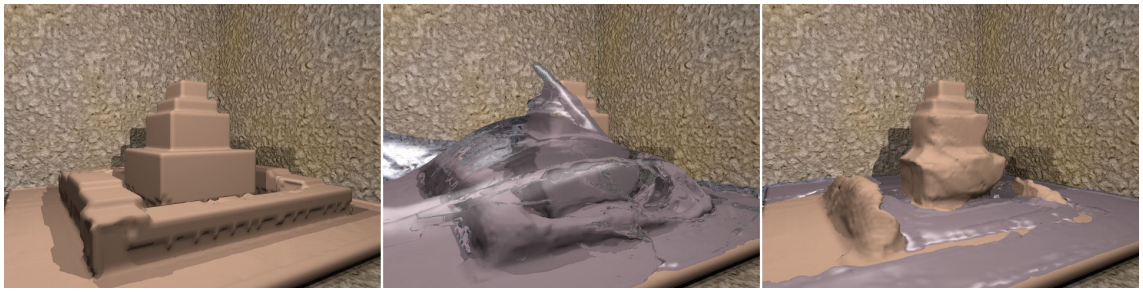


Figure 3.2: Crashing waves erode a sand castle [Woj+07]

A 3D Eulerian approach is used in a paper by Wojtan et al. [Woj+07] to simulate 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. Figure 3.2 demonstrates the results of the method on a scenario where waves erode a sand castle.

3.2 Lagrangian Approach

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 properties. This representation makes some things much simpler, e.g., the *mass conservation* condition (Equation 2.2) is automatically satisfied, provided the

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

particles do not disappear. It also addresses the memory consumption disadvantage of the Eulerian approach when working with nonuniform scenes, as the calculations are performed only in the regions where the fluid is present. Generally, the particle-based methods are less accurate than the Eulerian grid-based methods due to the difficulties in dealing with the spatial derivatives on an unstructured particle cloud but they are much faster which allows their use in real-time applications. [BS09]

3.2.1 Smoothed Particle Hydrodynamics

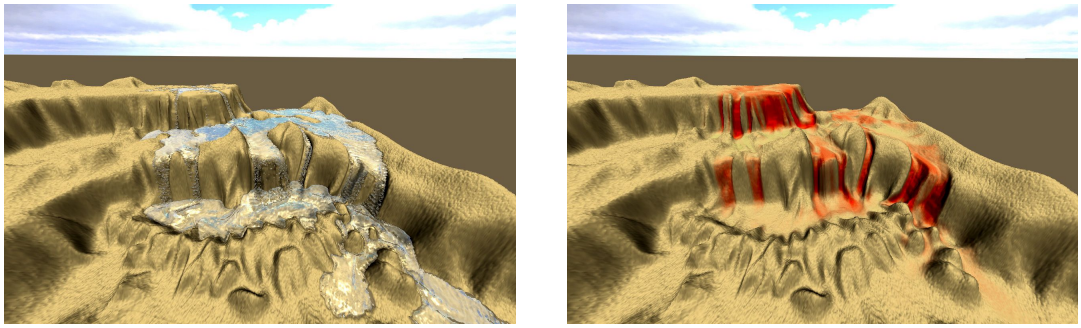
Smoothed particle hydrodynamics (SPH) is an approximate numeric solution of the Navier-Stokes equations for the fluid dynamics (Equation 2.1 and 2.2). It was developed in 1977 by Gingold and Monaghan [GM77] and independently by Lucy [Luc77]. Initially it was designed for the use in astrophysics but later 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 the Lagrangian viewpoint. The particles have a *smoothing radius* assigned, a distance over which their quantities are *smoothed* by a kernel function. Put in another way, the attributes of the particle can be calculated only with the use of all the neighboring particles within the distance defined by the smoothing radius. The contributions of each particle are weighted by their distance and their density, the weights are given by the kernel function used. There is a wide variety of kernel functions to use, including Gaussian function and the cubic spline. A more detailed description of the SPH method can be found, e.g., in [Mon92] or [DG96].

Solenthaler et al. [SSP07] proposed a unified particle model for the simulation of melting and solidification. 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.

Adams et al. [Ada+07] proposed an adaptive sampling algorithm for particle-based fluid simulation. They introduce a sampling condition allowing them to change 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 to the best of our knowledge, Krištof et al. in *Hydraulic Erosion Using Smoothed Particle Hydrodynamics* [Kri+09] were the first ones to try to couple SPH with erosion. They represent the water flow with the SPH particles that erode the terrain and transport the sediment. They define a donor-acceptor scheme that describes the advection between SPH particles and use it to 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



(a) Fluid erodes the terrain (b) Red color highlights the eroded regions

Figure 3.3: Erosion of multi-level waterfalls [Kri+09]

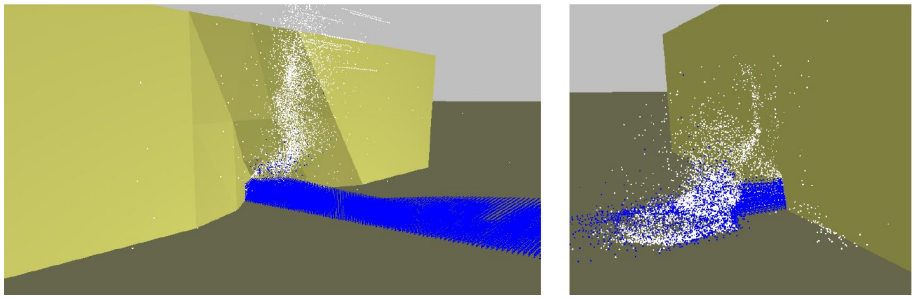
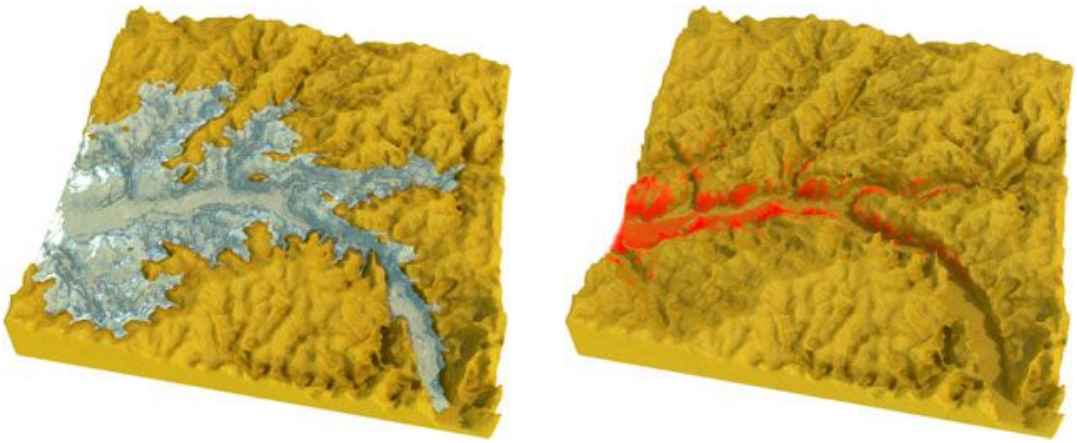


Figure 3.4: An arch created by a stream of water [Cre+14]

in the boundary particles. Figure 3.3 shows an example of an erosion simulation of multi-level waterfalls.

Later, Crespin et al. extended the method in [Cre+14]. They represent the terrain as a 3D generalized map [Lie94]. This representation allows them to simulate a fully 3D scene composed of multiple geological layers. The topological correctness of the model is achieved via a collision detection system that handles the geometric simplices through generic operations. Boundary particles are used to mediate the sediment exchange between the fluid particles and the terrain, similarly to the approach used in [Kri+09]. Figure 3.4 demonstrates the ability of the method to simulate erosion of concave structures.

Jiang et al. [JSZ18] used a particle-based approach for real-time dissolution simulation. They represent both the solid and the fluid by particles. They simulate dissolution when the local excitation energy exceeds a user-specified threshold (activation energy). Their model allows to simulate different types of materials by modifying the activation energy of the particles. They demonstrate the capability of their solution to simulate multiple phenomena, such as dissolution, melting and hydraulic erosion. Figure 3.5 shows an example of their dissolution simulation on a river bed where the running water dissolves the surface. Sediment carrying and depositing are ignored in their simulations.



(a) Fluid simulation

(b) Resulting terrain; red color marks the removed particles

Figure 3.5: Hydraulic erosion using dissolution model [JSZ18]

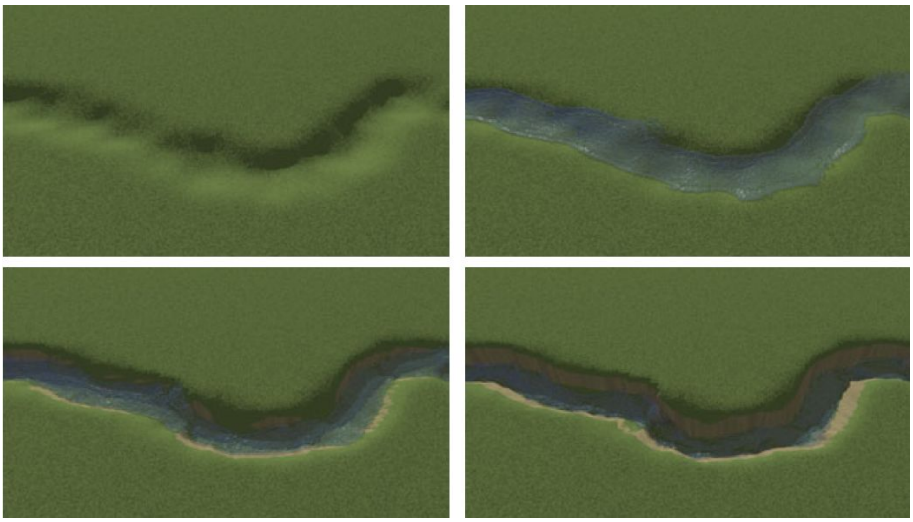


Figure 3.6: Canyon deepened by a river [Béz+10]

3.2.2 Other particle-based fluid models

Bézin et al. use a Lagrangian approach to achieve a hydraulic erosion simulation at interactive rates in [Béz+10]. They represent the terrain by a triangular mesh and use a particle-based model to simulate the fluid. When a particle collides with the mesh, erosion is applied. The erosion amount is accumulated on the vertices of the mesh over several iterations of the method before the mesh is actually adjusted. The method is implemented on the GPU to get real-time response. However, the method is not capable of working with different materials and cannot deal with complex topological changes such as merging and splitting. Figure 3.6 shows a simulation of a canyon being eroded by a river.

3.3 Semi-Lagrangian Approach

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

Chapter 4

Data Structures for Erosion Modeling

In computer graphics, many types of representation of solid objects exist. However, not all the representations are suitable for the use in erosion simulations. This chapter describes the data structures which are the most common in the erosion 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 also contain other data describing the state or the characteristics of the material. The most important simplification of the data structure is that it considers the whole column to be made of the same material with the same properties (Figure 4.1).

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 to create very fast and efficient solutions and its memory requirements are significantly better than those of the remaining data structures commonly used in terrain modeling. Considering that each element stores n bytes of data, the representation of the terrain as a grid of 1024 x 1024 elements will need only n MB to store the

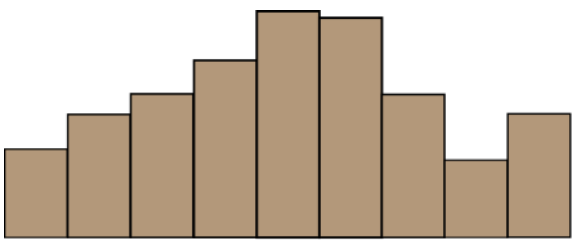


Figure 4.1: An example of height map data structure

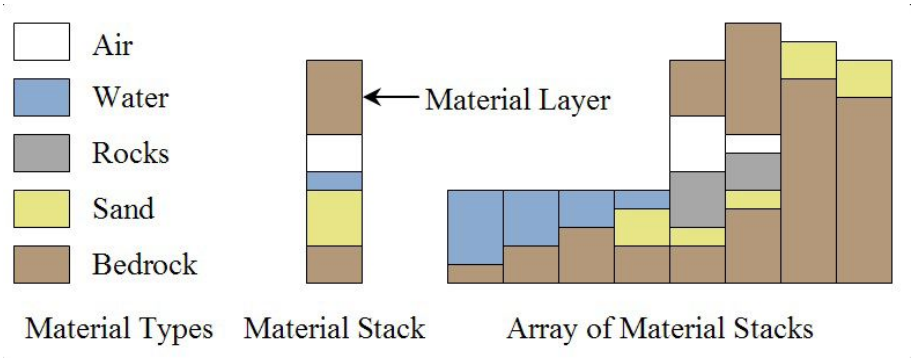


Figure 4.2: An example of layered height map [Pey+09]

terrain [BF01]. The main disadvantage of the 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].

4.2 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, terrain usually consists of several layers of materials with different characteristics. The authors integrate this idea by changing the height map structure - each element is now consisting of a one-dimensional array of elements, each one containing the information of a layer (height of the layer and the information about the material). They take advantage of the fact that the material layers are usually very thick. The height of the layer can be measured and all the information can be stored at once instead of describing every voxel. The memory requirements are similar to the ones of the height field data structure, with the difference that for each element in the 2D array k elements representing k layers are stored. To store the terrain with 1024 x 1024 elements $k \cdot n$ MB will be needed.

Peytavie et al. [Pey+09] extended the approach and created a framework that offers high level tools for authoring complex scenes. Later, Löffler et al. proposed a method that allows to render the layered height maps at interactive frame rates [LMS11].

The layered height map data structure is very useful for representing a layered terrain. It is even capable of representing a terrain with concave features if the air is considered to be another layer in the data structure (Figure 4.2). However, representing an object of general geometry could quickly degenerate to a volumetric representation.

This data structure is used, e. g., in [BF02], [MDH07] and [Cor+17].

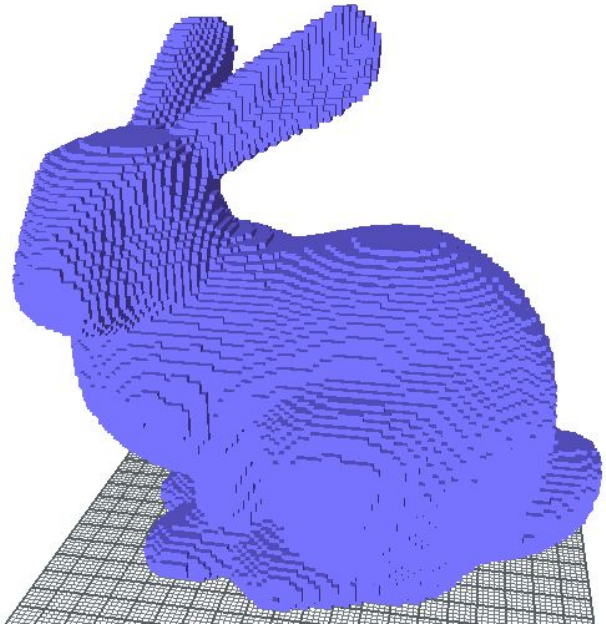


Figure 4.3: Volumetric representation[1] of the Stanford bunny model [Lev+05]

4.3 Volume Grid

Volume grids are also very popular in the field of erosion modeling. The volume representation divides the volume of the scene into voxels, 3D cubes of the same size. Each voxel then contains the information about the corresponding part of the scene. Figure 4.3 shows an example of a voxelized 3D model.

The techniques using voxel grids have high precision and give good results. Voxel grids are capable of describing any 3D structures and so they provide the means of modeling features that cannot be modeled by the previous approaches. The main drawback of this data structure is that it has high memory requirements. To model a scene with resolution 1024 x 1024 x 1024 using a voxel technique, n GB of data will be needed ([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 [Jon+10], [Dor+99] and [Bea+07].

4.4 Octree

An octree data structure is a volume grid with adaptive resolution. It is a hierarchical data structure where each cell may be subdivided if higher resolution is needed in the corresponding part of the scene. The cells are subdivided using a regular 2x2x2 subdivision scheme, dividing each cell into eight children nodes. A subdivided cell

[1]The volumetric model was created using Binvox (`http://www.cs.princeton.edu/~min/binvox/`)

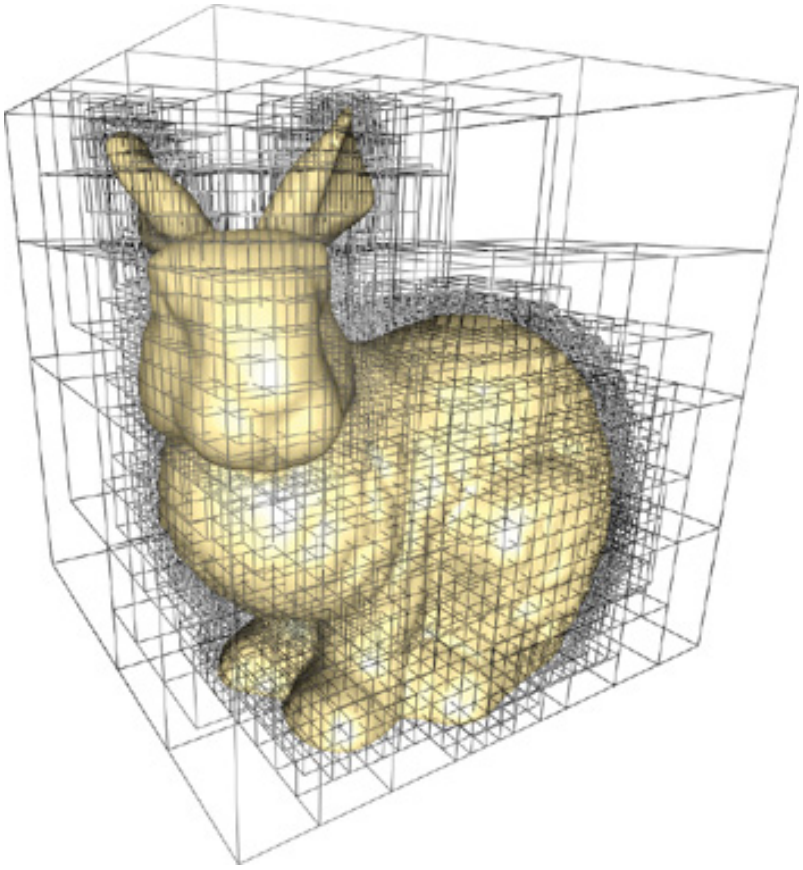


Figure 4.4: An octree representation of a 3D model [LHN05]

is called an *internal node*, a cell without children is called a *leaf*. Figure 4.4 shows a 3D model surrounded by an octree. [LHN05]

To represent an object with the same level of detail, an octree has lower memory requirements than the regular volume grids as it can increase the grid resolution only in the necessary regions. However, the increased complexity of the data structure makes the erosion simulation computations more difficult. An octree has been used, e.g., in mesh repair algorithms [Ju04] but to the best of our knowledge it has not been used for erosion simulation. It is included in the overview of the data structures for the sake of completeness.

4.5 Triangular Mesh

Another data structure which can be used for erosion modeling is a triangular mesh. It represents an object as a set of vertices connected by edges. Three vertices connected by edges represent a triangular face, the set of all faces represents the surface of the object. An example of a triangular mesh model is captured in Figure 4.5.

One of the advantages of this structure 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

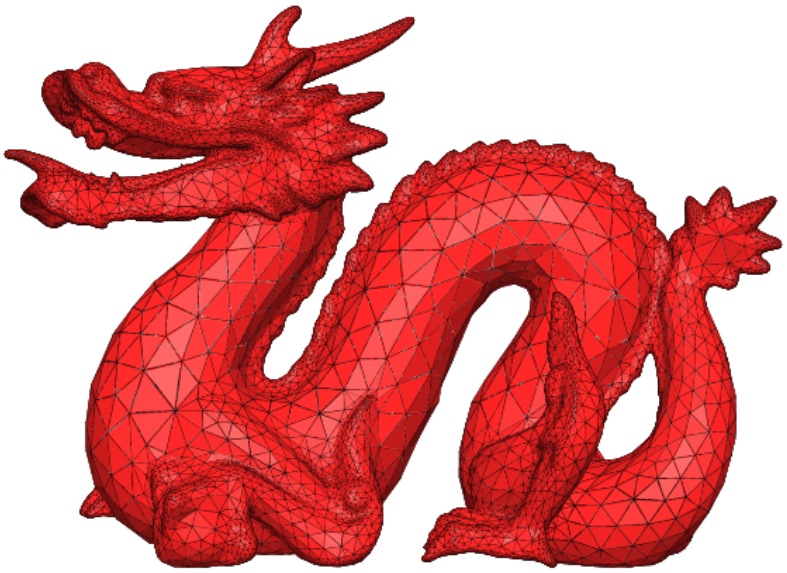


Figure 4.5: Stanford dragon triangular mesh model [Si14]

triangular mesh is that its resolution can change throughout the scene, allowing the creation of a very detailed mesh in the important areas while the flat unimportant regions are modeled with lower resolution.

The use of the triangular meshes for the simulation of erosion also has disadvantages. When an object is eroded, its surface changes. When simulating such a phenomena using triangular meshes, the vertices of the mesh have to be moved to simulate the surface change. By changing the position of a vertex, an inconsistency can be created if the vertex penetrates the mesh. This inconsistency has to be dealt with before the erosion simulation can continue. The topic of mesh repair will be discussed in more detail in Section 5.

Purchart in [Pur09] uses an adaptive triangular 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 not supported.

4.6 Tetrahedral Mesh

Tetrahedral mesh is a volume variation of the triangular mesh data structure. The model is not represented only by its surface, the whole volume is stored as a set of tetrahedra. Figure 4.6 shows an example of the Stanford dragon model [Lev+05] converted to a tetrahedra representation.

Tetrahedral mesh loses the rendering advantage of the triangular mesh as its surface has to be extracted in order to render the tetrahedronized model. However, in some applications the volume approach can be more desirable as it can give us additional information about the topology of the object. The tetrahedral data structure is used, e.g., in [TJ10].

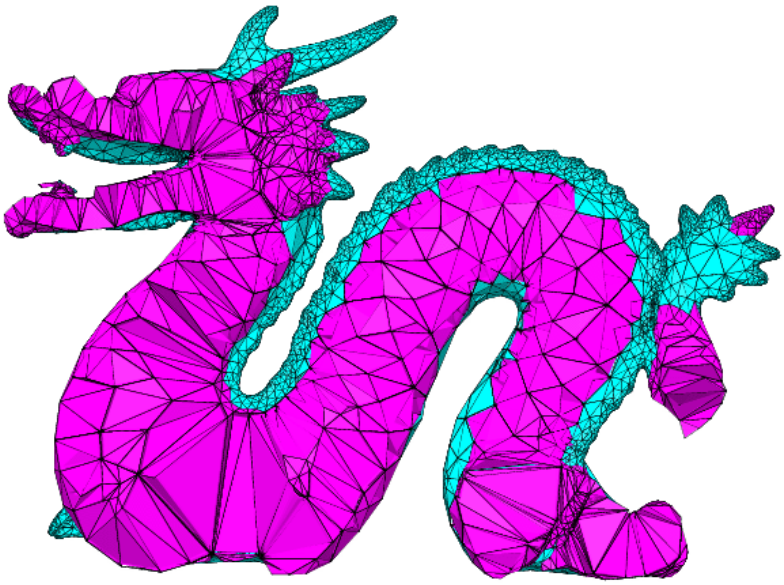


Figure 4.6: Tetrahedral representation of the Stanford dragon model [Si14]

Chapter 5

Repair of Intersecting Meshes

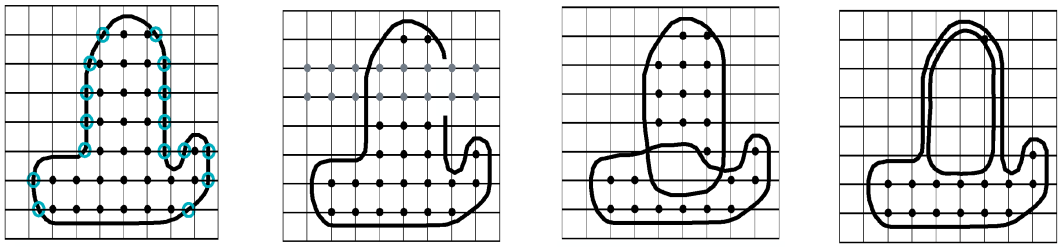
Erosion simulation of geometric objects represented as triangular meshes can lead to the creation of a mesh inconsistency. If the mesh is heavily eroded, it is possible for two distinct parts of the mesh to overlap, creating a mesh self-intersection. The self-intersection corrupts the topology correctness of the mesh and has to be repaired before the erosion simulation can continue.

Self-intersections and mesh-to-mesh intersections are mesh defects that are very common and as such many methods exist that address the problem. The approaches can be divided into two main categories: *global* (*volumetric*) and *local* (*surface oriented*) approaches [ACK13].

5.1 Global Approaches

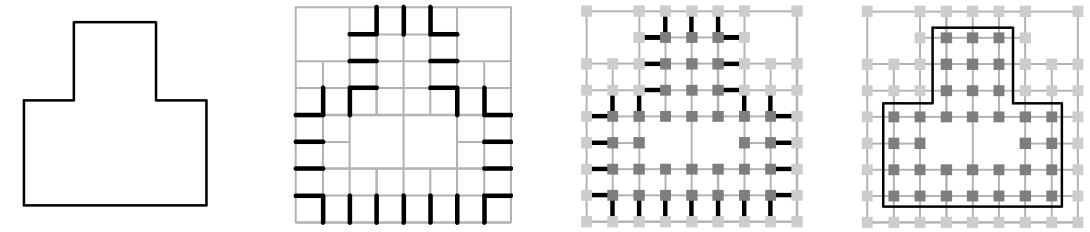
Global approaches convert the polygonal mesh to an intermediate representation which is then used to generate a new valid mesh. Early methods utilize volumetric representation [ABA02]. They convert the mesh to a regular voxel grid where each voxel either describes the signed distance from the surface or contains a binary (solid/empty) classification. The output polygon mesh is generated using a surface extraction method. A representative of this approach is the method proposed by Nooruddin and Turk [NT03]. They use parity count to decide whether a voxel is interior or exterior (Figure 5.1a, 5.1b). The parity count may fail if the mesh contains self-intersections (Figure 5.1c) or thin features (Figure 5.1d). The authors propose a ray stabbing method to address this issue. The method casts a ray through the mesh and looks for the closest and the farthest intersection. The voxel is then classified as an interior if it lies between the two intersection points, otherwise it is classified as an exterior. For a single ray, some voxels may be misclassified as interior voxels. Using multiple rays in different directions decreases the risk of a mistake. Marching cubes algorithm is then used to extract the valid explicit surface.

While being effective at repairing the defects, this approach is sensitive to the resolution of the intermediate structure. Low resolution in detailed areas can cause excessive smoothing and a loss of details. The alteration of the mesh can also lead



(a) A closed model (b) A model with a hole (c) A model with intersecting parts (d) A double-walled model

Figure 5.1: Classification of voxels as interior or exterior [NT03]



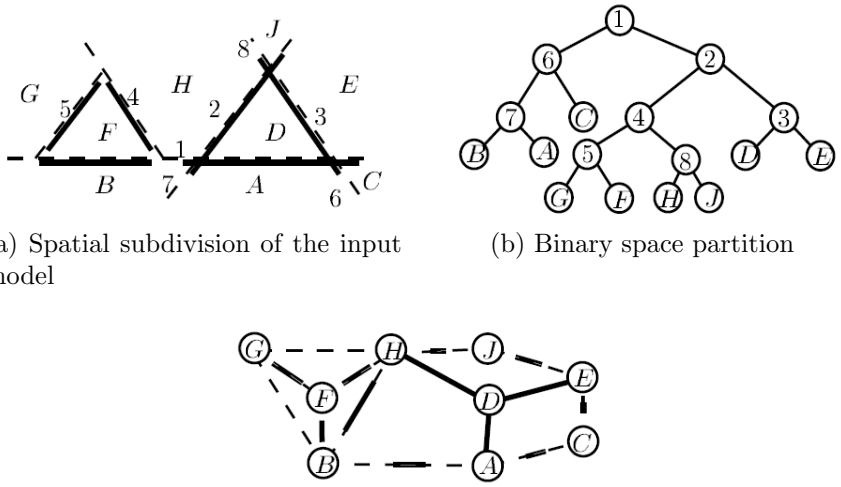
(a) Original model (b) Intersection edges (c) Signs generation (d) Model reconstruction

Figure 5.2: Octree-based model repair [Ju04]

to a mass loss. Adaptive approaches exist that change the resolution of the intermediate structure according to the required detail. Ju [Ju04] proposed a technique with the use of an adaptive octree. He takes the original model (Figure 5.2a) and marks the edges of the grid that intersect the mesh polygons (Figure 5.2b). Afterward he uses this information to generate signs at the grid points, so that each intersecting edge exhibits a sign change (Figure 5.2c). A contouring method is then used to reconstruct the output mesh (Figure 5.2d). The method is capable of preserving details and sharp features, however, thin features are removed by the method. A similar approach is used by Bischoff et al. [BPK05], where morphological closing operations are used to repair holes and gaps present in the input.

Binary space partition (BSP) trees have been also used to solve this problem [MF97]. An input polygon mesh is converted to a BSP tree by using the supporting planes of the input polygons as the splitting planes (Figure 5.3a). The leaves of the tree correspond to closed convex spatial regions (Figure 5.3b). Cell adjacency graph (Figure 5.3c) is constructed and used to determine whether a cell is solid or not. However, the output mesh may contain singular edges and vertices.

Level set methods [OF02] represent the interface as the zero level of an implicit function. A typical function used is the signed distance from the surface mesh discretized over the volume. The new intersection-free explicit surface is then found as the zero level set of the function. Level set methods may cause an excessive smoothing and a loss of features. These problems can partially be solved by using tracker particles to rebuild the level set in under-resolved regions [Enr+02].



(a) Spatial subdivision of the input model

(b) Binary space partition

(c) Cell adjacency graph

Figure 5.3: Binary space partition example [MF97]

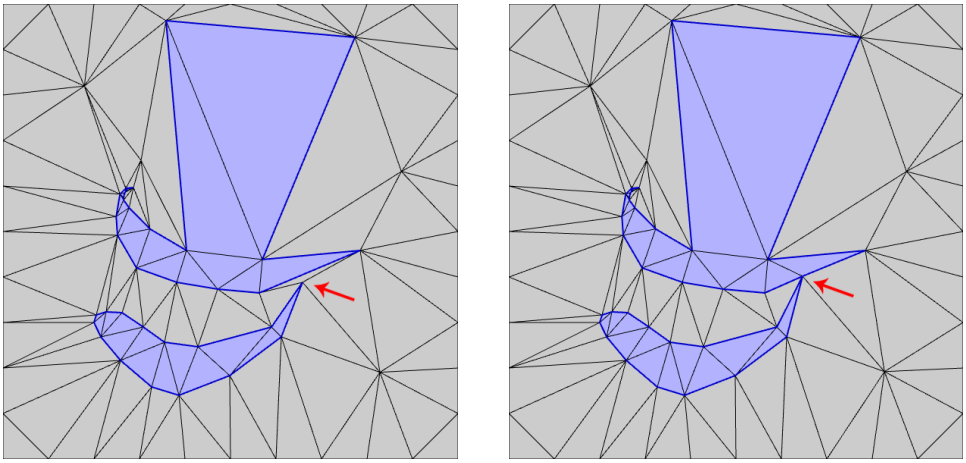
Deformable simplicial complexes (DSC) [MB12] method addresses the problems of the level set method. It represents the interface explicitly as a triangular mesh in 2D or a tetrahedral mesh in 3D. The surface is evolved by moving the mesh vertices. If the vertex displacement causes the creation of a degenerate triangle or tetrahedron, the vertex is moved as far as possible and mesh improvement operations are applied (Figure 5.4). The topology changes are handled automatically [Chr+14].

The use of the global approaches is appropriate if the mesh is highly inconsistent. It typically allows to create very robust methods at the cost of lower accuracy and efficiency.

5.2 Local Approaches

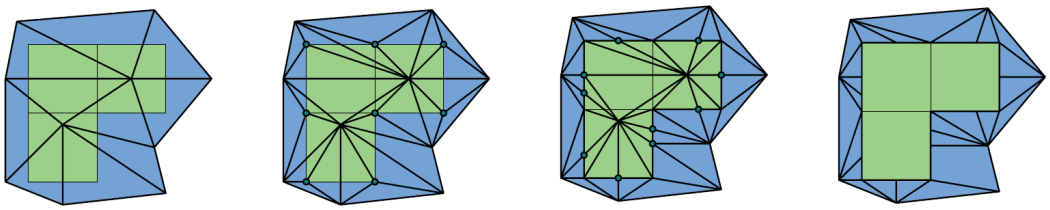
Local approaches work directly with the input mesh and identify individual self-intersections which are then repaired locally, usually one at a time, and leave the rest of the mesh unchanged. These approaches change the input mesh as little as possible which is desirable in applications where the accuracy of the solution is the main interest. The local approaches are more suitable when the intersections are located only at isolated parts of the mesh.

Some of the local approaches take a global method and apply it only in the local scale. Bischoff and Kobbelt [BK05] use a local voxel grid to repair the mesh. They locate the voxels containing the defects and only those voxels are modified. They subdivide the triangles intersecting the critical voxels so that each triangle lies completely outside or completely inside (Figure 5.5b, 5.5c). Then the interior triangles are discarded (Figure 5.5d) and the resulting holes are filled. Attene uses a similar approach but restricts the set of input meshes to raw digitized meshes in [Att10]. The self-intersecting faces are assumed to be small, so they are discarded and the



(a) Before topology change (b) After topology change

Figure 5.4: 2D example of deformable simplicial complexes, exterior triangles are light grey, interior blue [MB12]



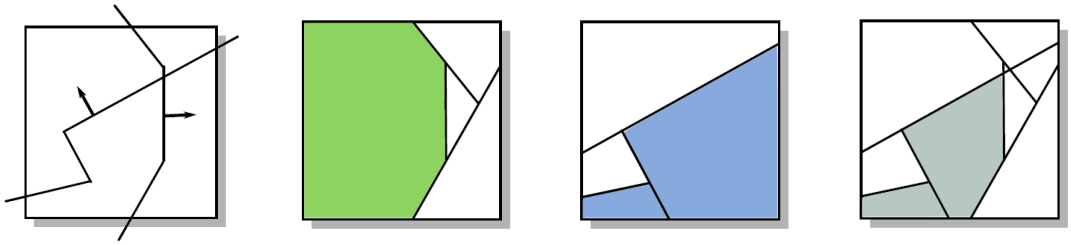
(a) Initial configuration (b) 1-to-3 triangle splits (c) 2-to-4 triangle splits (d) Interior triangles are discarded

Figure 5.5: Mesh repair using a local voxel grid [BK05]

generated holes are fixed. The method can fail in some cases but for the selected class of models it usually succeeds. Wojtan creates a signed distance field around the mesh and checks for possible intersections in [Woj+09]. New parts of the mesh are then generated using the marching cubes method in the cells containing the intersections. Afterward, the original mesh and the newly generated parts are stitched together. These methods depend highly on the grid resolution and can also lead to changes in the volume encompassed by the mesh.

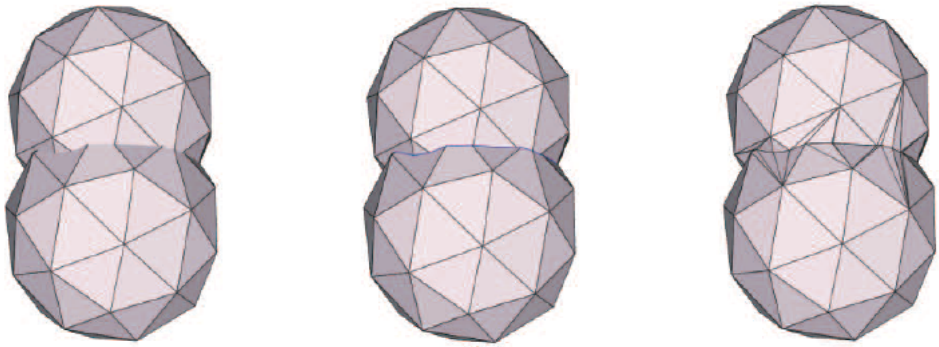
The problem of solving an intersecting geometry is complicated due to the finite precision of calculations which can cause an incorrect result of a predicate test, leading to unpredicted behavior. Campen and Kobbelt avoid this problem by using an adaptive octree data structure combined with a plane-based BSP representation [CK10]. The computations of the intersections are restricted only to those cells of the octree where intersections actually occur. Within these cells, the input polygons are converted to a plane-based BSP representation (Figure 5.6). The actual calculations of the intersections are performed during the generation of the resulting mesh.

Other group of methods does not rely on the use of an intermediate data structure but works directly with the geometry of the mesh. Zaharescu et al. search the mesh



(a) Two input intersecting surfaces (b) BSP-rep of the first surface (c) BSP-rep of the second surface (d) Merged BSP

Figure 5.6: 2D example of BSP-based local mesh repair [CK10]



(a) Input meshes (b) Triangle intersections (c) Output manifold mesh

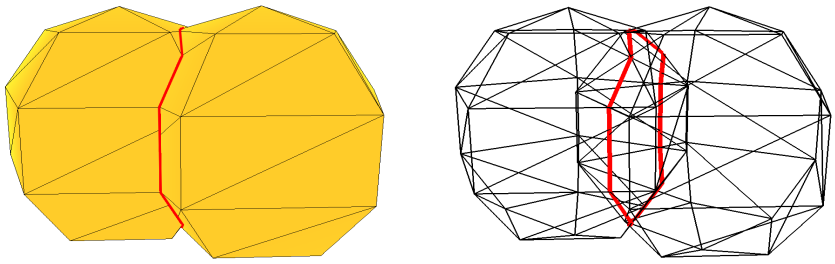
Figure 5.7: Local geometric mesh repair [ZBH11]

for valid and partially valid faces in [ZBH11]. Valid faces do not contain intersections and should be preserved, while partially valid faces contain intersections and only a part of them should be kept in the resulting intersection-free mesh. An example of the method is shown in Figure 5.7. Two spheres (Figure 5.7a) intersect at the intersection boundary captured in Figure 5.7b. These partially valid triangles are triangulated using constrained 2D Delaunay and the interior triangles are discarded (Figure 5.7c). Finally, the two parts of the mesh are stitched together. Exact arithmetic is used to achieve numerical stability of the intersection calculations, boundary cases are avoided by using virtual perturbation method.

Brochu and Bridson [BB09] also use a local stitching method to avoid expensive computations. They restrict the input data to dynamic polygon meshes and take advantage of the motion data. A possible collision is detected before it actually takes place, when the two parts of the mesh get close together. The proximate faces are identified and discarded and the resulting holes are stitched together. A similar approach is used in [Cla+13] to simulate interactions between solids and liquids.

5.2.1 Neighbor Tracing Method

Lo and Wang use neighbor tracing method (tracing the neighbors of intersecting triangles - TNOIT) to speed up the process of finding the intersection boundary and



(a) Intersecting models (b) Intersection boundary

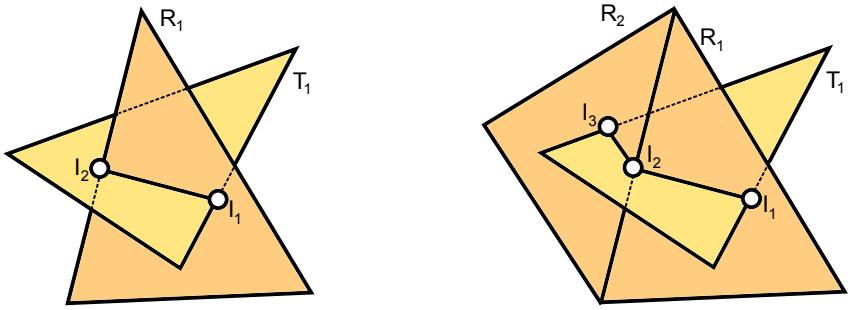
Figure 5.8: Two low polygon models and their common intersection boundary

to achieve reliable results in [LW04]. The intersection boundary is the boundary where the two meshes meet and an inconsistency is created. The boundary is formed by a set of connected line segments and each pair of intersecting triangles adds one segment to this set. The intersection boundary does not have to be planar, as it precisely follows the shape of the intersecting meshes. Figure 5.8 captures two low polygon sphere models and their common intersection boundary.

To find the intersection boundary, they first identify the intersecting triangles. A naive approach would check each pair of triangles for a possible intersection, leading to quadratic complexity with respect to the number of mesh faces. This can be alleviated by adding an additional spatial subdivision technique. A regular grid is used, where each cell contains information about triangles which are located inside. The search for possible intersections is then performed only in the cells the tested triangle intersects.

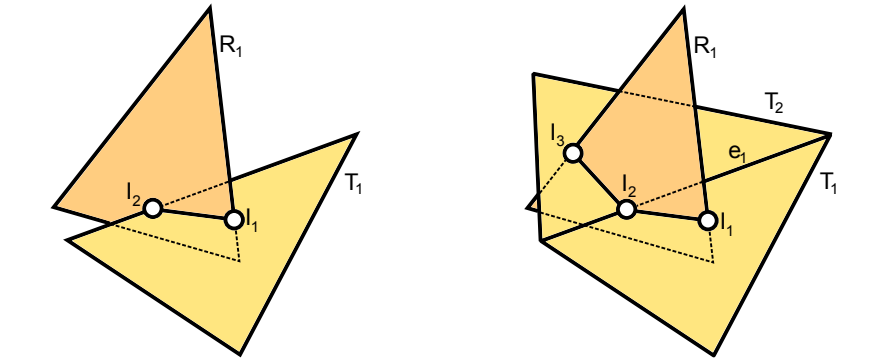
Once the first intersected triangle is located, the evaluation can be sped up by tracing over its neighbors. The TNOIT algorithm takes advantage of the mesh connectivity and traces the intersection boundary across the intersected edges of the intersecting triangles. Considering intersecting triangles T_1 and R_1, the intersection between them forms a polygonal region if the triangles are coplanar, otherwise it forms a line segment I_1I_2 where I_1 and I_2 are the segment endpoints. To trace the boundary correctly, the relative position of the segment endpoint I_2 inside the triangle T_1 has to be determined. The segment endpoint I_1 is coincident with the segment endpoint I_2' from the preceding step of the algorithm and so it is irrelevant for the neighbor tracing. There are four cases how the triangle T_1 can intersect the triangle R_1 [LW04]:

1. the intersection segment ends inside the triangle T_1 (Figure 5.9),
2. the intersection segment ends on an edge of the triangle T_1 (Figure 5.10),
3. the intersection segment ends at a vertex of the triangle T_1 (Figure 5.11), and
4. both T_1 and R_1 lie in the same plane, the intersection is a polygonal region (Figure 5.12).



(a) Intersection between T_1 and R_1 (b) Next step in boundary tracing

Figure 5.9: Intersection inside a triangle



(a) Intersection between T_1 and R_1 (b) Next step in boundary tracing

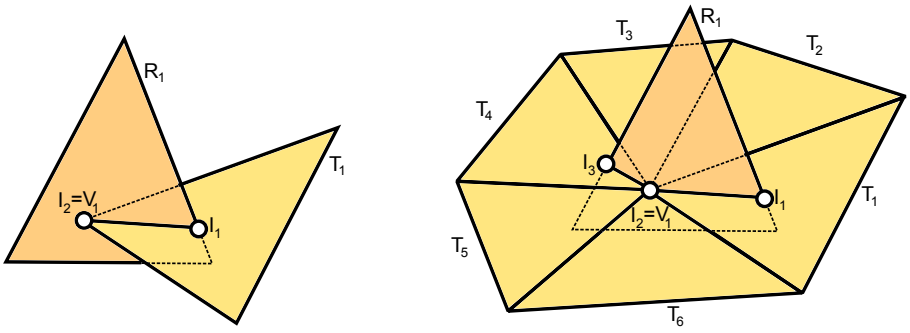
Figure 5.10: Intersection on an edge

An appropriate action has to be taken for each case so that the boundary is traced correctly.

Intersection inside a triangle If the intersection segment between T_1 and R_1 lies within the triangle T_1, the neighbor tracing is not needed. The endpoint I_2 is located inside the triangle T_1 (Figure 5.9a), so the next intersection segment has to start inside T_1 (Figure 5.9b).

Intersection on an edge If the intersection segment between T_1 and R_1 ends on the edge e_1 of the triangle T_1 (Figure 5.10a), the neighbor has to be traced across the edge e_1 to the triangle T_2 as shown in Figure 5.10b.

Intersection at a vertex In the rare case when the intersection segment ends at a vertex V_1 of the triangle T_1 (Figure 5.11a), all triangles that share the vertex V_1 need to be considered as possible neighbors for the intersection search. This situation is captured in Figure 5.11b where all triangles from T_2 to T_6 have to be checked for intersection with the triangle R_1 in order to find the next intersection segment. No specific order of the search is required and the boundary tracing can be reinstated as soon as the intersected triangle (T_4 in Figure 5.11b) is found.



(a) Intersection between T_1 and R_1 (b) Next step in boundary tracing

Figure 5.11: Intersection at a vertex

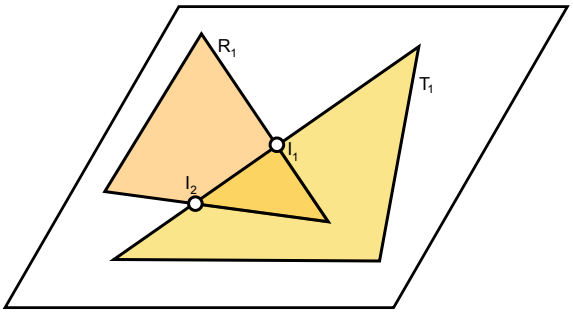
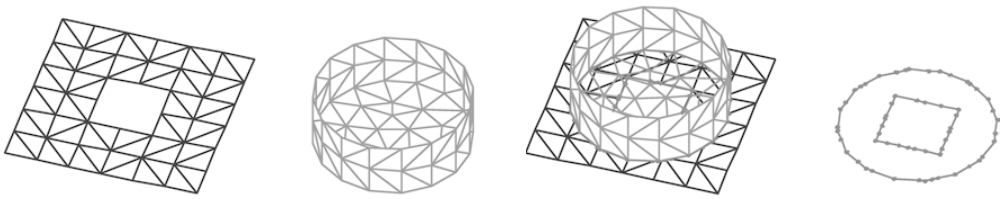


Figure 5.12: Intersection of triangles lying in the same plane

Intersecting triangles in the same plane If the triangles T_1 and R_1 lie in the same plane, in most cases their intersection does not have to be calculated unless we are working with open surfaces and the boundary edge participates in the intersection (Figure 5.12). An example of the intersection of planar surfaces is shown in Figure 5.13. A planar surface with a hole (Figure 5.13a) intersects with a cylindrical surface with a closed bottom (Figure 5.13b) as in Figure 5.13c, the intersection forms two closed loops (Figure 5.13d).



(a) Planar surface with a hole (b) Cylindrical surface (c) Intersecting meshes (d) Intersection loops

Figure 5.13: Intersection of planar surfaces [LW04]

Chapter 6

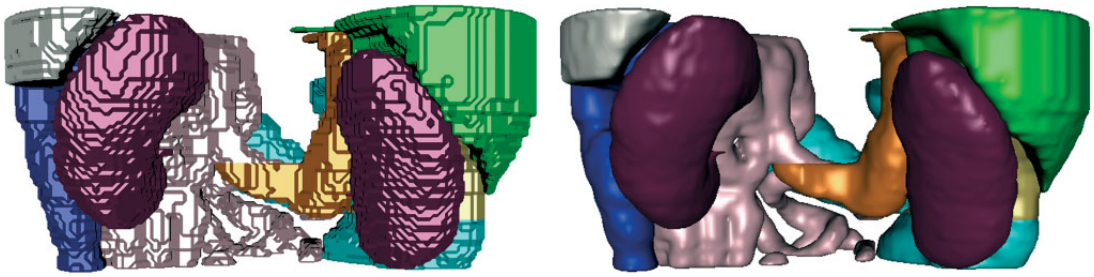
Materials

Materials are an important part of the erosion simulation. Real-life erosion scenes are usually formed of multiple materials and so a reliable means of material definition is needed.

Most of the work on domains with multiple materials focus on extracting a correct and consistent mesh for each of the materials present in the volumetric data obtained, e.g., from a medical scan. Wu and Sullivan [WS03] enhanced the marching cubes algorithm to reconstruct multiple material meshes. Their algorithm extracts boundary surfaces between different materials in a single sweep of the input data. They ensure the continuity and integrity of the resulting surfaces. An example of the reconstructed surface mesh is shown in Figure 6.1.

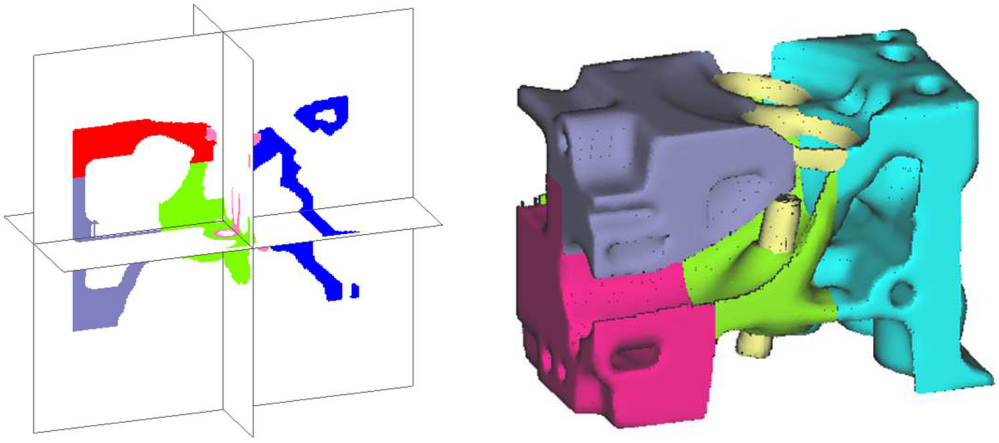
Zhang et al. [ZHB10] generate unstructured tetrahedral and hexahedral meshes using an octree-based isocontouring method. They introduce the notion of a *material change edge* which is used to identify the interface between two or several different materials. They propose a method to calculate the *minimizer point* for a cell shared by more than two different materials. They improve the quality of the resulting meshes in the postprocessing step.

Wang [Wan11] generates the mesh surfaces using a ray representation of a solid as an intermediate structure. At first, the algorithm converts the multi-material volumetric data to a ray representation. Then a filtering algorithm is used to be able



(a) Extracted material surface (b) Smoothed material surface

Figure 6.1: An example of reconstructed surface mesh of kidney region [WS03]



(a) Input multi-material volumetric data (b) Generated ray representation

Figure 6.2: An example of surface mesh generation from a ray representation [Wan11]

to process the rays using a parallel approach. Lastly, mesh surfaces are generated using a method based on dual-contouring.

Bronson et al. [BLW14] propose a method for generating multi-material tetrahedral meshes from input volumetric data. Their method can support any number of materials and produces tetrahedral meshes with guarantees on geometric fidelity and upper and lower bounds of dihedral angles. Figure 6.3 shows a tetrahedral mesh generated by their approach from a segmented MRI scan of a human head.

Faraj et al. [FTB16] work with multi-material tetrahedral meshes and propose a method for iterative remeshing. Their feature-aware approach is based on simple local topological operations. They are capable of generating high quality meshes at a user-defined resolution with the important model features preserved.

These approaches are suitable for static scenes, where the domains are strictly separated. For dynamic scenes or scenes, where the individual domains are blending into each other, the aforementioned approaches are often inappropriate. An example of such a scene could be an erosion scenario, where sand and pebbles of various size mix up to form a river bank that is being eroded by flowing water. Such a scene could be described using a volumetric approach similar to the one used by Benes et al. in [Ben+06]. The volumetric representation is very memory consuming, a layered data representation introduced by Benes and Forsbach [BF01] can be used instead to alleviate the problem. The layered data structure is a sufficient description of a terrain scene consisting of several layers of material, but for a general scene with gradually changing material, it converges back to the volumetric representation.

A different approach is used by Tychonievich and Jones in [TJ10], where a Delaunay deformable model is used to represent the eroded terrain and the material is defined for each cell of the Delaunay triangulation. A new mesh is generated every iteration of the method and the material properties have to be reconstructed using the resemblance of the two meshes.

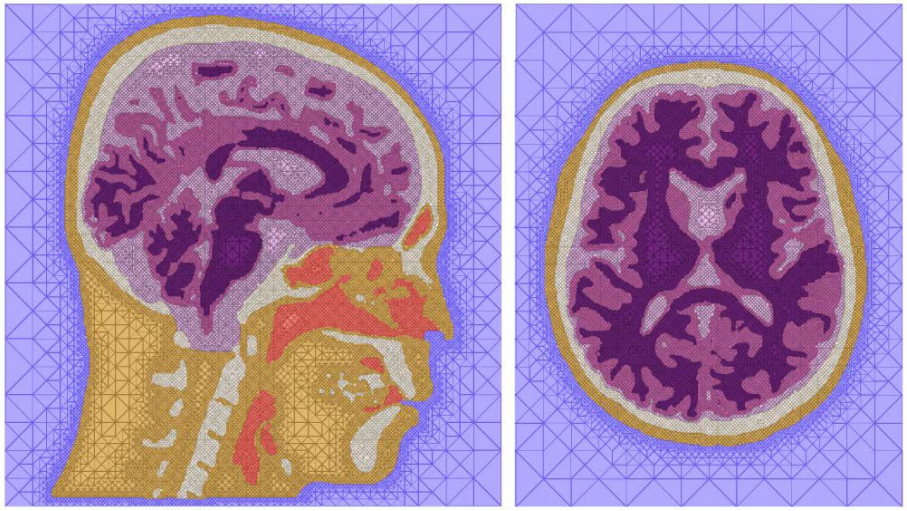


Figure 6.3: Tetrahedral meshes generated from MRI scan of a human head [BLW14]

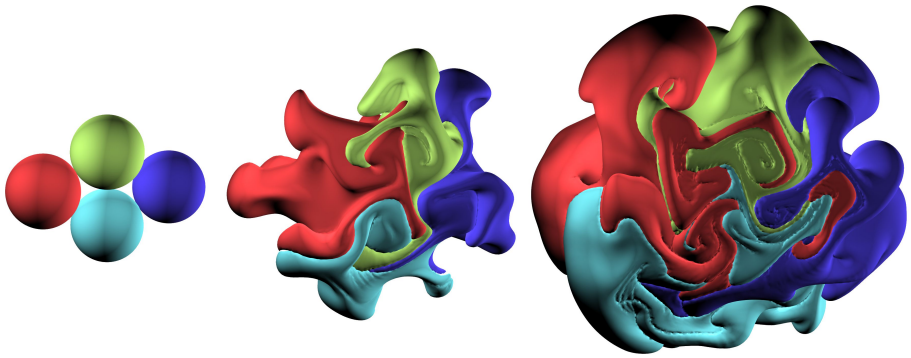


Figure 6.4: Alternating steps of normal flow and curl noise applied to four spheres [DBG14]

Da et al. propose a method for tracking the evolution of multi-material interfaces represented by non-manifold triangular meshes [DBG14]. The material labels are assigned to each half-face to be able to distinguish volumetric regions. They build on previous work in the area and address the non-manifold mesh operations, such as merging of similar materials, splitting, mesh improvement and merging of different materials. Their approach deforms the meshes and produces watertight non-intersecting outputs. Figure 6.4 shows the results of a simulation where alternating steps of outward normal flow and the curl noise were applied to four spheres composed of different materials.

Jiang et al. represent an object by particles in their method for dissolution simulation [JSZ18]. Different types of material can be simulated by lowering or increasing the activation energy of the particles. The activation energy affects the forces that are keeping the particles representing the solid object together; the lower the activation energy, the faster the dissolution will be. Figure 6.5 shows an example of a dissolution-based erosion on a layered terrain represented by particles with varying activation energy.

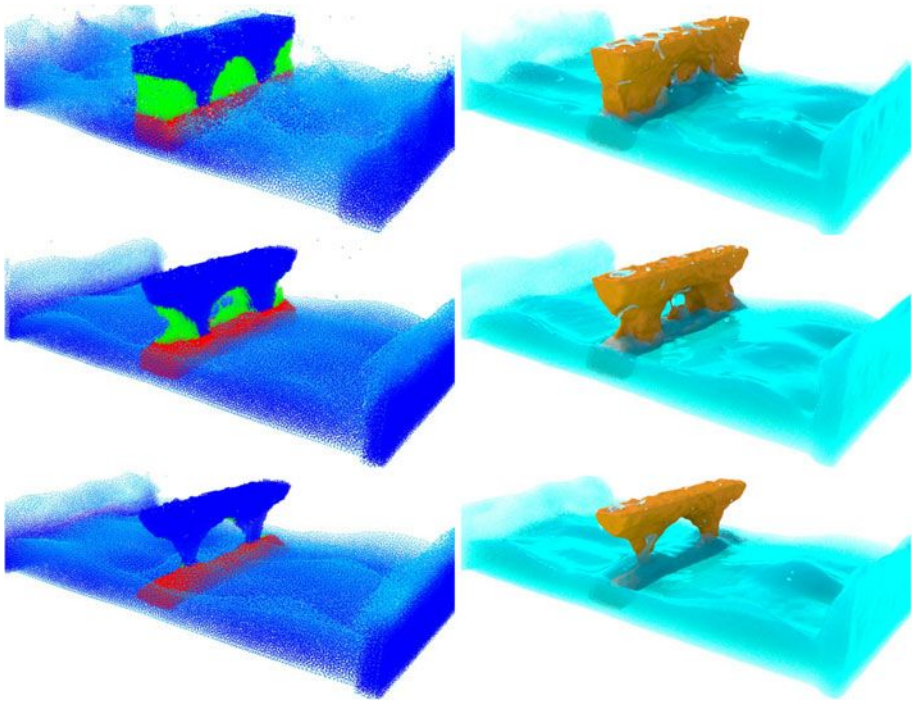


Figure 6.5: Dissolution simulation on layered terrain with three different material layers, each with different activation energy [JSZ18]

Chapter 7

Contributions

We propose a novel solution to the hydraulic erosion and weathering simulation problem. We represent the eroded object as a triangular mesh. This representation is particularly useful as it can describe fully 3D scenes, such as terrains with caves and tunnels, with lower memory requirements than the commonly used volumetric approaches. However, the use of a triangular mesh data structure brings certain pitfalls. During erosion simulation, an inconsistency can be created in the mesh if two parts of the mesh intersect as a result of the mesh alteration. The mesh inconsistency has to be repaired before the erosion simulation can proceed. Another important aspect of erosion simulation is the modeling of different materials, allowing us to simulate erosion of a non-homogenous scene.

The remainder of the thesis describes our contribution in more detail:

Chapter 8 describes our solution to the hydraulic erosion problem. The proposed solution [SKB15] represents the terrain using a triangular mesh and simulates the fluid using a particle-based approach. The solution allows for adaptive changes of the mesh resolution according to the local complexity of the terrain, which leads to lower memory requirements when compared to volumetric approaches. The use of triangular mesh data structure also supports the visualization of concave 3D features, allowing the simulation and visualization of erosion on terrain elements such as tunnels or caves.

Chapter 9 elaborates on this topic further and describes a more advanced erosion simulation approach [SKV19] that can be used to simulate erosion on terrains as well as on smaller individual objects. The method extends the erosion simulation by taking into account the local shape of the eroded object using mean curvature values. The method represents a simple unified approach for both weathering and hydraulic erosion simulation on triangular meshes.

Chapter 10 addresses the problem of repair of self-intersections and mesh-to-mesh intersections that can be created during the erosion simulation. The proposed method [SKB18] uses a local geometry-based approach to repair intersecting meshes accurately without the need to manipulate the input data or to employ arbitrary precision arithmetic. The solution is obtained through a careful classification of the cases that could result from a numerical imprecision of the floating point arithmetic.

Chapter 11 presents several possible approaches to multiple material modeling for the use in erosion simulation. The proposed approach [SK16] uses binary space partitioning (BSP) to simulate complex multi-material scenes. The approach allows the definition of a nontrivial scene composed of several materials, including the definition of a gradually changing material. A method for an automated creation of the BSP tree from input volumetric data is proposed to simplify the use of the method in complex scenarios.

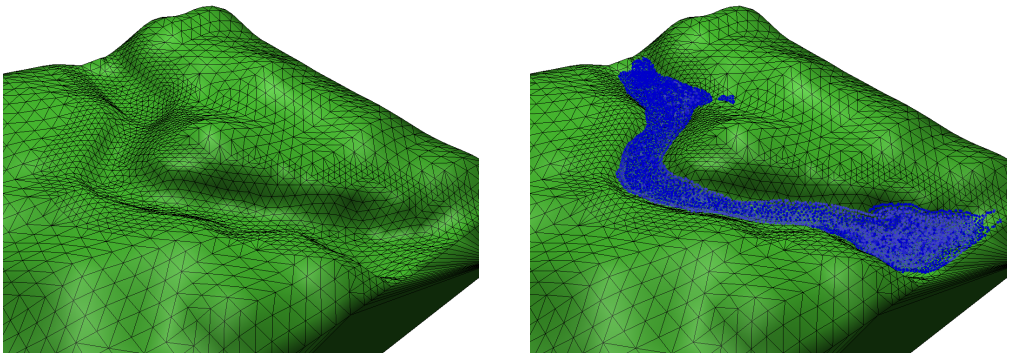
Chapter 12 shows that the erosion principles can be applied to other real-life problems. The proposed approach [Sko+17] takes inspiration in the erosion processes and applies them to the problem of aging simulation. The proposed method simulates aging of models created by deformation-based modeling and requires no training data. A user defines the position of wrinkles by selecting the position of endpoints of the desired wrinkles and the wrinkles are then generated using an erosion-inspired approach.

Chapter 8

Hydraulic Erosion Modeling on a Triangular Mesh

We propose a novel method for hydraulic erosion simulation that is using a combination of particle system fluid simulation coupled with terrains represented as triangular meshes. We simulate the fluid with the Smoothed particle hydrodynamics (SPH) particles [GM77], using an approach similar to the one used by Krištof et al. [Kri+09]. The SPH defines the way the properties of the particles, such as pressure, speed and direction, are computed. The pressure is calculated based on the density of the particles and determines the speed and direction of the movement of the particles. The main advantage of the method lies in its adaptability - the particles are restricted to the areas containing the fluid. The computation is then performed only in the regions where the fluid is present.

The terrain is represented as a surface triangular mesh in order to simulate the erosion of elements such as tunnels or caves. An example is shown in Figure 8.1. The model of a river bed has higher resolution in the areas where the erosion will be applied (Figure 8.1a).



(a) The original model has higher resolution in the erosion areas

(b) The fluid represented by the SPH particles

Figure 8.1: An example of the triangular mesh model in erosion simulation

Every iteration of the proposed method consists of the following steps:

1. The properties of the SPH particles, such as pressure, speed and direction, are calculated.

2. The fluid-terrain interaction is computed.

3. Each particle moves in the calculated direction.

4. The erosion/deposition sediment exchange is calculated for the particles colliding with the terrain.

5. The triangular mesh is updated according to the amount of sediment eroded/deposited at the vertices.

8.1 Fluid-Terrain Interaction

The motion of a particle is defined by the particle properties, by gravity, and by the surrounding terrain. The fluid-terrain interaction is a vital part of the erosion simulation. A particle is influenced by all the mesh faces within the reach of its radius. To speed up the search for the nearby faces, a regular spatial subdivision is used. Using this auxiliary data structure, we do not have to search for the faces in the whole mesh, but only in the cells that fall into the radius of the particle. Each face f in the selected cells is checked for an interaction with the particle p as follows:

1. Line l is constructed that goes through the position of the particle with the direction of the normal vector of the face f.

2. If the line l intersects the face f, the orthogonal distance d between the face f and the particle p is calculated.

3. If the distance d is smaller than the radius of the particle, the face f influences the motion of the particle p.

The face f contributes to the force affecting the particle with a force $\mathbf{f}_p[Pa]$ calculated using the penalty-force method [Ama06]:

$$\mathbf{f}_p = (k_s d + k_d(\mathbf{v} \cdot \mathbf{n}))\mathbf{n}, \tag{8.1}$$

where $k_s\ [Pa \cdot m^{-1}]$ is the penalty force stiffness, $k_d\ [Pa \cdot s \cdot m^{-1}]$ is the damping coefficient, $d\,[m]$ is the penetrated distance, $\mathbf{v}\ [m \cdot s^{-1}]$ is the velocity of the particle and $\mathbf{n}$ is the normal vector of the face.

8.2 Erosion and Deposition

When a particle collides with the terrain, we simulate the erosion of the surface or the deposition of the sediment, depending on the velocity and the direction of the particle. We calculate the erosion rate using a method similar to the one presented in the work of Wojtan et al. [Woj+07]. The erosion rate $\boldsymbol{\varepsilon}\,[m \cdot s^{-1}]$ is calculated using the equation by [Par65]:

$$\boldsymbol{\varepsilon} = k(\boldsymbol{\tau} - \boldsymbol{\tau}_c)^a, \tag{8.2}$$

where $k\,[m^3 \cdot N^{-1} \cdot s^{-1}]$ is the erosion coefficient, $\boldsymbol{\tau}\,[Pa]$ is the shear stress, $\boldsymbol{\tau}_c\,[Pa]$ is the critical shear stress and a is a constant usually set to 1. The critical shear stress $\boldsymbol{\tau}_c$ is a threshold value that has to be exceeded for the erosion to take place. The shear stress $\boldsymbol{\tau}$ is a force applied to the solid boundary and is defined via a power-law model [Woj+07]:

$$\boldsymbol{\tau} = K\boldsymbol{\theta}^m, \tag{8.3}$$

where $K\,[Pa \cdot s]$ is a constant usually set to 1, $\boldsymbol{\theta}\,[s^{-1}]$ is the shear rate and m is the power-law index, a constant determined by the material. The shear rate $\boldsymbol{\theta}$ can be approximated from the fluid velocity relative to the surface:

$$\boldsymbol{\theta} = \frac{\mathbf{v}_{rel}}{d}, \tag{8.4}$$

where $\mathbf{v}_{rel}\,[m \cdot s^{-1}]$ is the relative fluid velocity and $d\,[m]$ is the distance between the particle and the face in the triangular mesh.

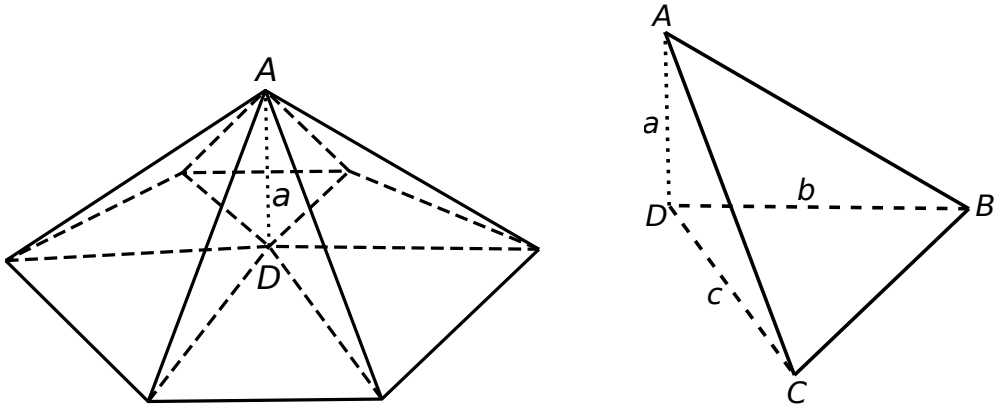
If the shear stress $\boldsymbol{\tau}$ exceeds the value of the critical shear stress $\boldsymbol{\tau}_c$, the erosion rate $\boldsymbol{\theta}$ has a positive value, and erosion takes place. The eroded sediment is assigned to the particle and is carried by the particle until the conditions for the deposition are fulfilled.

The deposition occurs when the shear stress $\boldsymbol{\tau}$ does not exceed the value of the critical shear stress $\boldsymbol{\tau}_c$. The erosion rate $\boldsymbol{\theta}$ is negative and the sediment carried by the particle is deposited onto the vertices of the face f.

8.3 Mesh Modification

When erosion or deposition takes place, the change of the mass has to be reflected in the mesh. The sediment is removed (or added in the case of deposition) uniformly from the vertices of the face f. Each vertex has a parameter assigned to store the amount of eroded (deposited) sediment. The contributions from individual particles are summed up and the mesh is only updated once in each iteration of the algorithm.

Each affected vertex needs to be moved in the direction of the normal vector according to the amount of associated sediment. However, we cannot use the amount



(a) The total volume change (b) The volume change for a single face

Figure 8.2: The volume change caused by the displacement of a single vertex

of sediment to directly determine the vertex displacement. Each vertex belongs to several faces and these faces can generally differ greatly in size and shape. The same vertex displacement could thus lead to different volume changes in two parts of the mesh with different resolution.

As we need to control the volume changes regardless of the local resolution of the scene, we have to calculate the vertex displacement based on the exact volume change. The total volume change caused by the displacement of a vertex is a sum of contributions of all the affected faces. Figure 8.2a shows an example of sediment deposition onto a vertex. The deposition caused the vertex to move from the position D to the position A, with the vertex displacement a. The volume change for one of the faces equals to the volume of the area bounded by the original face and the face created by displacing the vertex, this area always forms a tetrahedron (Figure 8.2b).

The volume of a tetrahedron V_i can be calculated as follows:

$$V_i = \frac{|a \cdot (b \times c)|}{6}, \tag{8.5}$$

where a, b and c are the edges of the tetrahedron sharing the vertex D. The total volume change V for the vertex displacement $|a|$ can be then computed as

$$V = \sum_{0}^{n-1} V_i, \tag{8.6}$$

where n is the number of faces sharing the displaced vertex. It can be derived that for $a' = ax$, where x is a constant, the total volume $V' = Vx$.

To compute the correct vertex displacement a', we first calculate the volume change V for the vertex displacement $|a| = 1$ in the direction of the normal vector. The vertex displacement a needs to be adjusted so that the volume change is equal to the amount of eroded/deposited sediment s:

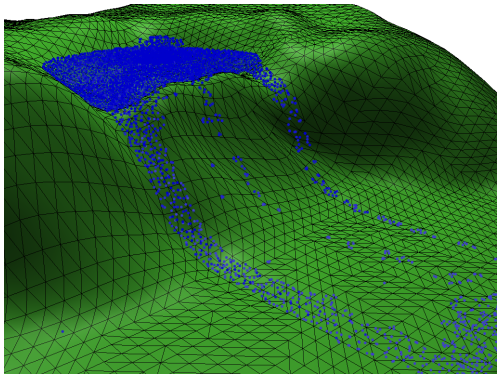


Figure 8.3: An example of an erosion simulation of an overfilling lake

$$a' = a\frac{s}{V}, \tag{8.7}$$

where a' is the required vertex displacement.

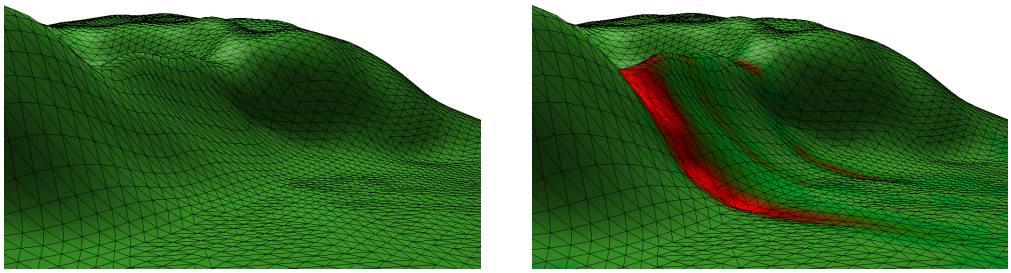
8.4 Results

We have developed our framework in C++ and run it on an Intel Core 2 Duo clocked at 2 GHz. We use the library Fluids v.2 [Hoe09] for the SPH fluid simulation that provides the necessary functionality and computes the particle properties during the simulation. The Navier-Stokes equations are scale-sensitive and the actual simulation scale of the fluid in the used implementation is approximately up to tens cm^3. As the terrain represents a large-scale scene, this inconsistency of scales can lead to less realistic behavior of the fluid.

We have verified the proposed algorithm by simulating several different erosion scenarios. Erosion is a long term process and a comparison with real-world data would be necessary to verify our algorithm. We rely the correctness on the computational fluid dynamics and we verified our solution visually.

We created a simple scenery of a lake being filled by water to validate the erosion process and to compare our solution to the results of the reference method by Krištof et al. [Kri+09]. The scene is shown in Figure 8.3. The lake represents a local depression, which is being filled with water particles. The lake eventually fills up and the fluid flows over the edge and erodes the underlying terrain surface. We use a simple particle-based visualization for the fluid, as we are interested in the resulting eroded terrain itself, not in the realistic image of the water.

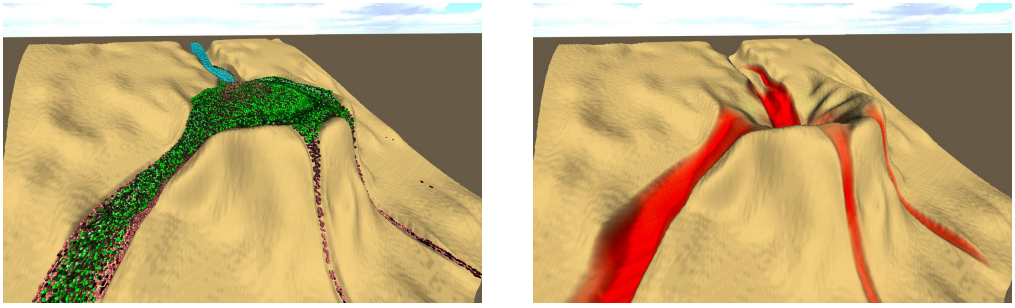
Figure 8.4 captures the alteration of the terrain during the simulation. Figure 8.4a shows the original terrain before the erosion was applied. The final eroded scene is depicted in Figure 8.4b with the eroded regions highlighted in red color. The results of the reference method are shown in Figure 8.5. It can be seen that our approach is capable of generating similar erosion effects as the reference method while working



(a) The original terrain

(b) The eroded terrain; red color highlights the eroded regions

Figure 8.4: An example of an erosion simulation



(a) The fluid eroding the terrain

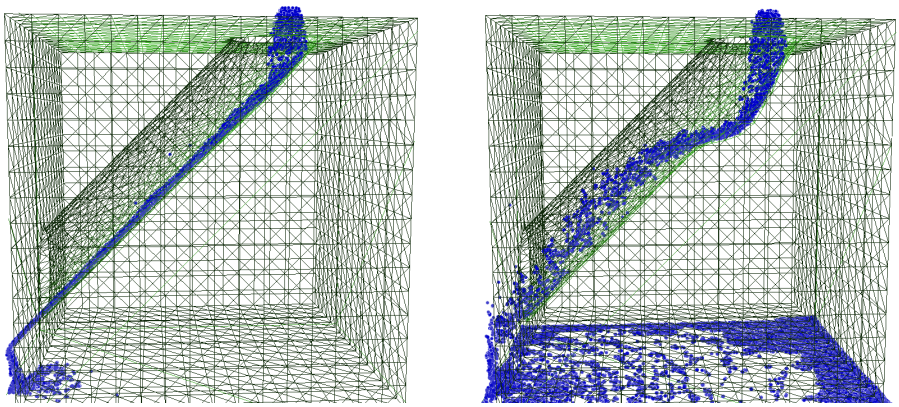
(b)) The eroded terrain; red color highlights the eroded regions

Figure 8.5: The results of the reference method [Kri+09]

directly on the triangular mesh. We do not use textures to enhance the visual quality of the generated scene in order to emphasize the geometry change.

Unlike the reference method, our method can also simulate fully 3D scenes with concave features. We modeled the demonstration scene captured in Figure 8.6a. The scene consists of a block of material with a tube-like cavity. The water runs through it and erodes its surface (Figure 8.6b). The fluid has the largest momentum before it collides with the mesh for the first time, hence the erosion is the strongest at that point. The fluid flow is affected by the changed surface of the mesh (Figure 8.6b).

Computational requirements were measured for the presented simulations. For the lake scene (Figure 8.3), consisting of 12,528 particles and 9,068 faces, we measured an execution time of one iteration of approximately 2.9 s. Each iteration of the concave scene simulation (Figure 8.6), consisting of 12,528 particles and 4,706 faces, took approximately 0.7 s. The goal of our implementation was to verify the applicability of our approach, without the emphasis on the speed of the simulation.



(a) The original scene

(b) The water erodes the mesh, leading to change of the flow of the fluid

Figure 8.6: Erosion simulation of concave features

8.5 Method Summary and Future Work

This Chapter describes a novel solution to hydraulic erosion simulation using a combination of terrains represented as triangular meshes with a fluid simulated as a particle system. This approach is capable of simulating fully 3D scenes containing features such as caves or overhangs with lower memory requirements than the existing volumetric solutions. The solution allows to adaptively change the resolution of the scene according to the shape and complexity of the terrain. The resolution can be increased in the regions that require more attention while it can be preserved in other parts of the mesh.

However, the use of the triangular meshes brings new challenges during the erosion simulation. The erosion can cause a creation of an inconsistency in the mesh, which would cause an incorrect behavior of the simulation and has to be resolved before the erosion simulation can continue. It is also more challenging to simulate erosion of a scene consisting of multiple materials when working with triangular meshes.

The following Chapters will present further improvements to the method introduced in this Chapter, as well as possible solutions to the challenges brought by the use of triangular mesh data structure as the eroded object representation.

Chapter 9

A Unified Curvature-Driven Approach for Weathering and Hydraulic Erosion Simulation on Triangular Meshes

This Chapter presents an extension of the hydraulic erosion simulation method that was presented in the previous Chapter. The method presented in this Chapter extends the base method by using curvature to control the degree of erosion at different parts of the eroded object. The erosion is then simulated by using vertex displacement in the direction given by the vertex normal and the discrete Laplace-Beltrami operator. The method can also be used to simulate weathering by simply applying the erosion globally to all vertices of the mesh, instead of limiting the erosion processes to regions that are in contact with the fluid particles.

9.1 Curvature estimation

Curvature is a surface property that describes how bent a surface is at a given point. As the proposed method works on discrete triangular meshes, the curvature cannot be calculated exactly and has to be estimated. There are many methods that can be used to estimate curvature on triangular meshes; for a thorough survey, the reader can refer to the paper by Váša et al. [Váš+16]. We have used the curvature estimation as proposed by Rusinkiewicz [Rus04] due to its simplicity and performance. However, any other curvature estimator could be used in its place as we only need the curvature to capture the rough shape of the eroded object.

We use mean curvature H in our simulation as its values correspond to the region types important for erosion; the mean curvature is negative in concave regions (valleys, gaps), zero in flat areas and positive in convex areas (hills). Throughout this Chapter, we will use the curvature color coding as depicted in Figure 9.1; negative mean curvature is represented by blue color, zero mean curvature by green color and

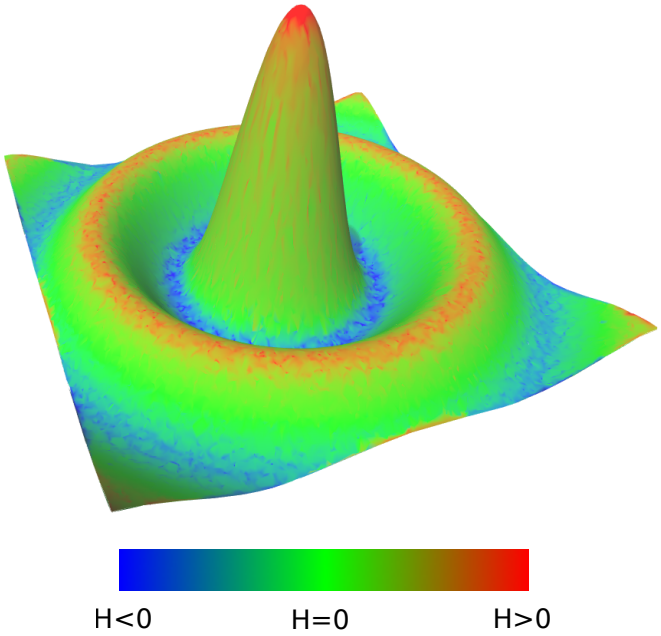


Figure 9.1: Curvature color coding. Negative mean curvature is represented by blue color, zero mean curvature by green color and positive mean curvature by red color

positive mean curvature by red color. As the curvature value is scale-dependent, we downscale the curvature value interval for each eroded object to fit in the interval $< -1, 1 >$. To preserve the zero curvature areas, we find value

$$H = max(|H_{min}|, |H_{max}|) \tag{9.1}$$

and scale interval $< -H, H >$ to interval $< -1, 1 >$.

We do not alter the curvature values for scenes where the curvature interval already fits inside the interval $< -1, 1 >$, as it could cause problems in scenes with almost no curvature.

9.2 Vertex displacement

We use the mean curvature value to estimate the desired magnitude of the erosion at the given vertex of the mesh. To estimate the direction of the vertex displacement, we use the vertex normal direction and the discretization of the Laplace-Beltrami operator.

We use the uniform discretization of the Laplace-Beltrami operator, which assigns a vector $\mathbf{l}_i$ to each vertex V_i, such that

$$\mathbf{l}_i = \frac{1}{\|N(i)\|} \sum_{j \in N(i)} (V_j - V_i), \tag{9.2}$$

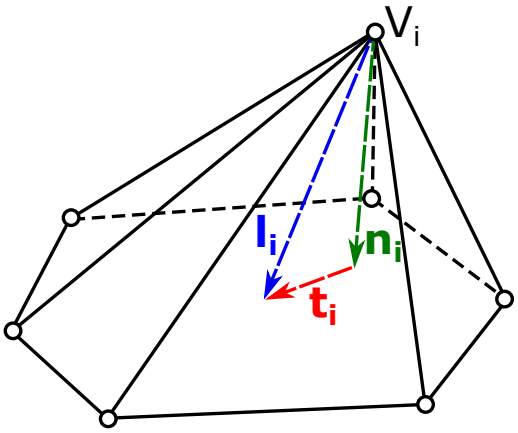


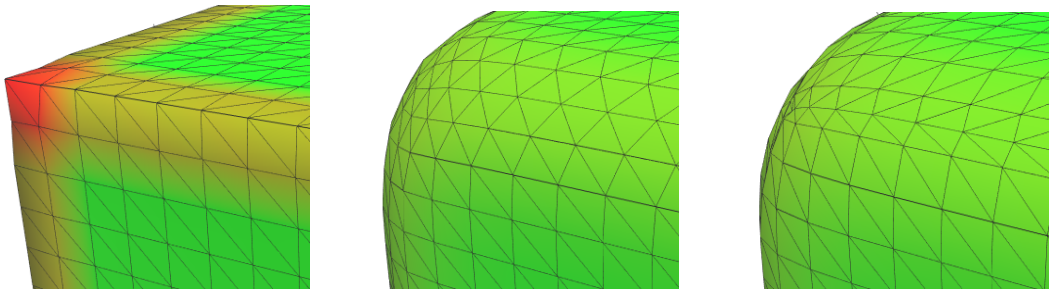
Figure 9.2: Uniform discretization of Laplace-Beltrami operator

where $N(i)$ is the set of vertices in the one-ring neighborhood of the i-th vertex. The resulting vectors $\mathbf{l}_i$ capture the tangential and normal offset of the vertex V_i from the average of its neighbors. In Figure 9.2, the vector $\mathbf{l}_i$ represents the uniform discretization of the Laplace-Beltrami operator, the vector $\mathbf{n}_i$ represents the normal offset and the vector $\mathbf{t}_i$ represents the tangential offset.

As the uniform discretization of the Laplace-Beltrami operator captures also the tangential offset, it tends to regularize the distribution of the mesh vertices. While this behavior can be undesirable in many applications, it very well suits the needs of the erosion simulation on triangular meshes in the regions of high positive or negative curvature.

The vertex displacement can generally cause the creation of small and badly shaped triangles if the direction of the displacement is not chosen carefully. The uniform discretization of the Laplace-Beltrami operator and its ability to regularize the vertex distribution alleviates the problem and results in formation of more even triangles. Figure 9.3 captures the difference between results of the erosion simulation of a cube when uniform discretization of the Laplace-Beltrami operator and the opposite direction of the vertex normal is used for the vertex displacement direction. The left image shows the original mesh, the center image shows the results of the simulation after 100 steps if the Laplace-Beltrami operator is used for the vertex displacement direction, and the right image shows the corresponding results if vertex normal is used instead. It can be seen that the use of the uniform discretization of the Laplace-Beltrami operator results in more regular and nicely shaped triangles.

On the other hand, the displacement direction based on the uniform discretization of the Laplace-Beltrami operator can cause problematic artifacts in the regions where the mean curvature is close to zero, as the tangential shift can potentially damage the mesh. For this reason, we use the direction of the uniform discretization of the Laplace-Beltrami operator for weathering simulation where we erode only mesh regions with positive mean curvature, while for the hydraulic erosion simulation we use a combination of the two aforementioned approaches.



(a) Original mesh (b) Mesh after 100 steps (Laplace-Beltrami operator) (c) Mesh after 100 steps (vertex normal)

Figure 9.3: Influence of the direction of vertex displacement on the quality of the resulting mesh. Using curvature color coding as described in Figure 9.1

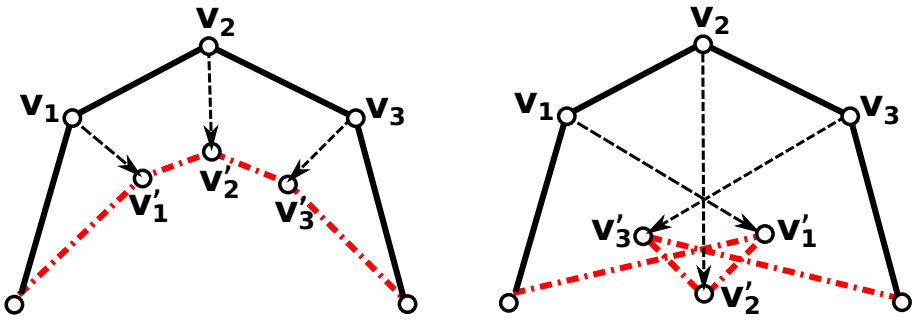


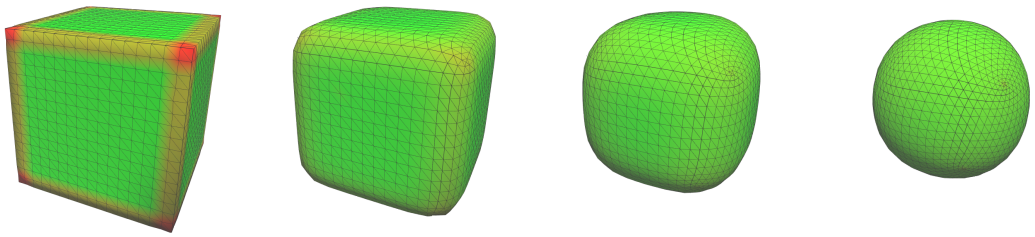
Figure 9.4: Small displacement step results in the desired smoothing effect (left), while too big displacement can damage the mesh (right)

9.3 Weathering

We simulate the weathering processes by displacing the vertices in the areas with positive mean curvature in the direction given by the uniform discretization of the Laplace-Beltrami operator. The displacement magnitude has to be chosen carefully in order to assure that each step of the simulation ends with a valid mesh. If the displacement is too big, inconsistencies can be created in the mesh. Figure 9.4 demonstrates the problem. If a small displacement step is applied, we achieve the desired smoothing effect of erosion. If the applied displacement is too big, the mesh can become tangled as a result.

To estimate a reasonable length of the displacement, we take into account the average length of the edges of the eroded mesh. The smaller the triangles of the mesh, the shorter the displacement has to be in order not to damage the mesh. We have experimentally confirmed that 1/10 of the average edge length is a good threshold for the maximum displacement length in a single step of the simulation. However, for some highly irregular meshes this approach can still fail and create inconsistencies in the highly detailed parts of the meshes; in such a case we detect the problem by finding the mesh self-intersections and we restart the simulation with shorter displacements per step.

For each vertex V_i we calculate the corresponding displacement length d_i as follows:



(a) Input model (b) After 30 steps (c) After 100 steps (d) After 200 steps

Figure 9.5: Validation of the weathering approach. After 200 steps of the simulation, the input cube model has been eroded into a sphere. Using curvature color coding as described in Figure 9.1

$$d_i = H_i \frac{e_{avg}}{x}, \tag{9.3}$$

where H_i is the mean curvature at the vertex V_i, e_{avg} is the average edge length and x is the modifier that limits the maximum length of the displacement to be $1/x$ of the average edge length e_{avg}. The position of the vertex V_i at the time step t is then calculated as

$$V_i^t = V_i^{t-1} + d_i \mathbf{l}_i, \tag{9.4}$$

where $\mathbf{l}_i$ represents the uniform discretization of the Laplace-Beltrami operator at the vertex V_i.

To enhance the erosion effect in the protruding regions even more, we can use a power of the mean curvature to calculate the erosion strength. As we still want to keep the relation between the maximum vertex displacement and the average edge length, the formula for the calculation of the vertex displacement length will become

$$d_i = H_i^n \frac{e_{avg}}{x}, n \in \mathbb{R}_{\geq 1}, \tag{9.5}$$

where n is the power modifier of the mean curvature value.

To validate the correctness of the weathering effect of the presented approach, we have applied the erosion simulation to a cube model. Figure 9.5 shows the original cube model and the model after 30, 100 and 200 steps. The power modifier n was set to 1 for this simulation. As can be seen in Figure 9.5, the algorithm converts the cube to a sphere as expected.

9.4 Hydraulic Erosion

We propose a similar approach to simulate hydraulic erosion. We use the particle-based simulation library Flex by Nvidia [NVI] to simulate the fluid. However, as we

do not rely on any specific features of the library, it could be easily exchanged for any other particle-based fluid simulation system.

To simulate the hydraulic erosion, we limit the erosion processes to the regions of the mesh that are in contact with the fluid particles. For each fluid particle that collides with the mesh, we search for all the vertices that are within the region of influence $d_{influence}$ of the particle. In our simulations, we set the region of influence $d_{influence}$ to be equal to the average length of the edge of the mesh, as it results in including most of the vertices of the affected triangles while excluding any vertices that are too distant from the fluid particles. To speed up the search for the close vertices, we use a simple auxiliary grid structure which stores the information about the spatial subdivision of the mesh.

As we want to have a smooth transition between the eroded and the still parts of the mesh, we calculate the distance factor $f_{distance_i}$ of the vertex V_i as follows:

$$f_{distance_i} = \frac{d_{influence} - d_{part}}{d_{influence}}, \tag{9.6}$$

where d_{part} is the distance between the vertex V_i and the closest particle. The distance factor $f_{distance_i}$ is then used to influence the hydraulic erosion speed as follows:

$$V_{h_i}^t = V_{h_i}^{t-1} + d_i f_{distance_i} \mathbf{l}_{n_i}, \tag{9.7}$$

where $\mathbf{l}_{n_i}$ is the vertex displacement direction. For the displacement direction, we use a combination of the vertex normal and the uniform discretization of the Laplace-Beltrami operator. For vertices with original mean curvature value of zero we use only the vertex normal, while for the vertices with mean curvature of H_{min} or H_{max} we use only the Laplace-Beltrami operator. For the remaining values, we use a linear interpolation of the vectors.

Unlike in the case of weathering, in hydraulic erosion simulation we have to be able to erode all vertices in the affected areas, regardless of their mean curvature value. The stream of water erodes flat areas as well as gaps and protrusions, but the speed of the erosion will differ. The strength of erosion in protruded areas is the highest, followed by flat areas and gaps. To mimic this behavior, we use a different approach for scaling the mean curvature. We scale the mean curvature value interval $< H_{min}, H_{max} >$ to fit in the interval $< 0, 1 >$. That way, the gaps are eroded more slowly and protrusions faster, resulting in a visually plausible simulation of erosion.

9.5 Results

We have performed extensive testing to demonstrate the visual quality of the results of our approach. This section presents the results of our weathering and hydraulic erosion simulation algorithms.

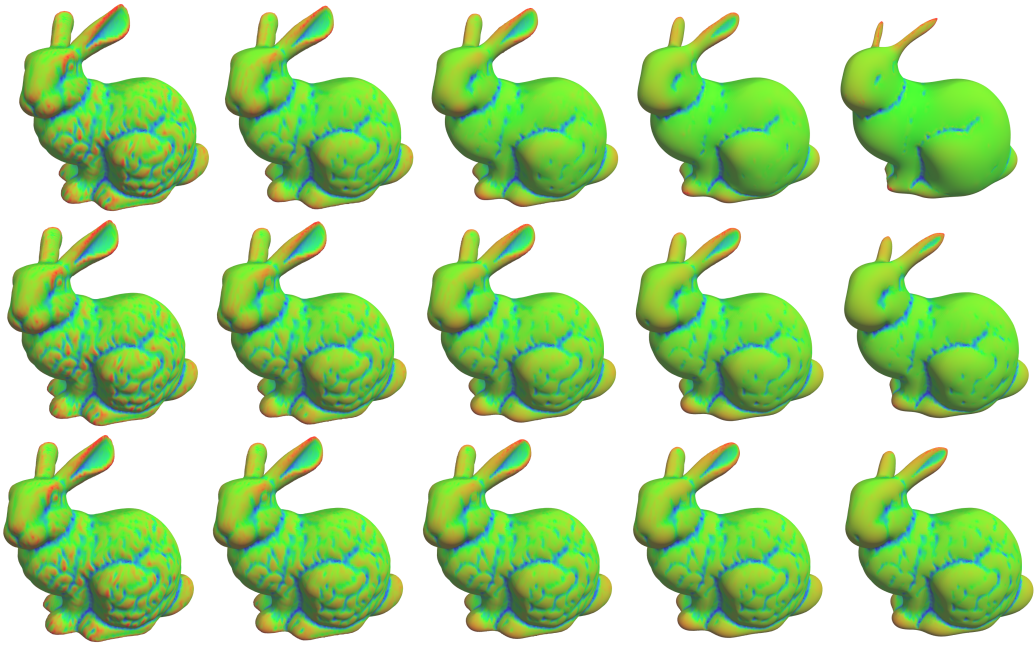


Figure 9.6: Influence of the power modifier n. Top to bottom: power modifier $n = 1$, $n = 2$ and $n = 3$. Left to right: original model, simulation after 20, 50, 100, and 200 iterations. Using curvature color coding as described in Figure 9.1

9.5.1 Weathering

Figure 9.6 shows the results of the weathering method applied on the model of the Stanford bunny [Lev+05]. The figure demonstrates the influence of the power modifier n as introduced in Eq. 9.5. For the first row of images, the power modifier $n = 1$, for the second row $n = 2$ and for the last row $n = 3$. Each row then represents the state of the simulation after increasing number of iterations. It can be seen that the increase of power modifier n results in slower erosion rate in flatter regions.

The effect of the power modifier n is the most significant on models with distinct protruded regions, such as the hand model in Figure 9.7. The figure shows the original model and the result of the simulation after 50 iterations with increasing power modifier. The figure demonstrates a possible problem in weathering highly protruded models if using low power modifier n. It can result in the creation of very thin features composed of very small triangles and ultimately it can cause a mesh to self-intersect.

To prevent such undesirable behavior, we improve the mesh by removing small triangles after every iteration. We used a tenth of the average edge length as a threshold for the face removal and the power modifier $n = 5$ in the simulation captured in Figure 9.8. The simulation erodes the protruded finger regions in a visually plausible way.

Figure 9.7: Influence of the power modifier n on a highly protruded model. Top to bottom, left to right: original model, simulation after 50 iterations with power modifier $n = 1$, $n = 2$, $n = 3$, $n = 4$, $n = 5$, $n = 10$ and $n = 20$. Using curvature color coding as described in Figure 9.1

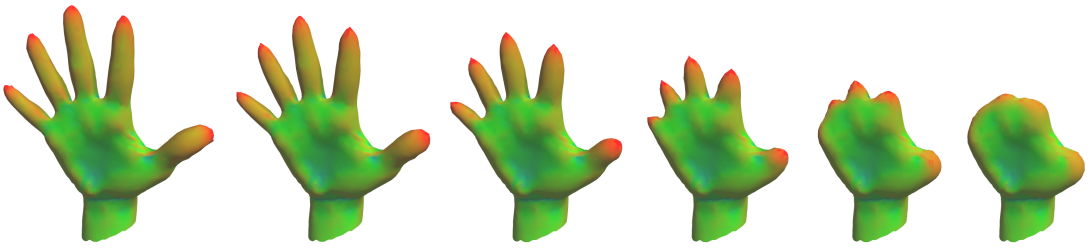


Figure 9.8: Weathering of a highly protruded model. Small faces are removed to prevent damaging the mesh. Left to right: simulation after 20, 50, 100, 200, 300, and 400 iterations. Power modifier $n = 5$; using curvature color coding as described in Figure 9.1

9.5.2 Hydraulic erosion

As mentioned before, we use particle-based simulation library Flex by Nvidia [NVI] for the fluid simulation. Figure 9.9 shows our hydraulic erosion method applied on the model of the Stanford bunny [Lev+05]. We pour the water over the bunny and simulate the hydraulic erosion using the algorithm presented in Section 9.4 only in the mesh regions affected by the water flow. This simple example shows the capability of the method to restrain the erosion to the parts of the mesh where the fluid particles collide.

We show a more realistic scene in Figure 9.10. The model represents a narrow canyon through which the water is poured. The simulation results in the generation of undercut cliffs around the flow of the fluid. Figure 9.11 compares our results

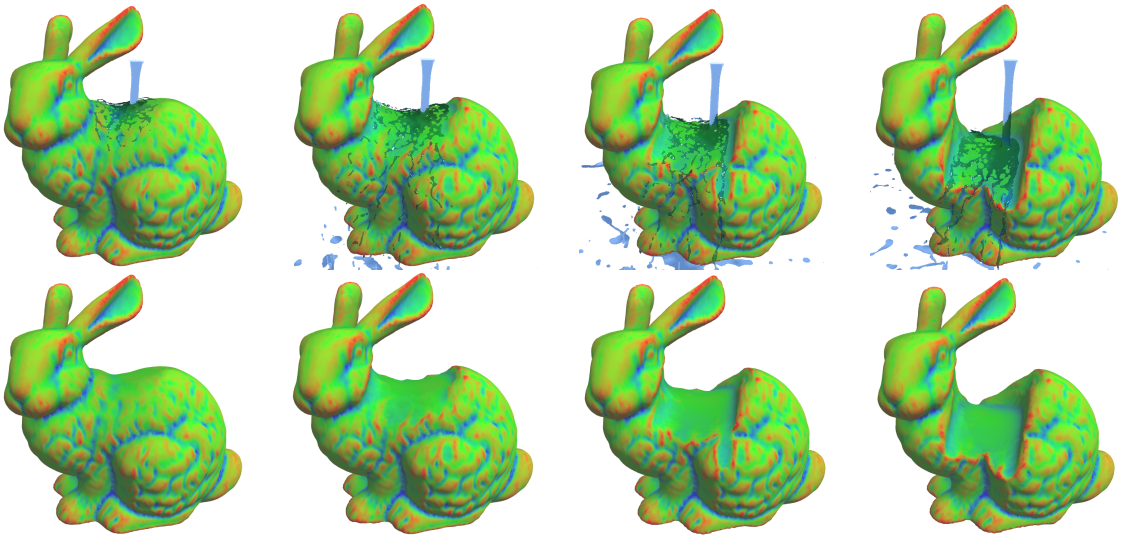


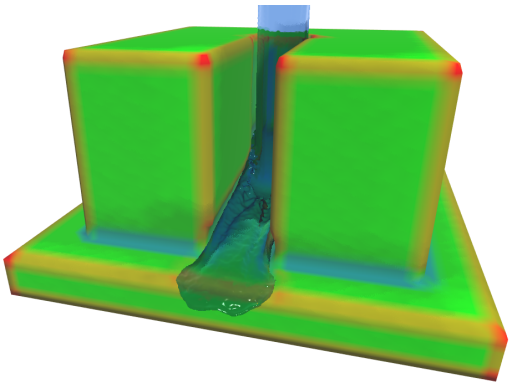
Figure 9.9: Hydraulic erosion simulation. Top: scene with simulated fluid; bottom: isolated model. Left to right: model after 40, 200, 400, and 700 iterations. Using curvature color coding as described in Figure 9.1

to the results of Tychonievich and Jones [TJ10] who demonstrate their approach on a scene of similar settings. To emphasize the effect of the hydraulic erosion simulation, we do not apply any deformation to the regions of the mesh that are not in direct contact with the fluid. This results in unnaturally flat looking side walls of the model; this could be improved by adding small amount of noise to the mesh vertices.

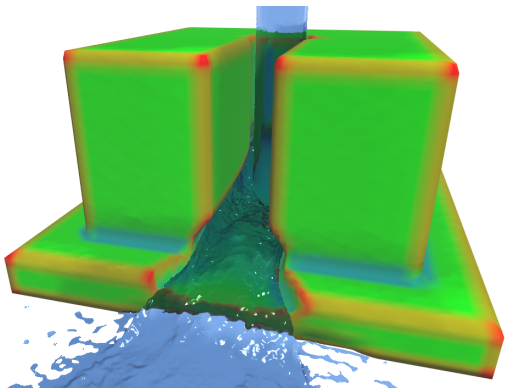
Our next example shows the hydraulic erosion simulation on real data of a river in the Northern Moravia in the Czech Republic; the region is showed on the map cut-out in Figure 9.12. The data were acquired by Lidar scanning and preprocessed by removing the vegetation and performing point reduction and afterwards triangulated using Delaunay triangulation. However, the data have very bad quality especially in the river region that is of our interest, as the Lidar scanning is incapable of scanning surfaces under water. For that reason, we performed further preprocessing using the MeshLab software [Cig+08]. We lowered the triangles representing the bottom of the river to create the river bed. Afterwards we used uniform mesh resampling followed by quadric edge collapse decimation. Figure 9.13 captures both the original triangulation and the data after preprocessing.

We simulated the process of creation of the undercuts in the river banks in the hydraulic erosion scenario captured in Figure 9.14. We simulated the erosion using the approach described in Section 9.4. The scene was set up with the following parameters: the power modifier $n = 1$ and the region of influence $d_{influence}$ of a particle was set to the average edge length. The simulation results in the creation of undermined river banks with distinct overhangs especially in the river bend regions, as demonstrated in Figure 9.14 and Figure 9.15.

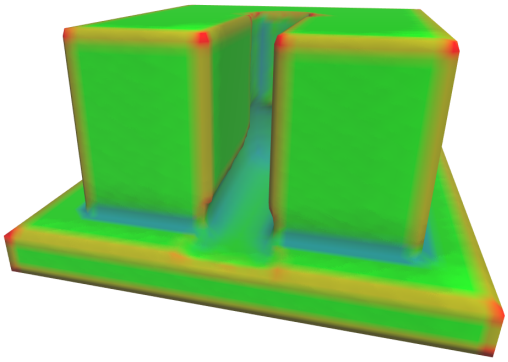
We used the same data to simulate the flooding of the area due to high amount of fast flowing water in the river. The scene was set up with the following parameters:



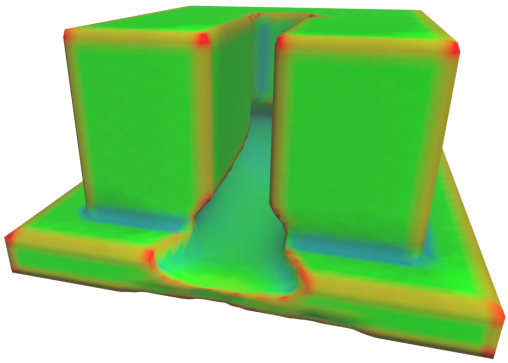
(a) Scene with water after 100 iterations



(b) Scene with water after 150 iterations

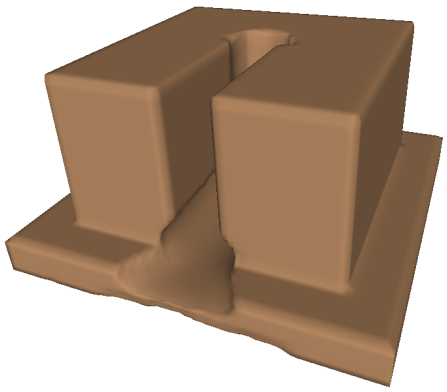


(c) Isolated model after 100 iterations

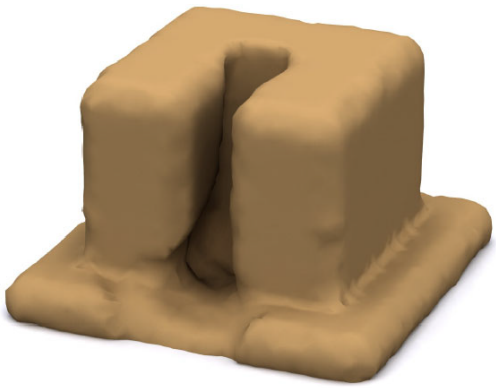


(d) Isolated model after 150 iterations

Figure 9.10: Pouring water through a narrow canyon. Using curvature color coding as described in Figure 9.1



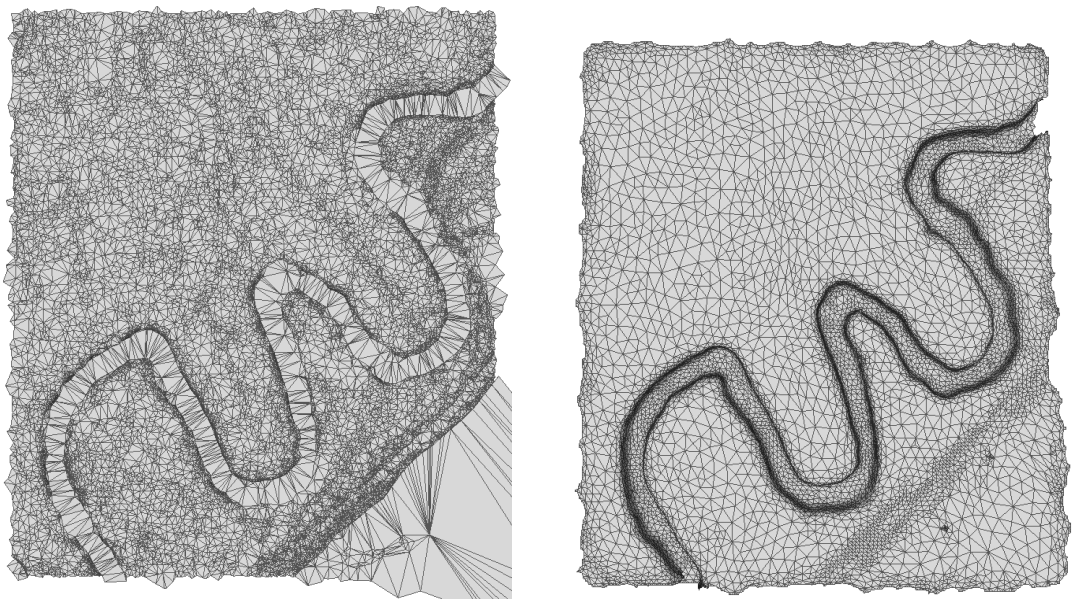
(a) Our approach



(b) Approach by Tychonievich and Jones [TJ10]

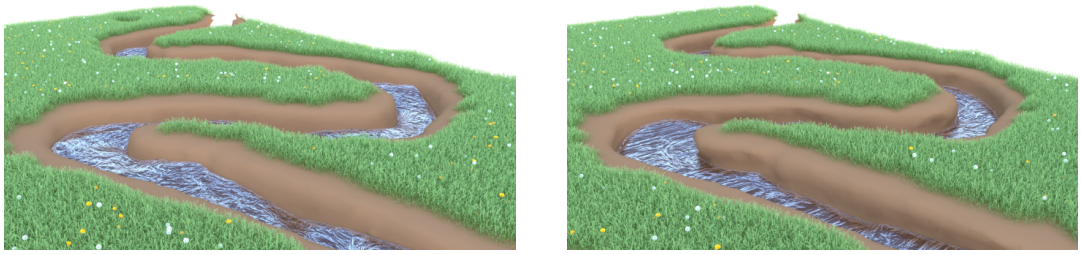
Figure 9.11: Pouring water through a narrow canyon

Figure 9.12: The river Odra in the Northern Moravia in the Czech Republic. Source: www.Mapy.cz



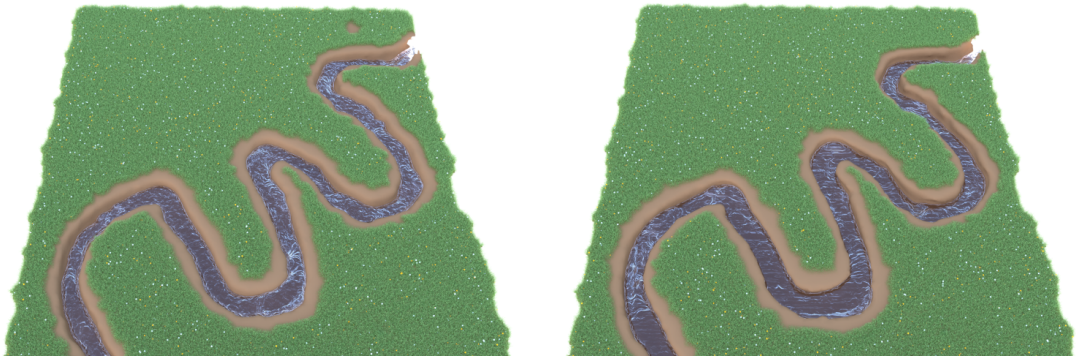
(a) Triangulated Lidar data (b) Data after preprocessing

Figure 9.13: Triangulated data of a river region captured in Figure 9.12



(a) Original model (b) Scene after 200 iterations

Figure 9.14: Running water creates undercuts on the river banks; front view



(a) Original model (b) Scene after 200 iterations

Figure 9.15: Running water creates undercuts on the river banks; top view

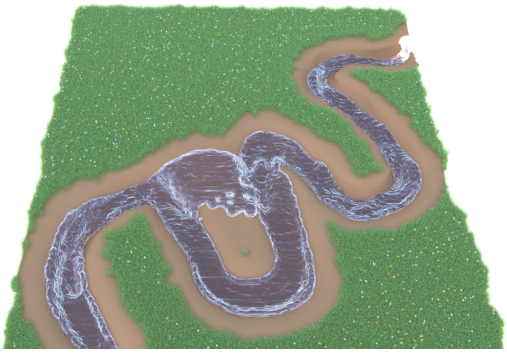
the power modifier $n = 1$ and the region of influence $d_{influence}$ of a particle was set to triple the average edge length. The increased region of influence $d_{influence}$ results in smoothing the edges of the river bed and creating more convincing results. Figure 9.16 shows the original scene and the result of the simulation after 100, 200 and 300 iterations. Figure 9.17 captures the mean curvature of the original scene and the final scene after 300 iterations.

9.5.3 Execution time

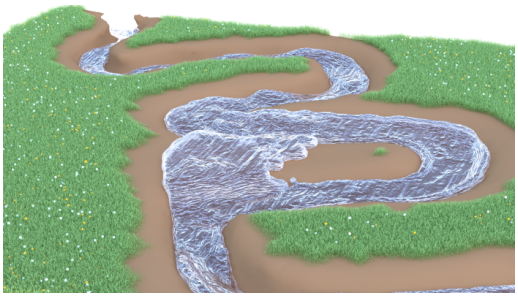
We have measured the execution time for the scenarios presented in this Section on a computer equipped by Intel i7-4770 3.4GHz, Intel HD Graphics 4600 and 16 GB RAM. We have measured the time required per iteration of the method, as well as the execution time of the most important substeps. The measured data is shown in Tables 9.2 and 9.4. Tables 9.1 and 9.3 show the number of vertices and faces of the corresponding scenes. For hydraulic erosion scenarios captured in Tables 9.3 and 9.4 we also record the average number of particles. Even though we did not dedicate any special effort to speed up the simulation, the simulation times are very close to real-time frame rates. The real-time simulation could be achieved by using more sophisticated auxiliary structures or by implementing the method on the GPU.

(a) Top view; original scene

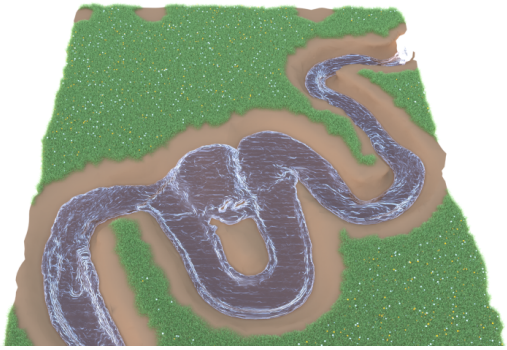
(b) Front view; original scene



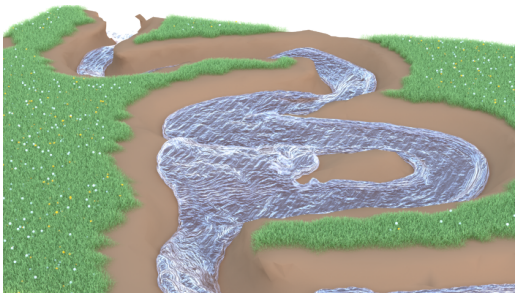
(c) Top view; after 100 steps



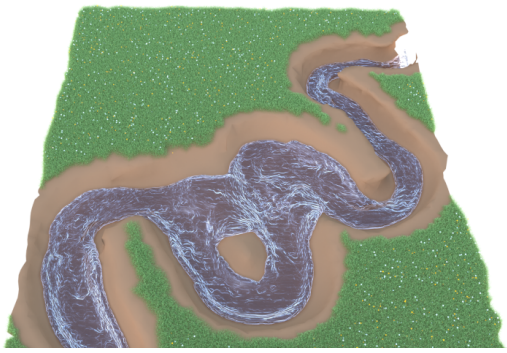
(d) Front view; after 100 steps



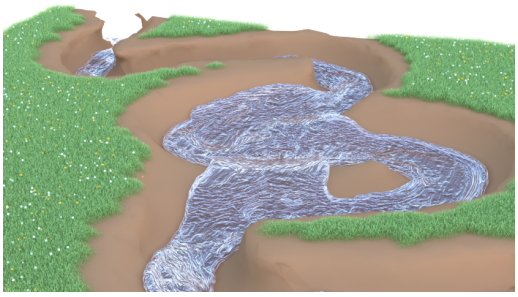
(e) Top view; after 200 steps



(f) Front view; after 200 steps

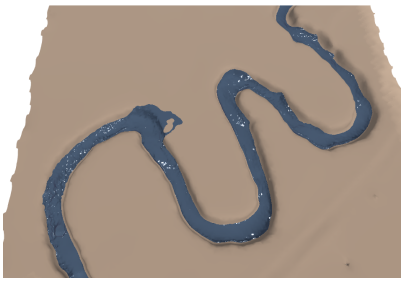


(g) Top view; after 300 steps

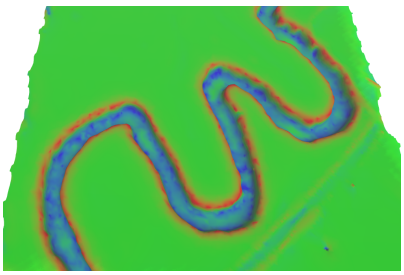


(h) Front view; after 300 steps

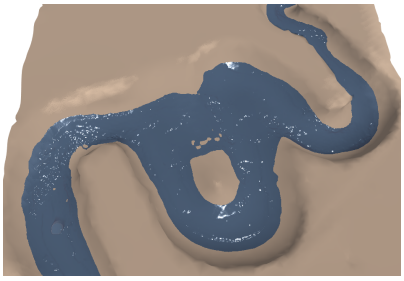
Figure 9.16: Fast water in the river erodes the river banks



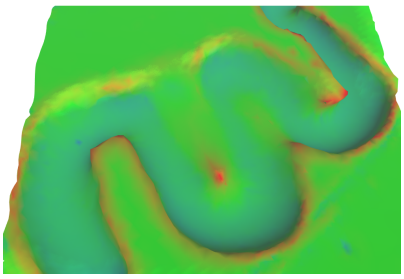
(a) Original scene; simple rendering



(b) Original scene; curvature



(c) Scene after 300 iterations; simple rendering



(d) Scene after 300 iterations; curvature

Figure 9.17: Simple rendering and mean curvature values visualization for the simulation in Figure 9.16. Using curvature color coding as described in Figure 9.1

Table 9.1: Size of the scenes for the weathering simulations

	vertices	faces
Bunny (Fig. 9.6)	34,835	69,666
Hand, n=1 (Fig. 9.7)	2,518	5,000
Hand, n=10 (Fig. 9.7)	2,518	5,000

Table 9.2: Execution time of steps of the algorithm (in ms) for the weathering simulations

	iteration [ms]	curvature calculation [ms]	Laplacian calculation [ms]	vertex displacement [ms]
Bunny (Fig. 9.6)	182.705	144.571	16.167	0.354
Hand, n=1 (Fig. 9.7)	27.464	9.907	1.234	0.028
Hand, n=10 (Fig. 9.7)	27.556	9.855	1.218	0.034

Table 9.3: Size of the scenes for the hydraulic erosion simulations

	vertices	faces	particles
Bunny (Fig. 9.9)	34,835	69,666	17,673
Canyon (Fig. 9.10)	17,130	34,256	67,561
River 1 (Fig. 9.15)	7,806	14,782	72,569
River 2 (Fig. 9.16)	7,806	14,782	72,193

Table 9.4: Execution time of steps of the algorithm (in ms) for the hydraulic erosion simulations

	iteration [ms]	calculate affected vertices [ms]	curvature calculation [ms]	Laplacian calculation [ms]	vertex displacement [ms]
Bunny (Fig. 9.9)	291.520	34.601	150.006	15.528	1.713
Canyon (Fig. 9.10)	309.441	116.228	63.749	6.069	0.834
River 1 (Fig. 9.15)	271.863	117.651	28.374	3.171	0.528
River 2 (Fig. 9.16)	478.560	370.570	28.346	3.220	0.524

9.6 Method Summary and Future Work

The method proposed in this Chapter presents a novel unified approach for the two most prevalent erosion processes that can be observed in nature, i. e., weathering and hydraulic erosion. Our approach works directly with models represented as triangular meshes and simulates the erosion by vertex displacement in the direction of the vertex normal or the uniform discretization of the Laplace-Beltrami operator, while the magnitude of the displacement is calculated based on the local mean curvature value.

The main advantage of this approach is that it offers a unified way of simulating weathering and hydraulic erosion on triangular meshes, without the need to employ other auxiliary intermediate data structures. This allows us to use the simulation method on a wide range of readily available triangular models. Triangular meshes also permit to model complex scenes with detailed features with varying density of vertices based on the local complexity of the scene.

The use of triangular meshes and the simplicity of the proposed method lead to very fast erosion simulation. We achieved almost interactive rates without deploying any special means to speed up the algorithm. As future work, it would be possible to achieve interactive simulation rates by performing the erosion calculations on the GPU.

We have demonstrated our approach on both artificial and real data and showed that our method results in the creation of visually plausible eroded models despite the use of very simple erosion simulation function. We have chosen to drive the erosion speed only on the basis of the local mean curvature value to emphasize the effect of

the curvature on the simulations. To increase the plausibility of the simulation, the erosion function could easily be replaced by more complex physically-based method, e. g., by the erosion function described in Chapter 8. Also, the plausibility could be improved by allowing the simulation of erosion of scenes composed of multiple materials or by handling the topological changes, such as splitting the model in two due to heavy erosion. In the following Chapters, we will propose possible solutions to these problems that are compatible with the triangular meshes that we use to represent the eroded objects.

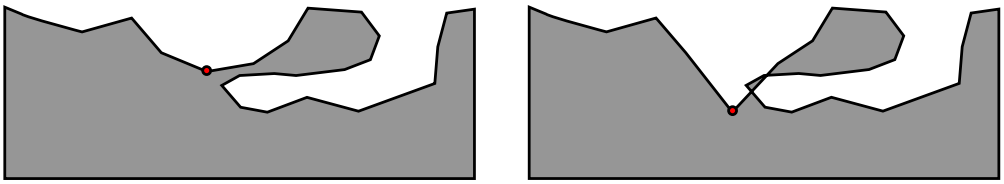
Chapter 10

A Simple and Robust Approach to Computation of Meshes Intersection

Heavy erosion can cause a creation of an inconsistency in the mesh. If a part of the mesh is heavily eroded, it is possible for two parts of the mesh to overlap, creating a topology inconsistency. A simplified 2D example of such a case is shown in Figure 10.1. Figure 10.1a shows a simple valid scene, the red point represents the vertex where the erosion will be applied. The eroded scene in Figure 10.1b shows an example of a created topology inconsistency. While erosion can cause creation of holes or breaking the mesh into pieces, deposition can cause unwanted mesh merging.

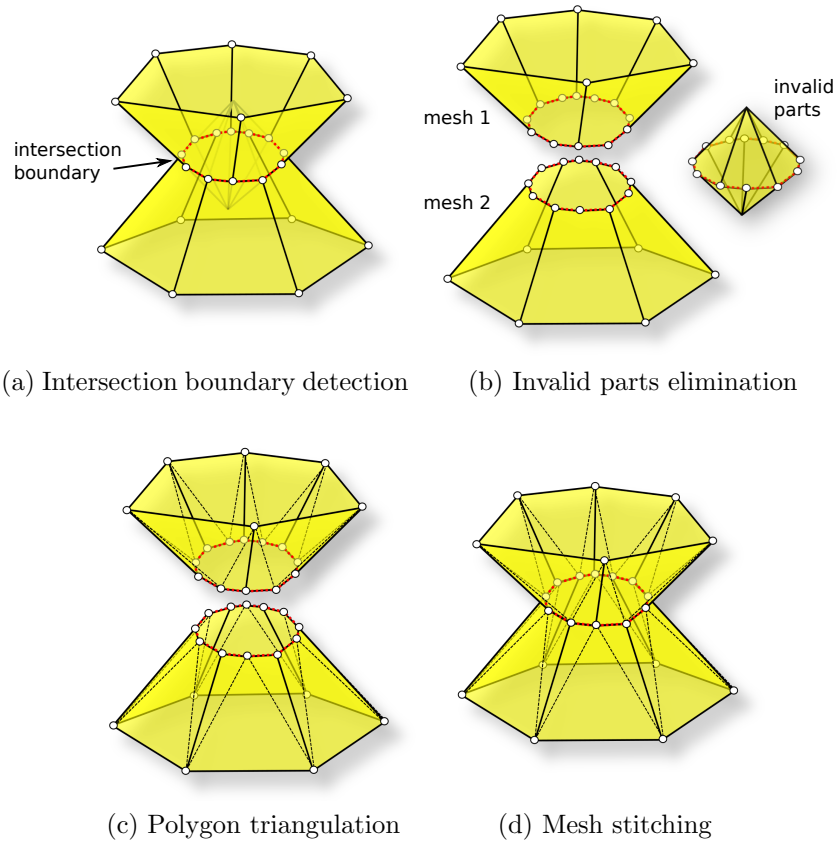
Many methods exist that address the problem of repair of intersecting meshes, for an overview the reader can refer to Chapter 5. The state-of-the-art methods usually concentrate on either being robust, accurate, or efficient. To the best of our knowledge, none of them satisfy all the three properties. We focus on the accuracy of the solution as it is very often overlooked.

Figure 10.2 shows an overview of our method. The input are two triangular meshes (or two nonadjacent parts of a single mesh) which are examined for intersection. In the first step we identify the intersection boundary (Figure 10.2a), then cut both parts of the mesh along the boundary (Figure 10.2b), triangulate the newly generated polygons (Figure 10.2c), and then we stitch the mesh together (Figure 10.2d).



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

Figure 10.1: A simple 2D example of an inconsistency in the scene with concave features



(a) Intersection boundary detection (b) Invalid parts elimination

(c) Polygon triangulation (d) Mesh stitching

Figure 10.2: Method overview

10.1 Intersection Boundary Detection

Our method for intersection boundary detection is based on the *tracing the neighbors of intersecting triangles* (TNOIT) algorithm by Lo and Wang [LW04]. The TNOIT algorithm repairs mesh (self-)intersections only if the intersection segments of the boundary are calculated precisely enough to be correctly classified into one of the cases discussed in Section 5.2.1. It fails when the intersection is not determined correctly due to the numerical imprecision of the calculation.

The problem can be alleviated by increasing the precision of the floating point arithmetic as shown in [KLN91]. However, this decreases the algorithm performance and may limit time-critical applications such as real-time simulations. Moreover, it only shifts the problems to higher frequencies. Another way of solving the problem is to use a virtual perturbation method (e.g., Simulation of Simplicity [EM90]) to slightly displace the vertices of the mesh, avoiding the boundary cases altogether. However, the vertex displacement can be unacceptable in applications where a precise solution is needed.

We address this issue by a careful inspection of all the cases which can occur due to the imprecise calculation of the intersection, while using only floating point arith-

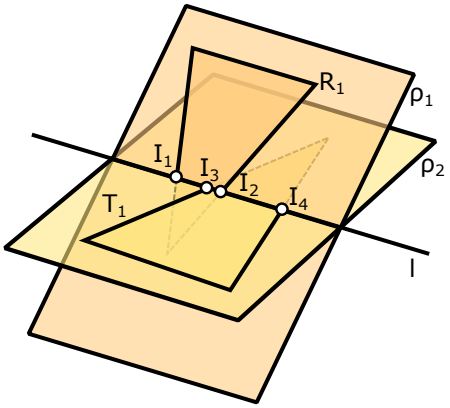


Figure 10.3: Nonparallel planes containing triangles T_1 and R_1 intersect at the line l

metic. Our solution represents a robust way of handling mesh (self-)intersections in applications where the accuracy of the solution is the main concern.

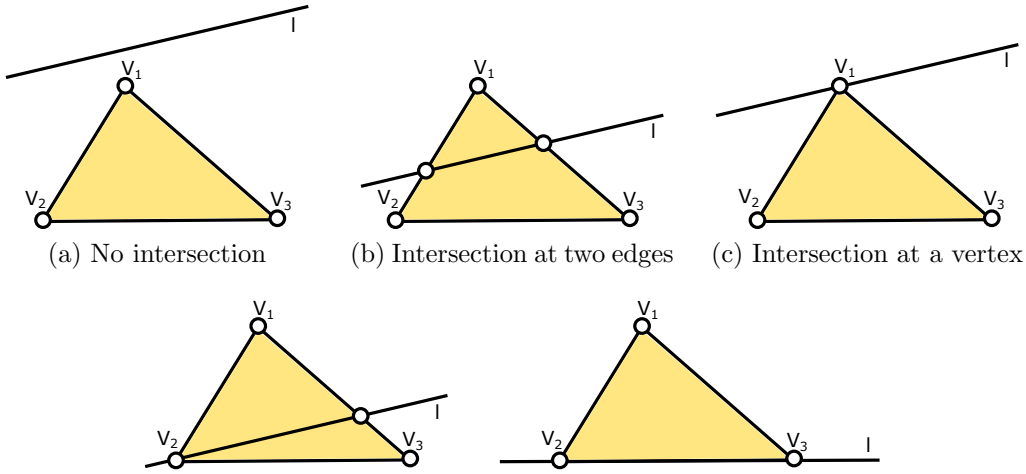
Let us open the description of the proposed solution by an analysis of the case where numerical imprecision could cause problems. Let ρ_1 and ρ_2 be two nonparallel planes intersecting at line l. Triangles T_1 and R_1 lying in these planes intersect only if they intersect the line l and if the corresponding intersection segments overlap. An example in Figure 10.3 shows this case; triangle T_1 intersects the line l in the line segment I_1I_2, triangle R_1 intersects the line l in the segment I_3I_4. The line segments overlap, i.e., the line segment I_3I_2 is the intersection of T_1 and R_1.

To correctly determine the intersection of the triangles, we need to identify the corresponding intersecting segments for each of them. Considering a triangle T located in the plane ρ and a line l in the same plane, there are five ways the triangle T can intersect the line l.

1. Line l does not intersect the triangle T (Figure 10.4a).
2. Line l intersects the triangle T at two of its edges (Figure 10.4b).
3. Line l intersects the triangle T at one of its vertices (Figure 10.4c).
4. Line l intersects the triangle T at one vertex and the opposite edge (Figure 10.4d).
5. Line l intersects the triangle T at two of its vertices and the edge formed by these two vertices is in the line l (Figure 10.4e).

The cases 1-2 can be handled even in the context of floating point arithmetic, but the situation is more difficult when we get closer to the boundary cases, because even a small error can cause an incorrect classification of the intersection cases.

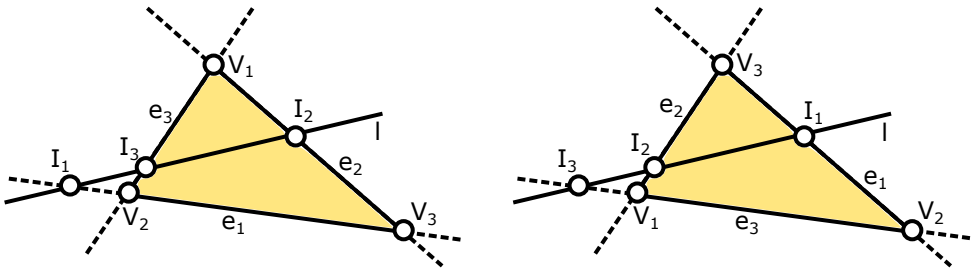
We suggest a solution to this problem based on a careful classification of all the possible cases which can be caused by the incorrect calculation of the intersection. For each pair of triangles, we first identify the intersection line l (Figure 10.3).



(a) No intersection (b) Intersection at two edges (c) Intersection at a vertex

(d) Intersection at a vertex and an edge (e) Intersection at two vertices

Figure 10.4: Possible intersections between a triangle and a line lying in the same plane



(a) An example situation (b) Vertices rotated counterclockwise

Figure 10.5: Intersections of line l and the edges of the triangle

Then we calculate the intersection segment for each triangle. The intersection line l is guaranteed to lie in the same plane as the triangle, allowing us to compute the intersections of the line l and all edges of the triangle. An example of the situation is shown in Figure 10.5a. Line l can intersect the line containing an edge in three ways: intersection lies outside the edge (I_1), at a vertex, or inside the edge (I_2, I_3).

Based on the position of the intersection points I_1, I_2, I_3, we classify the situation into one of the cases depicted in Figure 10.4. The problem gets more complicated when we take into consideration the fact that some (or all) of the intersection points might not have been determined correctly due to the numerical imprecision of the floating point arithmetic. We introduce the following substitute symbols for the relative position of the intersection points:

- 0 - the intersection point lies outside the edge,
- 1 - the intersection point lies at a vertex, and
- 2 - the intersection point lies inside the edge.

The order of the points is not relevant for the classification and so we can sort the symbolic representation in the ascending order. Using this notation, we can describe the situation from Figure 10.5a as 022 - I_1 lies outside the edge e_1, I_2 and I_3 lie inside the edges e_2 and e_3 respectively. Let us consider a similar situation where the vertices of the triangle are rotated counterclockwise (Figure 10.5b). The description of the situation using the substitute symbols would be 220. As mentioned above, we can sort the symbolic representation without the loss of generality, getting the same representation (022) for both cases. This simplification leaves us with 10 cases to address:

- 000 - all intersections outside the edges → Figure 10.4a,
- 001 - an impossible case,
- 002 - an impossible case,
- 011 - one intersection outside, two intersections at vertices,
  - two intersections at the same vertex → Figure 10.4c,
  - intersections at different vertices → Figure 10.4e,
- 012 - an impossible case,
- 022 - one intersection outside, two intersections inside the edges → Figure 10.4b,
- 111 - three intersections at vertices → Figure 10.4e,
- 112 - one intersection inside, two intersections at vertices,
  - two intersections at the same vertex, one intersection at the opposite edge → Figure 10.4d,
  - intersections at two different vertices and on the edge between them → Figure 10.4e,
- 122 - an impossible case, and
- 222 - an impossible case.

The *impossible cases* cannot occur as a result of a correct calculation. For these cases, no such straight line exists that all the intersection points would lie on it. An example of this situation is in Figure 10.6 where the line l intersects edges e_2 and e_3 close to vertex V_1. However, the incorrect calculation of the intersection puts the point I_3 outside the edge e_3 whereas the point I_2 inside the edge e_2, causing a contradiction with the original assumption that all the intersection points lie on the line l (points I_1, I_2 and I_3 are not collinear).

If the classification of the intersection ends up as one of the impossible cases, we know that the results must have been caused by numerical imprecision. In such a case we need to correct the results before the neighbor tracing algorithm can continue.

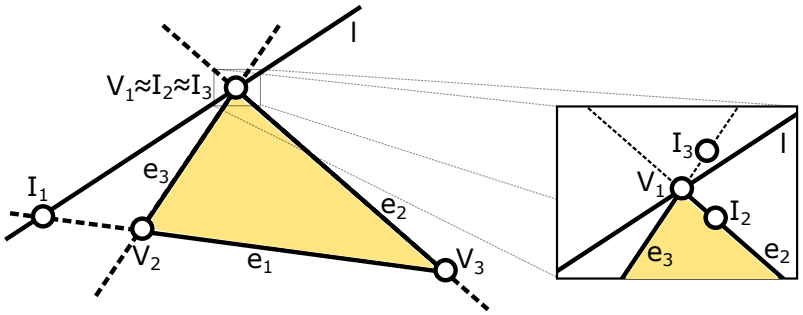


Figure 10.6: Impossible case of the intersection points as a result of numerical imprecision during the calculations

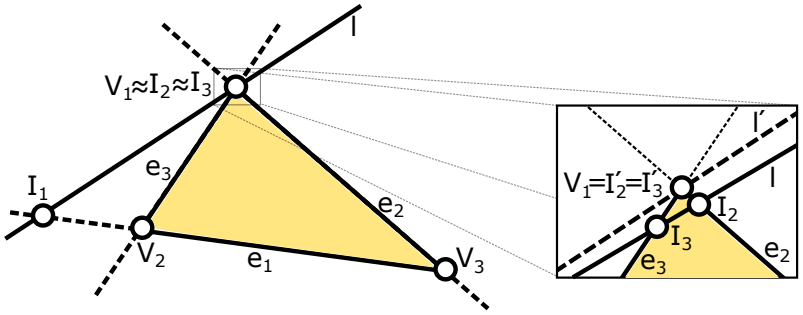


Figure 10.7: Incorrect intersection classification. The computed intersections I_2 and I_3 lie inside the edges while the correct intersections I_2' and I_3' lie at the vertex V_1

Even if the result of the classification is a valid option, some numerical error might have been introduced. The numerical errors during the calculation can cause an incorrect classification. Figure 10.7 shows an example of such a situation. Line l intersects the edges e_2 and e_3 at the vertex V_1 but the computed intersection points I_2 and I_3 are located inside the corresponding edges. This error causes an incorrect classification as the case 022 instead of the correct case 011.

We cannot tell if the classification is valid if any of the intersection points lies close to a vertex of the intersected triangle. However, if the mesh does not contain any nearly degenerate faces with edges shorter than the maximum error ε, we can continue the neighbor tracing. In such a case we can never stray away from the exact solution farther than to the direct neighbor. Figure 10.8 shows an example of such a situation where several incorrect classifications took place in a row. The green dashed line marks the exact intersection boundary while the red dotted line represents the computed intersection boundary affected by the computational error. The exact intersection is close to the vertex. The numerical imprecision causes an incorrect classification of the intersection with triangle T_6, leads the intersection boundary to T_5 instead of T_1. In an unlikely scenario it is possible for several incorrect classifications to chain, e.g., as in Figure 10.8, where the intersection is incorrectly classified for triangles T_6, T_5, T_4 and T_3. Triangles T_3, T_4 and T_5 should not be a part of the intersection at all, but they are included as a result of the numerical error of the calculation.

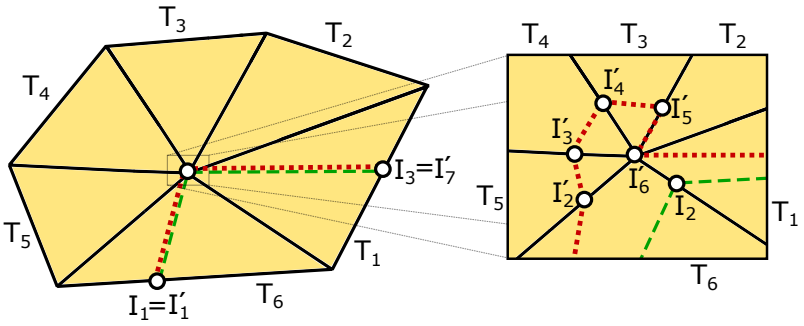


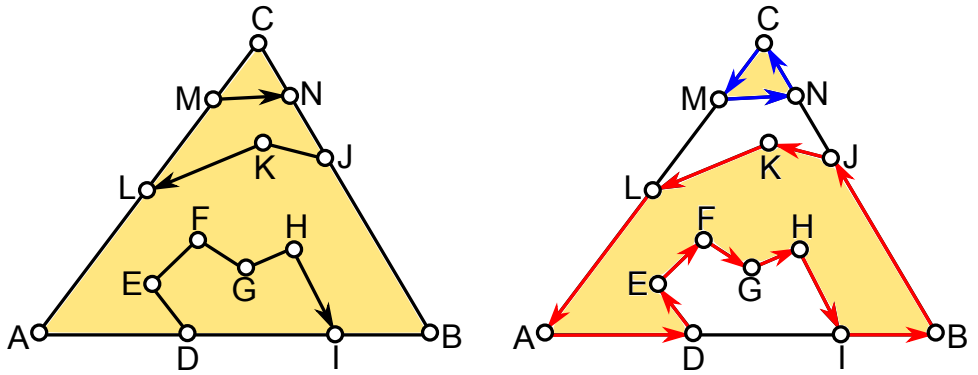
Figure 10.8: A chain of several incorrect classifications of the intersection. The green dashed line represents the exact intersection boundary, the red dotted line represents the computed intersection boundary

No intersection is found for triangle T_2. The failure to identify the intersection suggests that the previous segment or segments of the intersection boundary were affected by the numerical imprecision. Yet thanks to the condition we put on the input triangular mesh (no nearly degenerate triangles), we know the correct solution is going through a neighboring triangle. In such a case we have to test other possible classification solutions to find the right triangle where the intersection boundary continues. We start with the original case that we got as a result of the classification of the intersection of triangle T and look for the next closest option until we find the pair of triangles where the intersection boundary of the two meshes continues. The same approach is used if the original classification results in an impossible case.

10.2 Mesh Fixing

After the intersection boundary was found by using the algorithm from Section 10.1, we can fix the mesh. We propose a novel method for repairing the mesh by using the detected intersection boundary. For each intersection segment found during the neighbor tracing, we store the pair of triangles which formed the intersection. This information is used during the repair step to fix the connectivity and topology of the mesh.

For each triangle participating in the intersection, a new valid polygon is created by cutting off the inconsistent parts. To create the polygon, we first need to identify all the polylines that intersect it. We insert the polylines into an auxiliary data structure which helps to correctly trace the polygon as shown in Figure 10.9a. We start with an empty list and insert the vertices of the triangle (Figure 10.10a). We then insert the first point of each intersecting polyline into the list at the appropriate position corresponding to the line segment where the point is located in the original triangle (Figure 10.10b). Finally, we connect the end points of the polylines with the vertices that come after them in the polygon boundary (Figure 10.10c). Then we trace the boundary of the polygon: starting with any polyline, we trace the boundary following the pointers we set up in the auxiliary structure until we get back to the starting point (Figure 10.10d). The points we visited form the boundary of the new



(a) The original triangle and the intersecting polylines

(b) New polygons created by cutting off the inconsistent parts

Figure 10.9: Polygon reconstruction

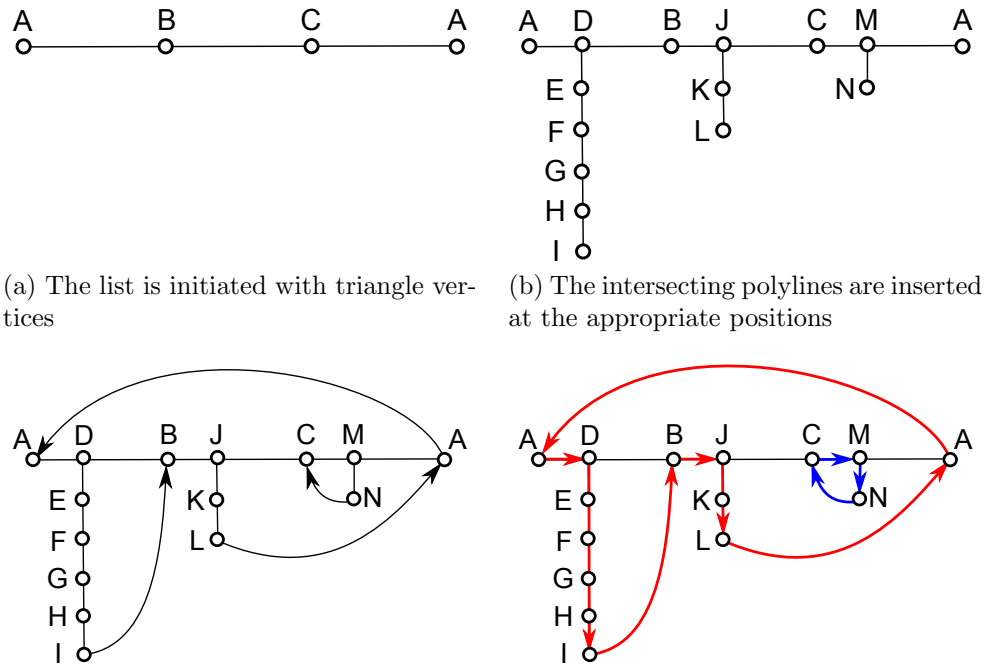
polygon. If any polyline is left unvisited, the original triangle needs to be cut into more pieces. We repeat this process until all the polygons are found (Figure 10.9b). The order of the processing does not affect the outcome of the algorithm.

Then we use the ear cutting algorithm [Mei75] to triangulate the polygons created in the previous step. The algorithm searches the polygon for *ears*, triangles such that two of its edges are consecutive edges of the polygon and the third lies completely inside it (a diagonal of the polygon). It has been proven by Meisters [Mei75] that each polygon with at least four vertices without holes has at least two ears, so the algorithm must find an ear each step. Once an ear is located, it is removed from the polygon and added to the triangulation. This process is repeated until there is only one triangle left. Finally, correct connectivity is restored through the shared intersection boundary.

10.3 Results

We tested our algorithm in various scenarios to demonstrate its robustness and correctness. Our implementation is a single-threaded C++ code and it was tested on an Intel Core i7 clocked at 3.07 GHz with 8 GB RAM.

Figure 10.11 shows the results of our algorithm for two intersecting spheres of different sizes and resolutions. The two spheres meet at the boundary captured in Figure 10.11b. The repaired mesh in Figure 10.11c is intersection-free; however, it contains many nearly degenerate faces along the boundary. This is a characteristic feature of the ear cutting algorithm [Mei75] used for the mesh repair. As changing the input mesh is generally not desirable, the nearly degenerate faces are created every time the vertices on the boundary are located close to each other. If changes of the mesh would be allowed, the mesh improvement methods could optionally be used to repair the problematic faces (Figure 10.11e). We used a combination of edge contraction algorithm (to eliminate the *needle-like* triangles) and triangle splitting (to repair the *caps*) in a way similar to [BK01].



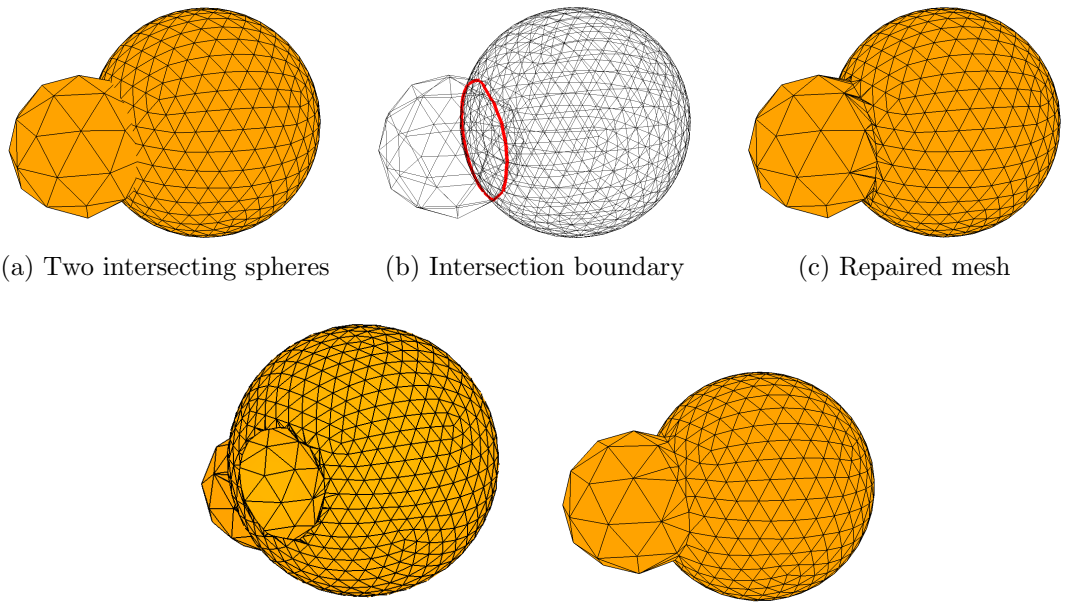
(a) The list is initiated with triangle vertices

(b) The intersecting polylines are inserted at the appropriate positions

(c) The end points are linked to the appropriate positions

(d) Polygon tracing; two polygons are found - highlighted with red and blue

Figure 10.10: Data structure used in polygon reconstruction



(a) Two intersecting spheres

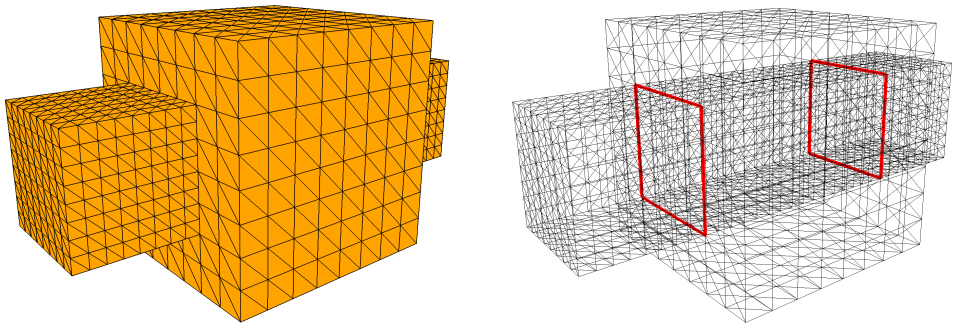
(b) Intersection boundary

(c) Repaired mesh

(d) Repaired mesh (inside look)

(e) Mesh improvement (optional step)

Figure 10.11: Intersection of two spheres



(a) Two intersecting cuboids (b) Intersection boundary

Figure 10.12: Intersection of two cuboids

Table 10.1: The number of occurrences of possible intersection cases for the scene captured in Figure 10.12, with increasing resolution, using the notation introduced in Section 10.1

faces	000	001	002	011	012	022	111	112	122	222	valid	impossible
336	11	8	0	0	5	8	0	0	0	0	19	13
3 072	33	1	55	0	1	38	0	0	0	0	71	57
49 152	116	0	201	0	5	188	0	0	0	0	304	206
196 608	224	2	364	0	0	426	0	0	0	0	650	366

Our algorithm addresses the problem of numerical inaccuracy of floating point calculations which can affect the result of the neighbor tracing algorithm [LW04] when operating near the boundary cases. We created a simple scene to demonstrate the problem. A cuboid with edge ratio 2:2:2 intersects a cuboid with edge ratio 4:1:1 in Figure 10.12a. The cuboids are aligned so that the intersection boundary is going exactly along the edges of the faces. The cuboids are rotated one degree along the x axis to magnify the problem of floating point arithmetic inaccuracy. The position of some of the vertices cannot be represented exactly in floating point arithmetic after the rotation, amplifying the inaccuracy of the subsequent calculations.

The scene was set up in such a way that we know the exact result of the intersection of the two cuboids and can compare it with the results of our algorithm. The intersection boundary is aligned with the edges, so we can assume that every triangle participating in the intersection will have one of its vertices or edges lying on the intersection boundary. Using the notation introduced in Section 10.1, the result for each segment of the boundary should be 011, 111, or 112. However, the measured results differ from these expected values due to the numerical imprecision of the floating point arithmetic. Table 10.1 summarizes the results measured for the two cuboids with increasing resolution. For the example in Figure 10.12 consisting of $3,072$ faces, the calculation results in impossible cases in 57 out of 128 instances. Furthermore, not even the valid results correspond to the expected outcome - none of the intersections was correctly identified as the cases 011, 111 or 112. Using the proposed method, we were able to repair the intersection despite the incorrect identification of the intersection cases.

Table 10.2: The resolution of the scenes

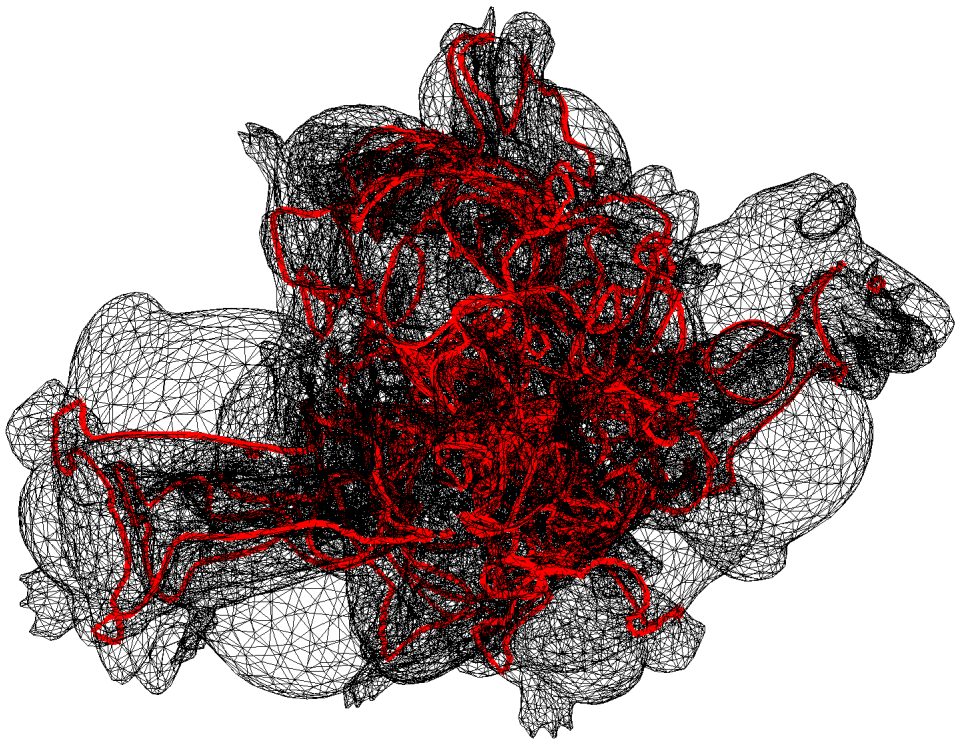
	faces	vertices
Two spheres (Figure 10.11)	1 360	686
Self-intersection (Figure 10.14)	5 120	2 564
Boolean - union (Figure 10.15c)	13 568	6 790
Two cuboids (1) (Figure 10.12)	49 152	24 582
Two cuboids (2) (Figure 10.12)	196 608	98 310
Dragons & bunnies (Figure 10.13)	76 716	38 372

Figure 10.13 shows an example of a complex intersection scene. The scene consists of six Stanford dragons and six Stanford bunnies [Lev+05] placed at random positions with random rotation. Figure 10.13a captures the intersection boundaries in the overlapped meshes that were identified by our algorithm and Figure 10.13b depicts the resulting output mesh that does not contain any intersections.

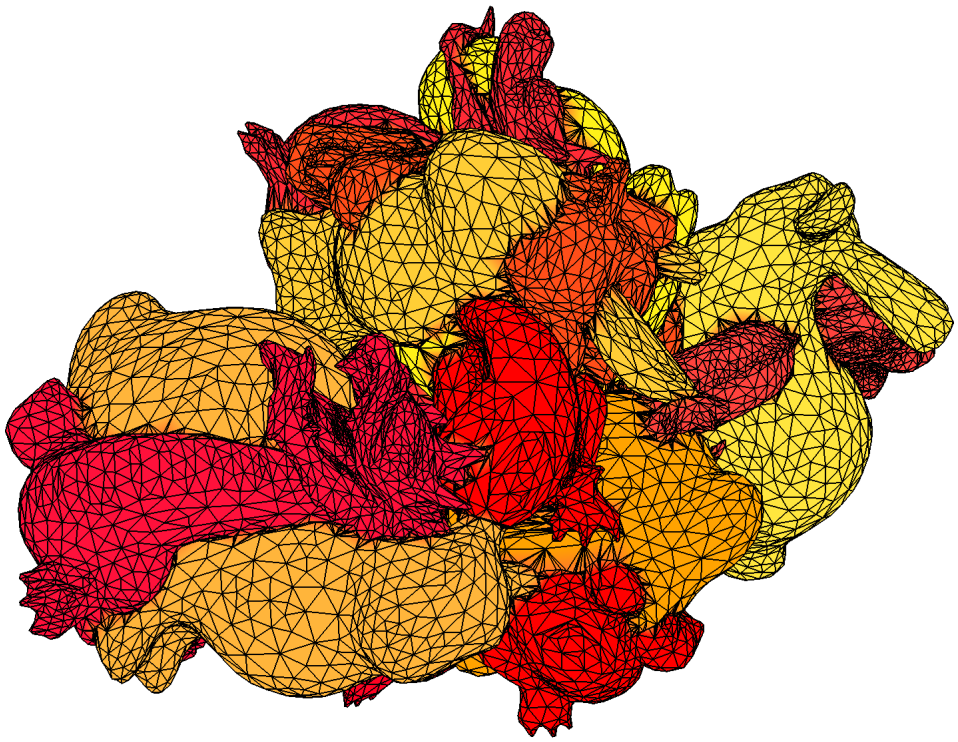
Our algorithm can detect and repair not only the intersection between separate objects, but also the self-intersections of a single model. Figure 10.14 captures such a self-intersecting object. The middle ring in Figure 10.14a represents an inconsistent part of the mesh, where the normals are reversed as a result of the mesh overlap - the normals are pointing inwards instead of outwards. We can detect the two intersection boundaries captured in Figure 10.14b using our algorithm and repair the mesh. As shown in Figure 10.14c, the operation can cause the mesh to split up into multiple pieces.

The algorithm can also be used to perform boolean operations on triangular meshes. Having two triangular meshes A and B (Figure 10.15a), we can perform boolean operations on them by setting the orientation of the normals of the mesh faces which are then used during the computation of the intersection boundary (Figure 10.15b). As we do not want to change the actual normals of the mesh, we use additional *temporary* normals to obtain the desired behavior; these *temporary* normals are used during the repair of the mesh to determine, which part of the mesh should be discarded. To get the *union* $A \cup B$, the *temporary* normals are identical to the actual normals (Figure 10.15c). On the contrary, the boolean operation *intersection* $A \cap B$ can be defined by setting all the *temporary* normals pointing inward the mesh (Figure 10.15d). For the *difference* $A \setminus B$, the *temporary* normals of A are pointing outwards, while the normals of B are pointing inwards (Figure 10.15e). Similarly, Figure 10.15f shows the result of the *difference* $B \setminus A$.

Table 10.2 contains the number of faces and vertices of the scenes presented in this Chapter. Table 10.3 shows the measured execution time. The measured data demonstrate that we usually find all the intersection boundaries long before we search the entire mesh, e.g., in the Two cuboids (2) example (Figure 10.12). Nevertheless, we cannot stop the calculation early, because we do not know if all the boundaries have been found until we finish searching the whole mesh. The Dragons & bunnies example (Figure 10.13) shows that for complex scenes with many separate intersection boundaries we have to go through most of the mesh to find all the intersections.

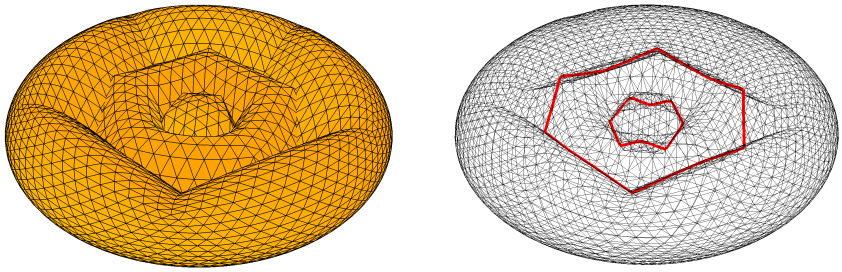


(a) 172 separate intersection boundaries detected in twelve different overlapping meshes



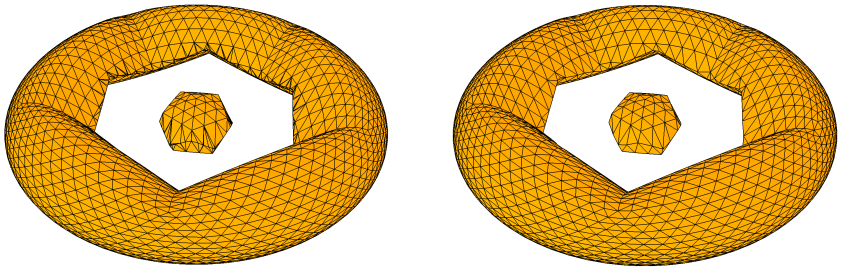
(b) A single output mesh without the intersections

Figure 10.13: An example of randomly placed six Stanford dragons and six Stanford bunnies [Lev+05]



(a) Self-intersecting model

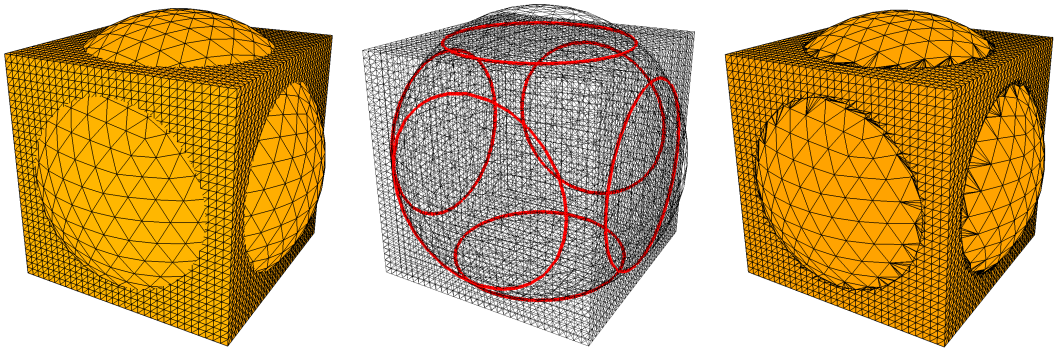
(b) Intersection boundary



(c) Repaired model

(d) Mesh improvement (optional step)

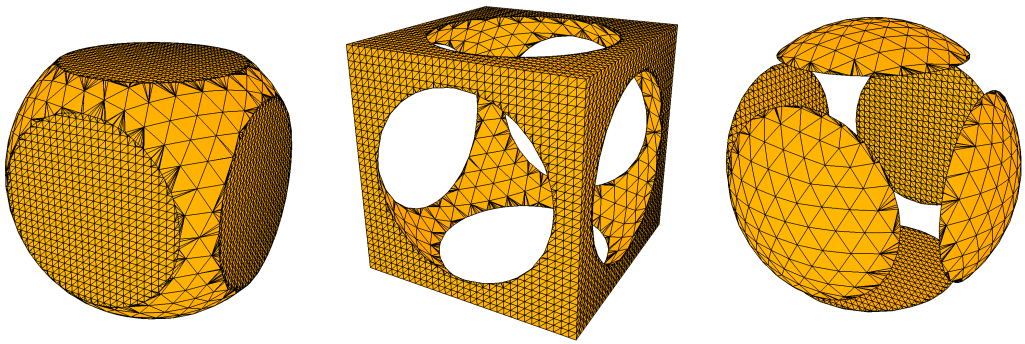
Figure 10.14: Example of self-intersecting model



(a) Meshes A and B

(b) Intersection boundary

(c) Union $A \cup B$



(d) Intersection $A \cap B$

(e) Difference $A \setminus B$

(f) Difference $B \setminus A$

Figure 10.15: Boolean operations on two triangular meshes; cube A and sphere B

Table 10.3: The execution time of individual steps of the algorithm (in ms)

	Find intersection boundaries	Search whole mesh	Fix mesh
Two spheres (Figure 10.11)	4	51	66
Self-intersection (Figure 10.14)	5	72	15
Boolean - union (Figure 10.15c)	24	221	43
Two cuboids (1) (Figure 10.12)	63	394	33
Two cuboids (2) (Figure 10.12)	24	1 454	76
Dragons & Bunnies (Figure 10.13)	7 358	7 631	476

The examples presented in this section demonstrate the ability of the method to find the intersection boundary and repair the intersecting meshes even in the non-trivial cases, such as a scene containing many separate intersection loops (Figure 10.13) or a scene with many boundary cases (Figure 10.12).

10.4 Method Summary and Future Work

In this Chapter, we have proposed an approach for the repair of intersecting meshes based on the neighbor tracing algorithm [LW04; McL+13] with the emphasis on the accuracy. Unlike the previous work, we do not use arbitrary precision arithmetic to achieve the accuracy of the solution, nor do we use virtual perturbation to avoid the undesirable boundary cases, as the change of input data can be unacceptable in some applications. Our method does not introduce any alteration of the input, works with floating point arithmetic and achieves accuracy through a careful classification of all sub-cases that could be caused by numerical imprecision. The neighbor tracing can be damaged by a numerical error near the boundary cases but for a mesh without nearly degenerate triangles, the intersection boundary can be correctly traced and repaired thanks to the classification of the intersection.

However, if this method is used to repair topology of dynamic meshes, such as meshes being deformed due to the influence of erosion processes, it usually cannot be guaranteed that no nearly degenerate triangles will be created in the mesh during the evolution of the mesh. In such cases, we remove the almost degenerate triangles from the mesh as a preprocessing step before the topology repair.

As the method works with single precision floating point arithmetic, it could be implemented on the GPU, where the higher precision operations can be very expensive. The transformation of the algorithm to be able to run it on the GPU is one of the possible avenues for future work.

Chapter 11

Complex Multi-Material Approach for Dynamic Simulations

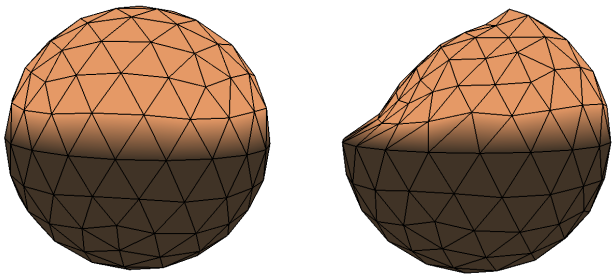
A real-life scene is composed of objects made of different materials which are eroded in a different way; hard and resistant materials are eroded slowly, while the erosion of soft materials is usually happening much faster. To be able to simulate such phenomena, we need means to consistently describe the material of an object. A common way of representation is to have a separate mesh for each material present in the scene. This approach is suitable for simulation of static scenes, where the materials are strictly separated. If the scene is dynamically changing or contains objects made of gradually changing material, a different approach is necessary.

We have tested several ways of material description that allow the definition of multiple materials for a single mesh, ranging from a material definition for each vertex of the mesh to a more sophisticated method of binary space partitions (BSP). We describe the methods in the following text.

11.1 Material in a Vertex

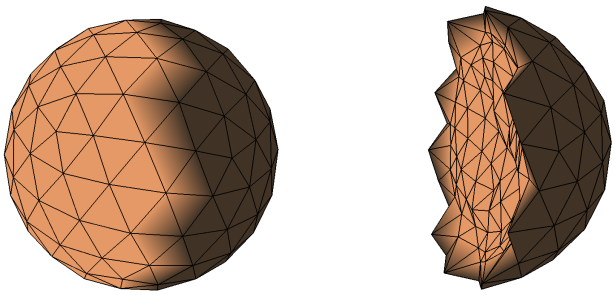
The most simple way to define multiple materials for a single mesh is to assign material properties to each individual vertex. This approach is very easy to implement; however, it has disadvantages as well. This kind of material definition applies only to the surface of the object, not to the volume. Using this approach, the result of the erosion simulation will change according to the direction of the erosion. If the erosion direction is parallel with the boundary of the individual materials, the erosion will be simulated correctly. Figure 11.1 shows an object made of two different materials. The dark brown material is hard and sturdy, while the light brown material is soft and easily erodible. Figure 11.1b captures the result of the erosion simulation.

The disadvantage of the approach is illustrated in Figure 11.2. The boundary between the materials is assumed to be straight and going through the middle of the object, perpendicular to the direction of the erosion. Figure 11.2b shows that the



(a) Original model (b) Eroded model

Figure 11.1: Material properties assigned to each vertex, correct case. Erosion force is applied from the left



(a) Original model (b) Eroded model

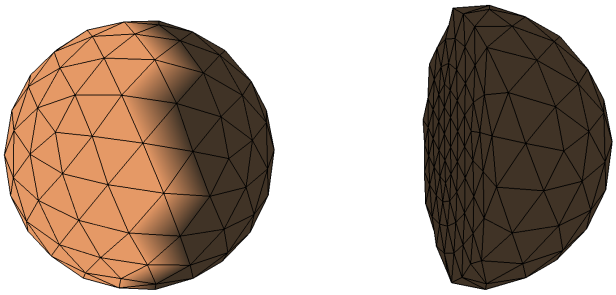
Figure 11.2: Material properties assigned to each vertex, incorrect case. Erosion force is applied from the left

soft vertices (light brown) have been incorrectly eroded beyond the boundary. However, if the boundary inside the object was curved, the result shown in Figure 11.2b could be correct. This ambiguity is the main downside of the method.

11.2 Division by a Plane

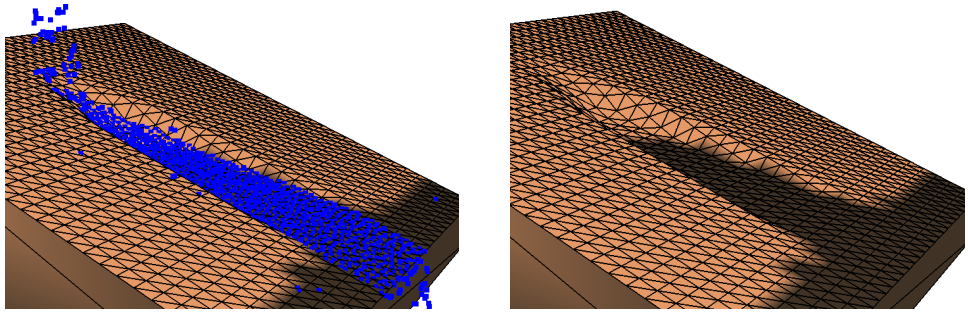
The problem of the previous approach can be reduced, if the material properties are defined for the whole volume of the scene, using a plane to separate different materials. Figure 11.3 shows the situation from Figure 11.2, but the material definition is done by a dividing plane. Material is not assigned to each vertex, but is dynamically determined based on the eroded vertex location during the simulation. The simulation gives the expected result, as shown in Figure 11.3b.

This approach allows the simulation of a simple terrain composed of several materials, imitating the layered nature of the terrain. Figure 11.4a captures a simple terrain composed of two materials being eroded by a stream of water. The hydraulic erosion is simulated using Smooth particle hydrodynamics (SPH) [GM77] particles, which erode the upper layer of the soft material, exposing the underlying hard one (Figure 11.4b). The particle-based SPH approach is very common in fluid



(a) Original model (b) Eroded model

Figure 11.3: Material defined by a dividing plane. Erosion force is applied from the left



(a) Stream of water erodes the terrain (b) Eroded model

Figure 11.4: Hydraulic erosion simulation using SPH particles

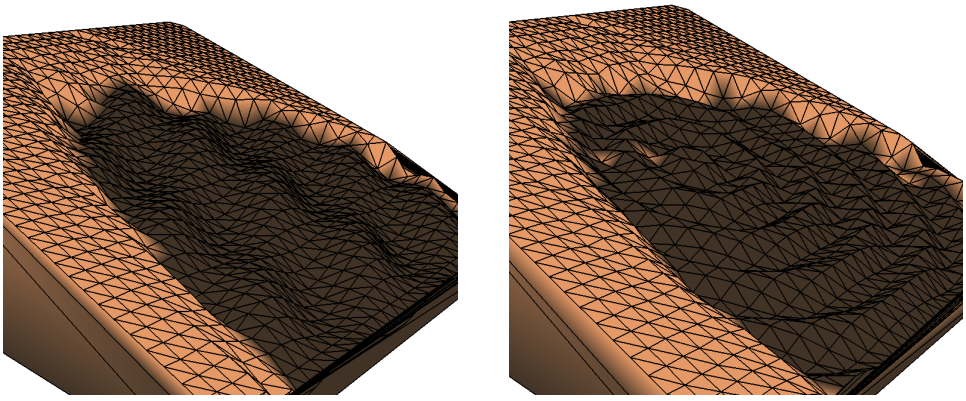
simulation. In hydraulic erosion simulation, it allows to speed up the simulation by concentrating the calculations to regions where the fluid is present. The erosion approach is similar to the method used by Krištof et al. [Kri+09] and Skorkovská et al. [SKB15], but any other erosion simulation could be easily used instead.

11.3 Division by a Function

The division by a plane can imitate the material distribution of a simple small-scale scene; however, in bigger scale, the boundary between materials will usually not be linear. To simulate a more complex boundary, an implicit division function can be used. The function defined by Equation 11.1 describes an implicit surface if $d = 0$. A point $A[x, y, z]$ lies on the left of the surface for $d < 0$ and on the right for $d > 0$.

$$F(x, y, z) = d; x, y, z, d \in \mathbb{R} \tag{11.1}$$

This approach allows the description of a nonlinear boundary analytically, without the use of a memory-demanding data structure such as the volumetric approach. Figure 11.5 shows a demonstrative example of the function division approach. The terrain starts as a plane and is eroded by water particles. The boundary between



(a) Division defined by Eq. 11.2 (b) Division defined by Eq. 11.3

Figure 11.5: Material defined by a dividing function

materials is defined by Equation 11.2 for Figure 11.5a and by Equation 11.3 for Figure 11.5b. This approach can be used with little effort to represent a hilly landscape. Unfortunately, a realistic boundary usually cannot be described by such a simple function. Finding the appropriate complex function that follows the desired boundary can be a troublesome task. Another downside of this approach is the need to enumerate the function for each eroded vertex to determine the material.

$$\cos\left(\frac{x}{2}+\frac{y}{3}\right)-\sin\left(\frac{y}{3}\right)-\frac{x}{3}-z+9=0; x,y,z\in\mathbb{R} \tag{11.2}$$

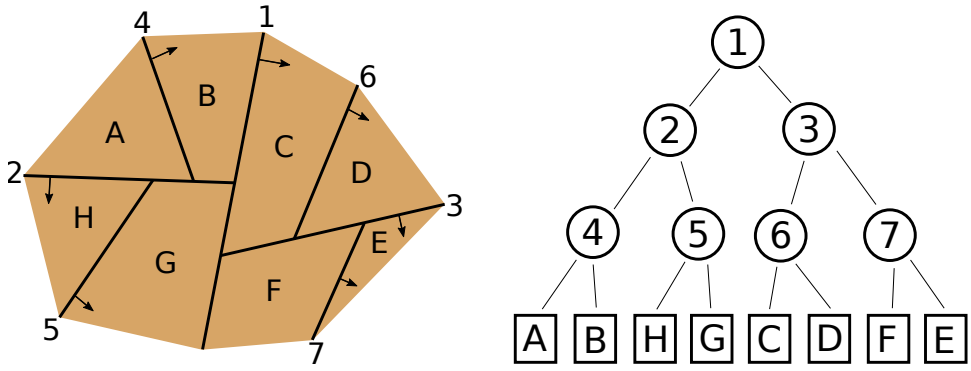
$$\sin\left(\sqrt{x^2+y^2}\right)-\frac{x}{3}+9-z=0; x,y,z\in\mathbb{R} \tag{11.3}$$

A distance function can be associated with the division function to enhance the capabilities of the approach. For a function defined by Equation 11.1, the absolute value of d grows with the increasing distance from the implicit surface. Therefore it can be used to simulate gradually changing materials if the material distribution is defined as a function of d.

11.4 Binary Space Partitions

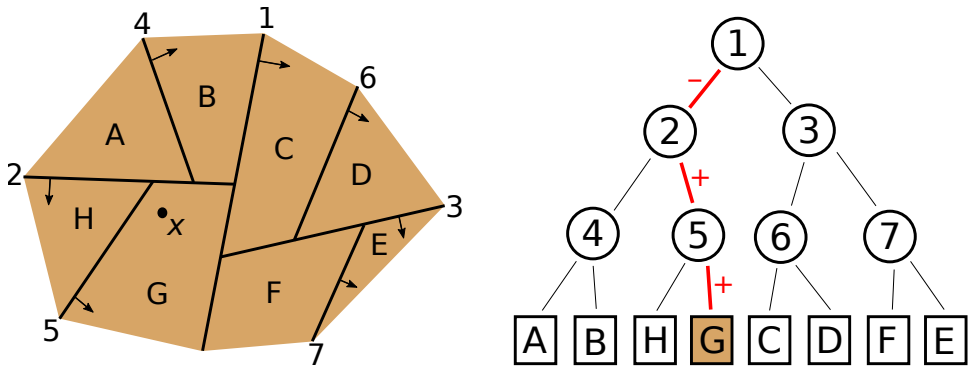
The methods of multiple material definition described in the previous sections can be used to describe the material distribution of a simple scene containing only two distinct materials. To simulate more complex distribution of the material, a more sophisticated approach is needed. We are using a binary space partitioning (BSP) [FKN80] approach that can be seen as an extension of the method described in Section 11.2, allowing the splitting up of the scene into more parts using multiple division planes.

The BSP method recursively subdivides the space into convex regions by splitting planes. The planes are stored in a binary tree as the inner nodes, while each leaf of



(a) Planar subdivision (b) BSP tree

Figure 11.6: A simplified 2D example of the BSP



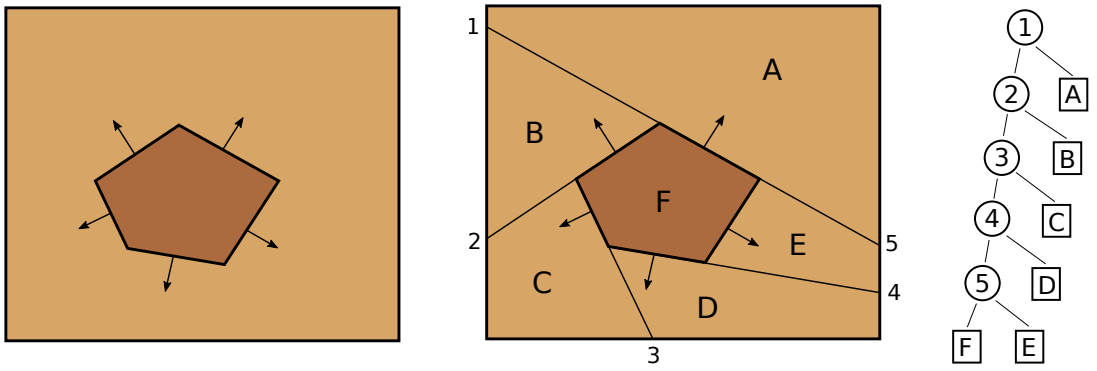
(a) Location of the point x (b) Path to x in the BSP tree

Figure 11.7: Location of a point using a BSP tree

the tree represents a convex part of the space and contains the information about the material of the subspace delimited by the corresponding planes. Figure 11.6 shows a simplified 2D example of the BSP subdivision scheme. The upper half-space delimited by each of the lines is marked by the direction of the arrow. 2D examples will be used throughout this section as they are more comprehensible than their 3D counterparts; however, the principles stay valid in the higher dimension.

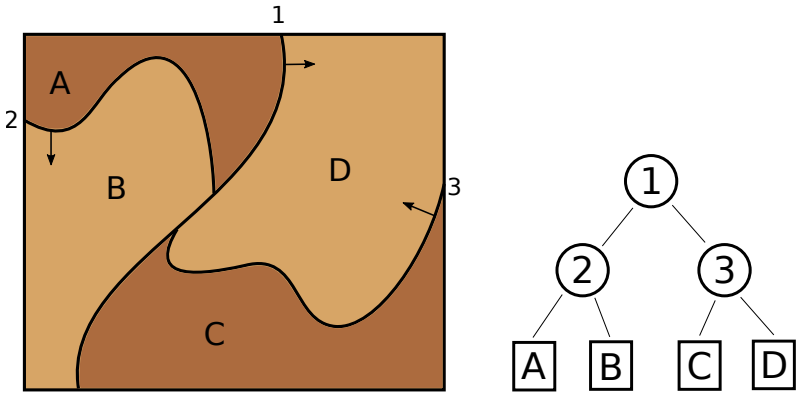
To decide which material should be assigned to a vertex, we need to search the BSP tree and determine, for each level of the tree, on which side of the division plane the vertex lies. When a leaf of the tree is reached, the subspace the vertex belongs to has been found. Figure 11.7 shows an example of a point location using a BSP tree. Figure 11.7a shows the planar subdivision and the position of the point x. The search process using the BSP tree is captured in Figure 11.7b.

The binary space partitioning approach can be used to split the scene up into convex regions of defined material properties. However, for some scenarios, the BSP tree can degrade to a list, thus increasing the search complexity. Let us consider an example captured in Figure 11.8. Figure 11.8a shows the material distribution in the scene - a convex region made of a single material is surrounded by a different type of material. Regardless of the order in which the edges of the polygon are



(a) Material distribution (b) Planar subdivision (c) BSP tree

Figure 11.8: Definition of a convex region using BSP



(a) Planar subdivision (b) BSP tree

Figure 11.9: Planar subdivision using nonlinear splitting functions

inserted into the BSP tree, the region F will always lie on the left side of the splitting line (Figure 11.8b), resulting in a degenerate tree (Figure 11.8c). However, these scenarios are not very common in erosion simulations.

The idea of binary space partitioning can be extended to work with nonlinear elements to simplify the definition of more complex scenes. Instead of using planes to split the space up into two parts, any curved surface that can be expressed by an implicit function can be used, following the approach introduced in Section 11.3. The extended BSP data structure significantly simplifies the description of nonlinear boundaries between materials. The scene in Figure 11.9 would require many splitting planes to obtain an approximate space subdivision, while the extended approach only needed three splitting functions to get the desired result.

11.5 Automated Generation of the BSP Tree

The binary space subdivision can be used as a memory efficient means of material distribution description in a scene. However, for big and complex scenes, the manual definition of splitting functions would be complicated and time-consuming.

We have proposed an automated method for the generation of the BSP tree. Typical material distribution information comes in the form of volumetric data. To construct the BSP tree, a triangular mesh needs to be extracted from the volumetric data for each value that represents a significant boundary in the input data. We use the marching cubes method [LC87] for the isosurface extraction with the threshold parameter defined by the user.

The extracted triangular mesh (or triangular meshes) is then used to construct the BSP tree. We construct the BSP tree incrementally, inserting a random face of the mesh one at a time. We use the Durstenfeld's [Dur64] optimized version of the Fisher-Yates shuffle method [FY56] to randomize the order of the faces of the mesh.

Algorithm 1 Face insertion

```
 1: procedure InsertFace(face)
 2:     plane ← root of the BSP tree
 3:     center ← center of the face
 4:     while plane != NULL do
 5:         if face lies in plane then return false
 6:         if F(center) > 0 then
 7:             plane ← plane.rightChild
 8:         else
 9:             plane ← plane.leftChild
10:     plane ← plane containing face
11:     return true
```

The insertion of a single face f is described by Algorithm 1. The algorithm searches through the tree and, for each level of the tree, it determines the position of the face f in relation to a splitting plane p. If the face f lies in the plane p, we can discard it because the division plane with the same parameters is already included in the corresponding branch of the tree. Otherwise, we move to the left subtree if $F(c) < 0$ and to the right subtree if $F(c) > 0$, where c is the center of the face f. When we reach an empty leaf, we can insert the plane that contains the face f.

Inserting a face into the tree based on the position of its center is a simplification that leads to smaller and simpler BSP trees. The center is a better representation of the face than its vertices, as each vertex is shared by multiple faces and so the probability of a vertex lying in a plane defined by another face increases. We can afford such a simplification thanks to the connectivity of the input mesh - the inserted face has neighbors that will probably be inserted in the neighboring cells of the BSP tree. However, if the inserted faces are big when compared to the size of the cells of the BSP tree in the corresponding part of the scene, some of the cells may not be correctly divided due to this simplification. This can cause small parts of the scene to be misclassified as an incorrect type of material. We call this approach the simplified BSP method (S-BSP).

To obtain an exact solution, the face f has to be inserted into all the subspaces it intersects. Instead of the center of the face f, all its vertices have to be evaluated. If all the vertices lie on the same side of the splitting plane p, the whole face lies on that

side and we can proceed with the search in the corresponding subtree. Otherwise, we have to continue the search in both subtrees of the current node. That increases the complexity of the insertion algorithm and also leads to the creation of bigger BSP trees, as some of the faces cause the tree to grow by more than one node. This approach will be referred to as the complete BSP method (C-BSP).

Another simplification of our algorithm applies to the way the material properties are assigned to the leaves of the BSP tree when a new plane is inserted. In the close proximity of the inserted plane, it is safe to assume that the space to the left of the plane belongs to the inside of the region delimited by the input triangular mesh, while the space to the right belongs to the outside. However, for nonconvex regions this may not hold in the regions located farther away.

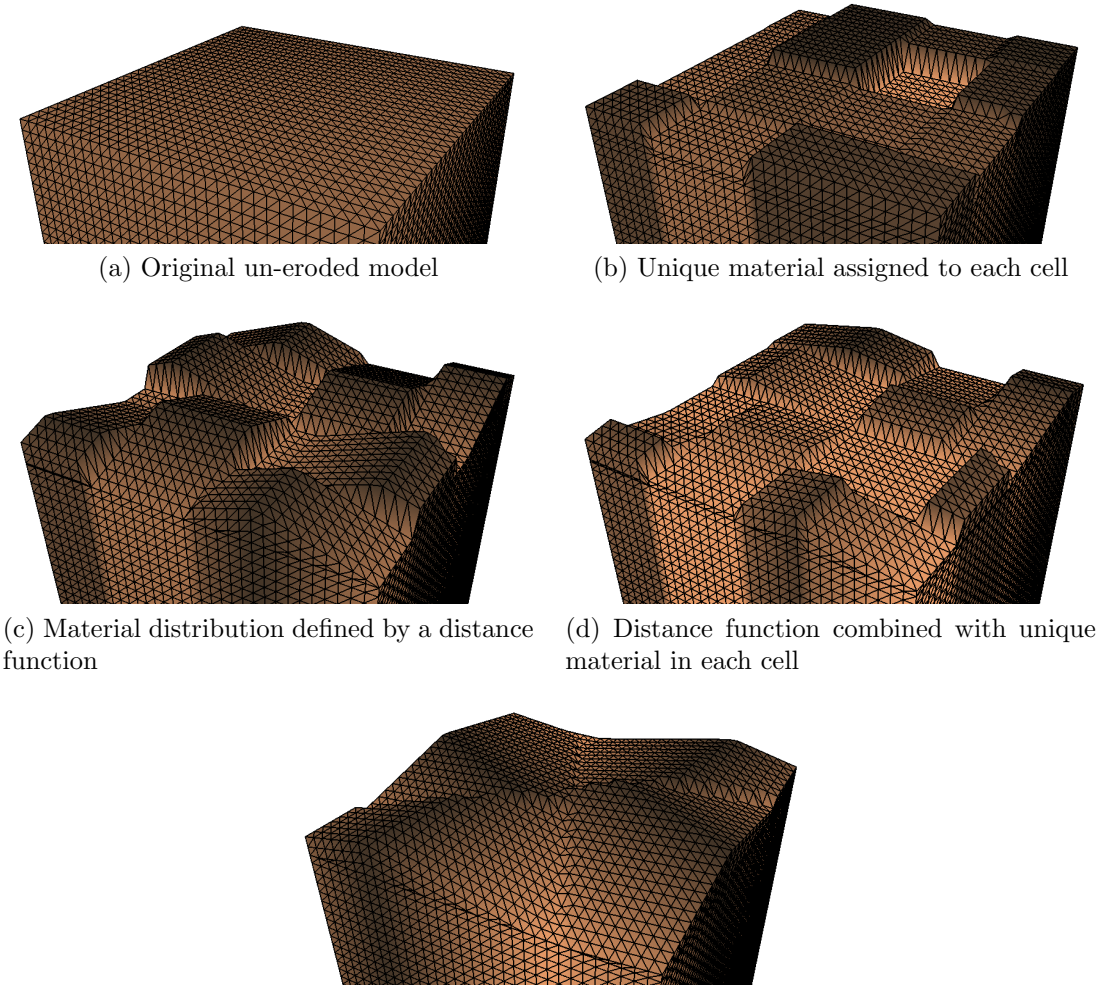
The introduced simplifications can cause an incorrect classification of the material properties, but as we show in the following section, our approach gives satisfactory results.

11.6 Results and Experiments

11.6.1 Splitting Planes

We have tested the proposed approach in various scenarios to test its applicability to the problem of material spatial distribution. Figure 11.10 shows an example of the basic BSP material definition. Figure 11.10a shows an initial un-eroded model; the same initial model was used in all the erosion simulations presented in Figure 11.10. The scene has been divided into sixteen vertical spatial cells; each one has been assigned a unique random material. The darker color of the mesh marks the regions made of a tougher material; the lighter the color, the less durable the material is. As can be seen in Figure 11.10b, the parts of the mesh made of a less durable material are being eroded much faster than the regions made of a tougher one.

The BSP material definition method can be extended to support the definition of a gradually changing material. The simulation of gradually changing material is very important in dynamic simulations such as erosion scenarios. It allows the representation of materials blending into each other, such as sand and pebbles at a river bank. We achieve the effect of gradually changing material by the addition of a distance function. The material properties are computed based on the value of d defined by Equation 11.1. Figure 11.10c shows an example of gradually changing material. The hardest material is assigned to the vertices lying on the division planes, as we move farther from the planes, the material gets softer. The two aforementioned approaches can be combined as seen in Figure 11.10d. Each spatial cell has been assigned a random unique material. With the growing distance from the division plane, the material gets softer. Figure 11.10e shows an example of BSP using general planes combined with a distance function.



(a) Original un-eroded model

(b) Unique material assigned to each cell

(c) Material distribution defined by a distance function

(d) Distance function combined with unique material in each cell

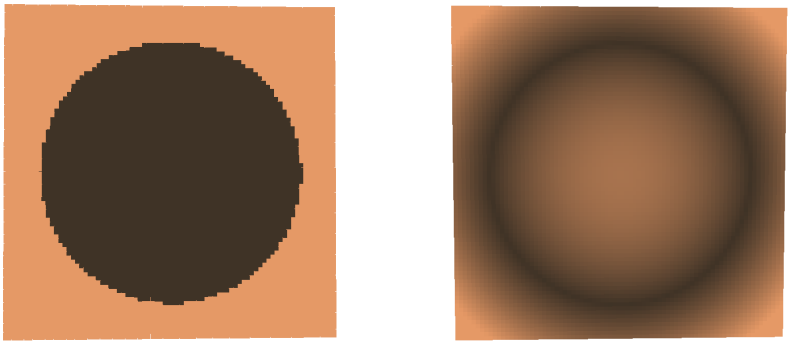
(e) Material defined by splitting planes in general positions

Figure 11.10: Material definition via BSP approach. Dark brown marks regions made of hard material; light brown marks soft material

11.6.2 Implicit Splitting Surfaces

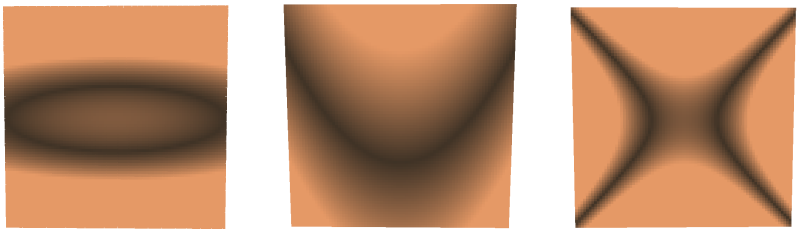
The generalization of the proposed method to allow the use of implicit splitting surfaces gives us a powerful tool for material definition. A single implicit surface can replace many splitting planes that would be necessary to create the same spatial subdivision. Similar to the base method, a distance function can be used to mimic a gradually changing material.

Figure 11.11 shows an example of the space partitioning by an implicit surface - a sphere. Figure 11.11a shows a scene with constant material properties in each subspace. In Figure 11.11b, a distance function is defined to simulate the material becoming softer farther from the boundary. As the 3D visualization of the space partitioning could become unclear for more complex scenes, 2D slices of the volume will be used to demonstrate the distribution of the material throughout this Section.

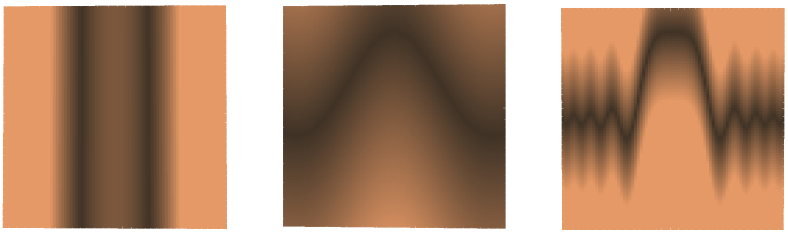


(a) Constant material properties (b) Gradually changing material

Figure 11.11: 2D slice of a scene subdivided by an implicit surface - a sphere



(a) Ellipsoid (b) Paraboloid (c) Elliptic hyperboloid



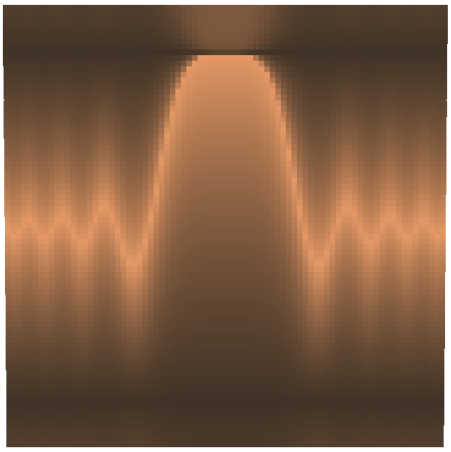
(d) Cylinder (e) Sin function (f) Sinc function

Figure 11.12: Examples of splitting functions combined with a distance function

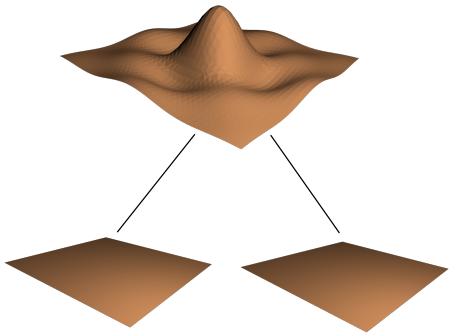
The 2D slices are created by sampling the BSP tree at regular intervals. Figure 11.12 shows other examples of implicit surfaces that can be used as splitting functions. A simple example of a space partitioning consisting of three implicit splitting functions is captured in Figure 11.13. Figure 11.13a illustrates the material distribution in a 2D slice of the scene, while Figure 11.13b shows the corresponding BSP tree.

An example of a more realistic scene is shown in Figure 11.14. The scene simulates the layered nature of a terrain - the bottom layers represent hard layers of bedrock, while the upper layers are defined by a normalized sinc function and imitate a hilly landscape. Figure 11.14a shows the material distribution in the middle slice of the scene. Figure 11.14b shows a single mesh that was sculpted from an initial cube model by an erosion simulation with the defined material properties.

Figure 11.15 captures the material distribution of a rocky scene where rocks and pebbles of varying size and durability are covered with easily erodible sand. The sand

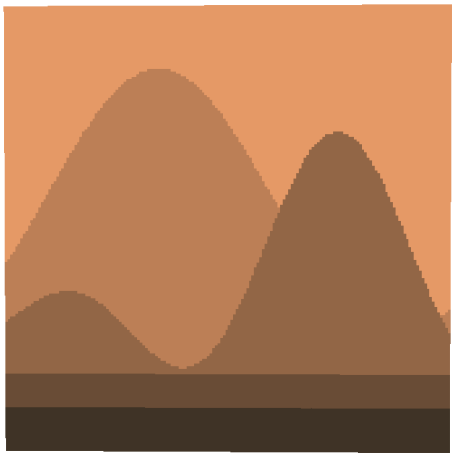


(a) 2D slice of material distribution

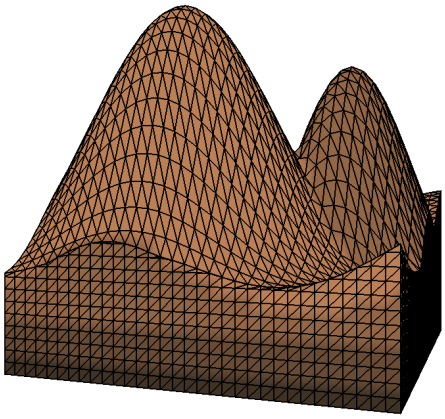


(b) BSP tree defined by implicit functions

Figure 11.13: BSP tree defined by implicit splitting surfaces



(a) Material distribution



(b) Mesh sculpted from a cube by an erosion simulation

Figure 11.14: Hills example. Dark brown marks regions made of hard material; light brown marks soft material

is gradually eroded away and the underlying stones are being revealed (Figure 11.16). However, this sort of material distribution is not very suitable for the BSP approach using implicit splitting functions. As we generally do not need to subdivide the space inside the individual spheres or ellipsoids that define the stones, the BSP tree would often degenerate into a list.

11.6.3 Automated Generation of the BSP Tree

To examine the method for the automated generation of a BSP tree, we used a volumetric model of a cave complex[1] with the resolution of $120 \times 120 \times 60$ voxels. The

[1]The volumetric model was adapted from a model distributed as a part of MagicaVoxel software (`https://voxel.codeplex.com/`)

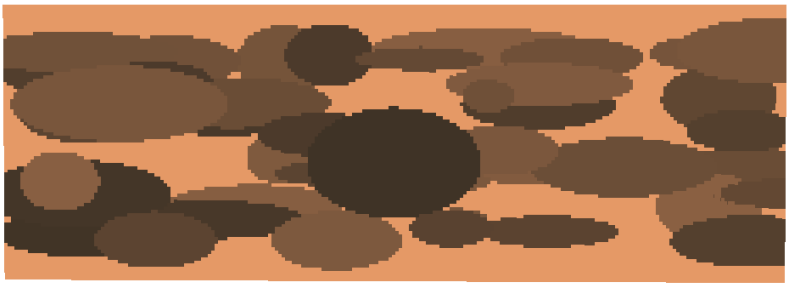
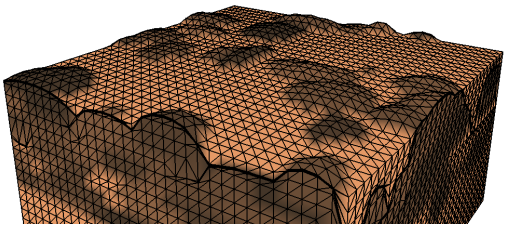
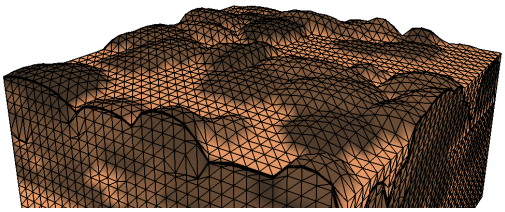


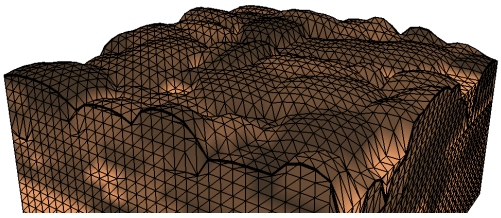
Figure 11.15: Material distribution of a rocky scene



(a) The rocks are covered with sand



(b) The sand gets eroded



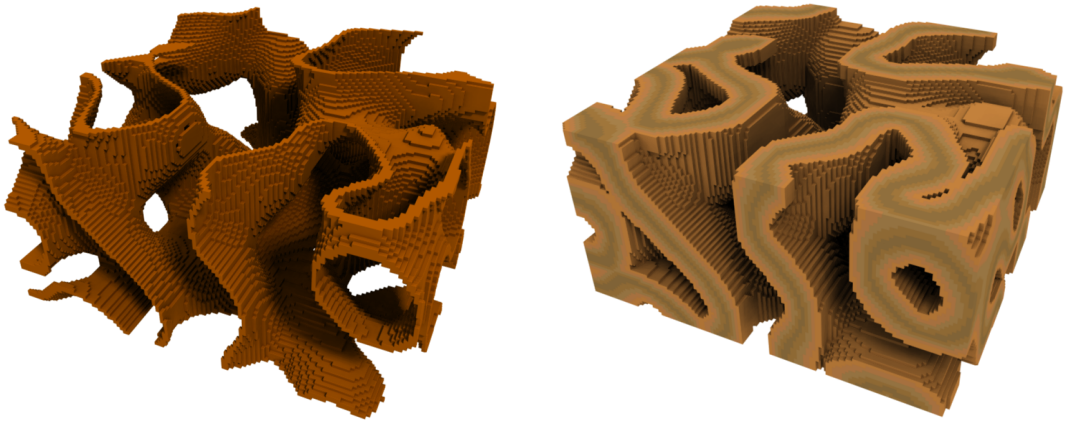
(c) The rocks are revealed

Figure 11.16: Rocky scene. Dark brown marks regions made of hard material; light brown marks soft material

model is shown in Figure 11.17a. We expanded the model using dilation to represent the changing material. The voxels that were full in the base model were assigned the hardest material. The voxels created by the dilation were assigned a material based on the distance from the full voxels; the farther the voxel, the softer the material, as illustrated in Figure 11.17b.

The expanded volumetric model was used as an input for the marching cubes algorithm [LC87]. We selected four material thresholds and extracted the corresponding isosurfaces. The triangular meshes representing the individual isosurfaces were afterwards used to generate the BSP tree following the algorithm described in Section 11.5. The four meshes combined consisted of 684,892 faces.

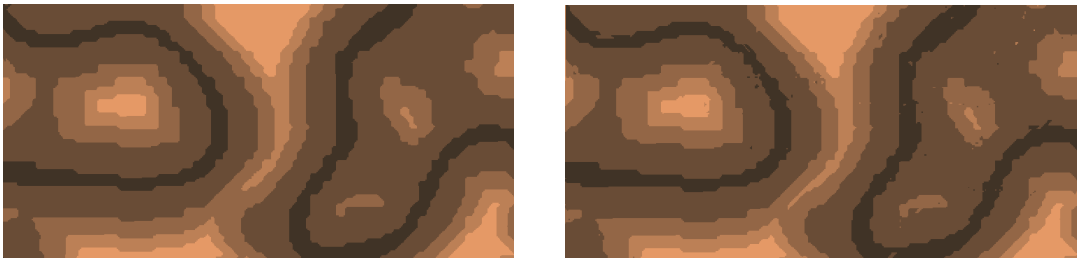
Two approaches were used to generate the BSP tree. Figure 11.18a shows the outcome of the complete BSP (C-BSP) method where each face is inserted into all the tree branches it intersects. Figure 11.18b shows the outcome of the simplified BSP (S-BSP) method where the faces are inserted into the BSP tree into a single branch based on the position of the center of the face. As can be seen in the pictures, the simplified method can cause misclassification of small regions of the scene. It is up to the user to select the appropriate method for their simulation, depending



(a) Volumetric model of a cave complex

(b) Volumetric model of a cave complex created by dilation of the original model

Figure 11.17: Volumetric model of a cave complex. The lighter the color, the softer the material



(a) The complete BSP method (C-BSP)

(b) The simplified BSP method (S-BSP)

Figure 11.18: Material distribution of a single slice of a cave complex sampled from the generated BSP tree. The lighter the color, the softer the material

on whether or not their algorithm can handle the small disturbances in the material distribution.

We tested our approach on a medical CT-scanned data of a tooth to verify the applicability of the approach. Such a dataset can be used, e.g., in a medical simulation to study the erosion processes on the surface of the tooth and the subsequent creation of dental caries. The Tooth dataset [Eng00] consists of $256 \times 256 \times 161$ voxels, each voxel is represented as a 16-bit value. Visualization of the original dataset is shown in Figure 11.19. A tooth consists of three different materials. The hardest material forms the topmost part of a tooth and is called enamel. The part below enamel is called dentin and consists of a softer material. The softest material can be found in the middle of a tooth in the dental pulp.

We identified the thresholds that separated the aforementioned parts in the Tooth dataset and used the marching cubes algorithm [LC87] to extract the corresponding isosurfaces. The extracted isosurfaces consisting of 325,704 faces were used to create the BSP tree, similarly to the previous example. The results of the C-BSP method are shown in Figure 11.20a, the results of the S-BSP method are captured in Fig-

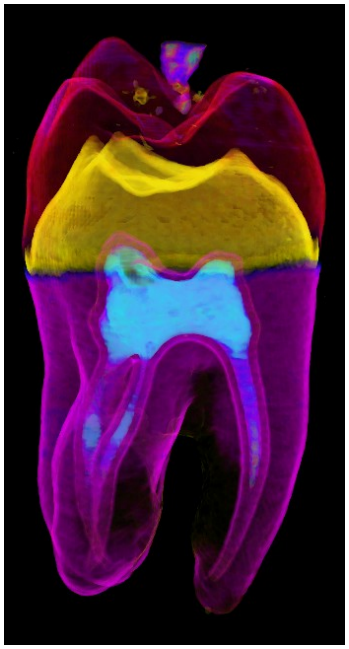


Figure 11.19: Visualization of the Tooth dataset [Eng00]

ure 11.20b. As can be seen in Figure 11.20b, the S-BSP method gives considerably inferior results for this dataset.

The introduced simplifications have a big influence on the efficiency of the spatial division. The removal of the faces lying in the planes that are already present in the corresponding branch of the BSP tree reduces the number of inserted faces by more than 80%. Furthermore, unlike the simplified generation of the BSP tree based on the centers of the faces, it does not affect the quality of the resulting space partitioning. Table 11.1 summarizes the number of faces of the extracted meshes, as well as the number of faces inserted in the BSP tree for both the S-BSP and the C-BSP approach. The size of the BSP tree and its height are average values based on multiple runs with random order of inserted faces. The difference in the number of inserted faces for S-BSP and C-BSP is not very significant and so the C-BSP method can be recommended for most scenarios.

Table 11.1: Properties of the BSP trees generated for the presented results. C-BSP: complete BSP, S-BSP: simplified BSP

	Cave (Fig. 11.18)	Tooth (Fig. 11.20)
Extracted faces count	684,692	325,704
C-BSP face count	115,257	48,196
S-BSP face count	95,511	42,574
C-BSP tree height	63	60
S-BSP tree height	62	58

The memory requirements of the original volumetric datasets and the generated BSP trees are compared in Table 11.2. Each node of the BSP tree takes 24 bytes:

(a) The complete BSP method (C-BSP)

(b) The simplified BSP method (S-BSP)

Figure 11.20: Material distribution of four slices of the Tooth dataset sampled from the generated BSP tree. The lighter the color, the softer the material

4 floats (32-bit) for the coefficients of the dividing plane and 2 pointers (32-bit) to children nodes. The material type is stored only in the leaves of the tree and so its contribution to memory requirements is insignificant. Table 11.2 insinuates the kind of scenarios for which the proposed approach is suitable. The Cave complex example (Figure 11.17b) consists of very thin layers of different materials. If we want to represent the scene without loss of detail, we need to extract several isosurfaces and the resulting BSP tree quickly grows in size. On the other hand, in the Tooth dataset (Figure 11.19) we can distinguish only three types of material. The BSP approach reduces the memory requirements for this dataset to approximately 5% of the requirements of the original volumetric data.

11.7 Method Summary and Future Work

In this Chapter, we have proposed a multiple material definition method based on the use of binary space partitioning (BSP). A BSP tree subdivides the scene into cells by splitting planes or general implicit surfaces and the material for each of the cells is

Table 11.2: Memory requirements of the presented results (in bytes). C-BSP: complete BSP, S-BSP: simplified BSP

	Cave (Fig. 11.18)	Tooth (Fig. 11.20)
Volume data	864,000 B	21,102,592 B
C-BSP	2,766,168 B	1,204,900 B
S-BSP	2,292,264 B	1,064,350 B

defined separately. The approach also supports the definition of a gradually changing material through the use of a distance function. To simplify the creation of the BSP tree, a method for automated generation of the tree from input volumetric data has been proposed. An isosurface extraction method is used to extract a triangular mesh which is used to construct the BSP tree. Simplifications have been introduced that have great impact on the size of the constructed tree, at the cost of some regions possibly being misclassified as an incorrect type of material. A complete BSP method with slightly higher memory requirements has also been proposed for the sake of simulations that cannot handle the material disturbances the simplified approach can cause.

The proposed solution is suitable for use in larger scenarios composed of several types of material where the memory requirements of the volumetric approach would be unacceptable. The memory consumption of the BSP approach depends on the number and complexity of the blocks of different materials, not on the size or complexity of the scene itself. However, if the material layers are too thin or too variable, the memory requirements of the created BSP tree can exceed the requirements of the volumetric approach. For future work, it could be possible to address this drawback by a method for automated detection of general splitting surfaces.

Chapter 12

Erosion-Inspired Simulation of Aging for Deformation-Based Head Modeling

Erosion simulation is usually considered in relation with terrain evolution or simulation of aging effects on individual natural or artificial objects. However, erosion simulation can also be used to simulate other phenomena in other fields of research. We have proposed a method that simulates aging of a human face using erosion-inspired approach. The approach is very simple and requires no training data, unlike most of the existing methods that focus on this problem (see [Sko+17] for a short overview of related existing methods). The approach is particularly helpful in scenarios where fast and simple aging simulation is required. An example of such scenario can be, e.g., the creation of 3D identikit using deformation-based modeling, such as the approach proposed by Martínek and Kolingerová [MK14].

The proposed method simulates aging of 3D triangular head models by creating wrinkles using an erosion-inspired approach. The vertices of the model are assigned an erosion factor that controls the deformation of the surface. A positive value of the erosion factor is assigned to the regions that should be enlarged to mimic a deposition process, while a zero erosion factor suggests that the given region should be preserved. Negative values of the erosion factor could be used to mark the regions that should be reduced, such as very deep wrinkles. However, negative values are not used in the proposed solution as they could cause an unrealistic appearance of the final model. After the values are assigned, the aging is simulated by moving the mesh vertices in the normal direction to a distance defined by the erosion factor.

The proposed solution is designed to simulate aging processes on 3D head models created using deformations. Deformation-based modeling uses a base input model and creates derived models by specifying the deformations, without altering the number of vertices or the mesh connectivity. As the mesh topology is the same for all the derived models, it is only necessary to define the erosion values for the base model and the same aging deformation can be afterwards used for all the derived meshes.

The proposed solution consists of the following steps:

1. Remeshing of the input mesh (optional)
2. Detection of the affected areas based on user-defined wrinkle endpoints
3. Local subdivision of the mesh
4. Computation of the erosion factor
5. Mesh deformation

12.1 Remeshing

The wrinkles on the face usually follow the direction of one of the principal curvatures. If the initial triangulation of the base model does not follow the principal curvatures, it may be reasonable to use an appropriate remeshing technique (e.g., a remeshing method proposed by Alliez et al. [All+03]) to create a new base model that will yield better results in the aging simulation.

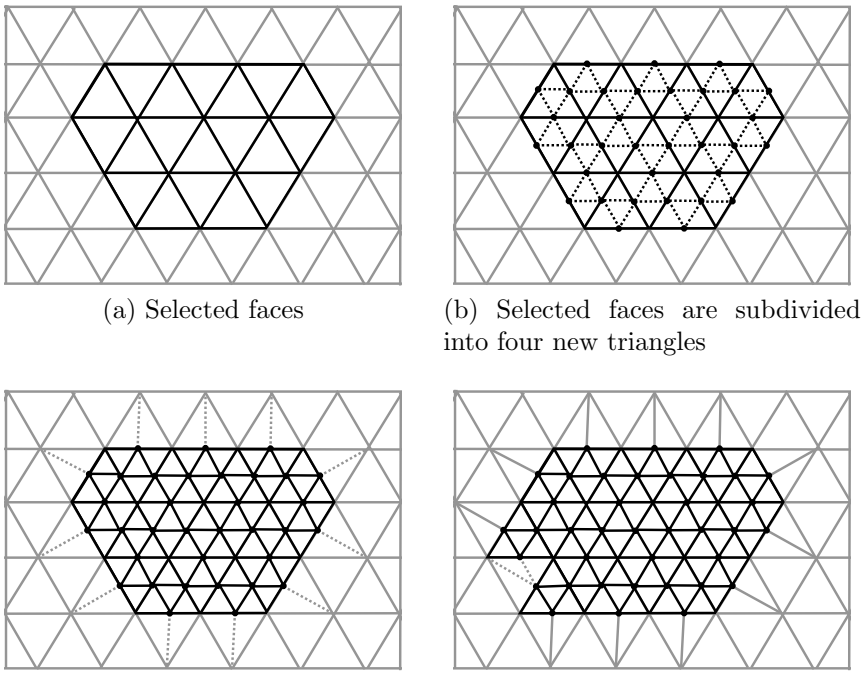
12.2 Detection of the Affected Regions

The affected regions are selected based on user-specified wrinkles. The user selects the endpoints of each wrinkle that should be imprinted on the model. For longer curved wrinkles, it is necessary to select also the vertices where the wrinkles substantially change direction. The user also influences the size of the region affected by the wrinkle by specifying a wrinkle effect radius r.

For each wrinkle, the remaining vertices that belong to the given wrinkle are detected automatically. For each wrinkle segment, a plane p is created that passes through the endpoints V_1 and V_2 and the average of the normals at the endpoints. To find the path between the vertices V_1 and V_2, the mesh is walked from the vertex V_1 in the direction towards V_2. At each step of the algorithm a vertex V_i is added to the path such that it is a neighbor of the last vertex in the path and is it the closest vertex to the plane p.

12.3 Local Subdivision of the Mesh

The base head model is usually not detailed enough for modeling of such fine features as the wrinkles and so a local subdivision of the mesh is used to refine the mesh in the affected regions. To select all the faces that belong to the affected region, the wrinkle paths detected in the previous step are used. For each vertex in the wrinkle path, incident faces up to the wrinkle effect radius (specified by the user)



(a) Selected faces

(b) Selected faces are subdivided into four new triangles

(c) Neighbors of the selected faces are repaired: case 1

(d) Neighbors of the selected faces are repaired: case 2

Figure 12.1: An example of the local subdivision scheme

are selected. A set is constructed as a union of the subsets belonging to each vertex; the set contains all the faces that can be affected by the aging simulation.

For the selected faces, a simple subdivision scheme is used (see Figure 12.1). For each selected face (dark faces in Figure 12.1a), three new vertices are created at the midpoints of its edges and four new faces are created (as captured in Figure 12.1b). Afterwards, the faces that do not belong to the affected regions but are neighbors of at least one of the affected faces have to be repaired, to preserve the topology of the mesh. If a face neighbors with a single face from the selected set, it is split into two (dotted lines in Figure 12.1c mark the face splits); if a face neighbors with two faces from the selected set, it has to be divided into three new faces (see Figure 12.1d, dotted lines mark the face split).

12.4 Erosion Factor

Erosion factor describes the magnitude of the deformation at a given vertex of the mesh. To locate the deformation only to the regions specified by the user, a positive erosion factor is assigned to all the vertices in the affected regions while the rest of the vertices is assigned a zero erosion factor. For each vertex V in the wrinkle path, incident vertices up to the wrinkle effect radius are selected and assigned a positive erosion factor f_i by a Gaussian function

$$f_i = \frac{e^{-\frac{\|V-V_i\|^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}, \tag{12.1}$$

where $\|V - V_i\|$ is the distance of the point V_i from the vertex V in the wrinkle path and σ is the variance, which is calculated as

$$\sigma = \frac{r^2}{3}, \tag{12.2}$$

where r is the user-defined wrinkle radius. The same approach is used if the user also specified the vertices around which the mesh should be enlarged to simulate the effects of weight gain or sagging skin.

The final erosion factor f_{V_i} is then obtained by defining the position of the wrinkles. For each vertex V_i within the affected area, i.e., each vertex that was assigned a positive erosion factor in the previous step, the erosion factor f_{V_i} is calculated as

$$f_{V_i} = f_i\sqrt{\|V_c - V_i\|}, \tag{12.3}$$

where V_c is the point from the wrinkle paths that is the closest to V_i and $\|V_c - V_i\|$ is the distance between the points V_c and V_i.

12.5 Mesh deformation

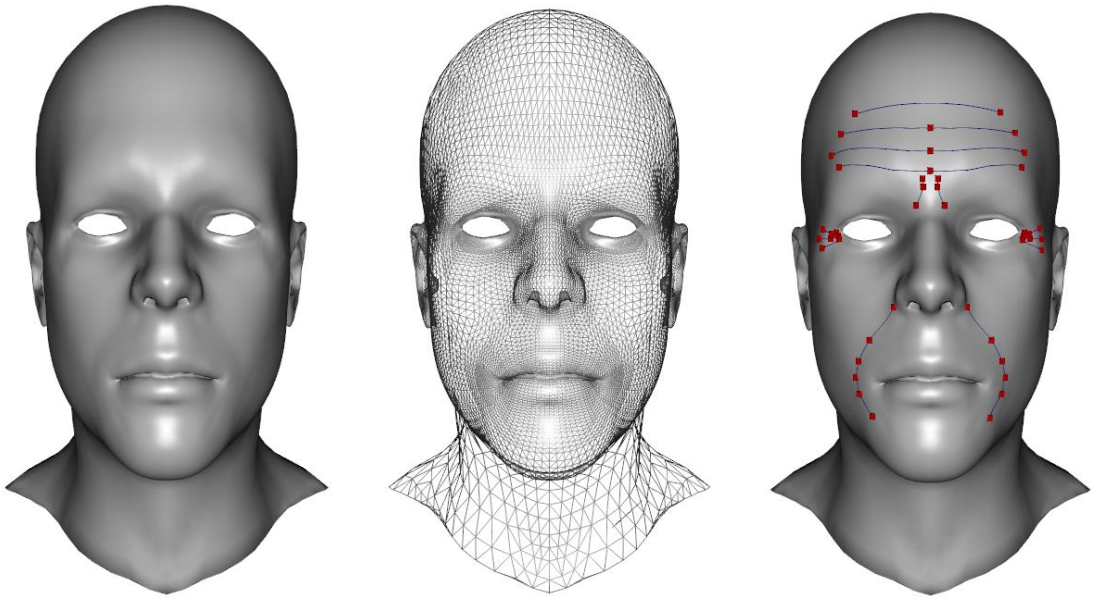
Finally, new positions P_i of vertices of the mesh are calculated as

$$P_i = V_i + \frac{f_{V_i}\mathbf{n}_i}{a}, \tag{12.4}$$

where V_i is the original position of the vertex, f_{V_i} is the erosion factor at the vertex V_i and $\mathbf{n}_i$ is the normal at the vertex V_i. Symbol a stands for a factor influencing the strength of the deformation. In our simulations, we have set a to 10 based on our experimental results. If a stronger wrinkle effect is required, the mesh deformation can be used iteratively until the desired effect is achieved.

12.6 Results

In this section, the results of the proposed method are shown. A head model [MK14] consisting of 10,925 vertices and 21,746 faces was used as the base model (see Figure 12.2). Figure 12.2a shows the base model, Figure 12.2b shows the wireframe of the model and Figure 12.2c captures the wrinkle control points that were defined by the user (red squares), as well as the wrinkle lines (blue lines) detected automatically by the algorithm described in Section 12.2.



(a) Base model

(b) Wireframe

(c) Control points defined by the user (red squares) and the detected wrinkles (blue lines)

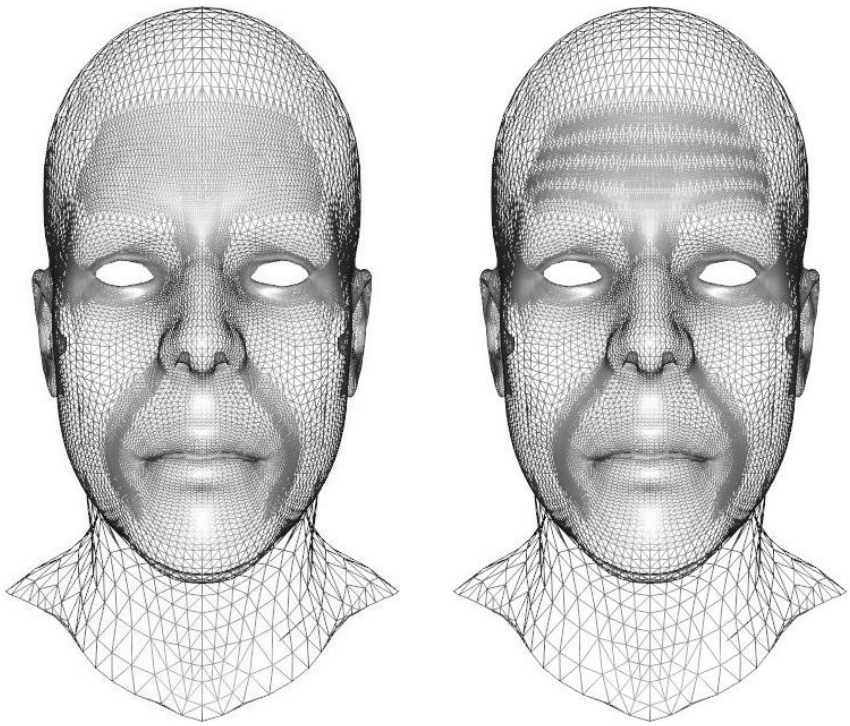
Figure 12.2: An example of a base head model

The base mesh is clearly not detailed enough in the affected regions, so it is necessary to subdivide the mesh using the local subdivision approach as discussed in Section 12.3. The mesh subdivision was applied twice to achieve better results. The first subdivision (Figure 12.3a) was applied around the detected wrinkle lines with a radius r defined by the user. The second subdivision (Figure 12.3b) was applied with a radius of $r/3$.

The erosion factor was calculated for each vertex of the mesh following the algorithm presented in Section 12.4. Each vertex was then moved in the direction of the normal to a distance defined by the erosion factor, as discussed in Section 12.5. The results of the aging simulation are shown in Figure 12.4.

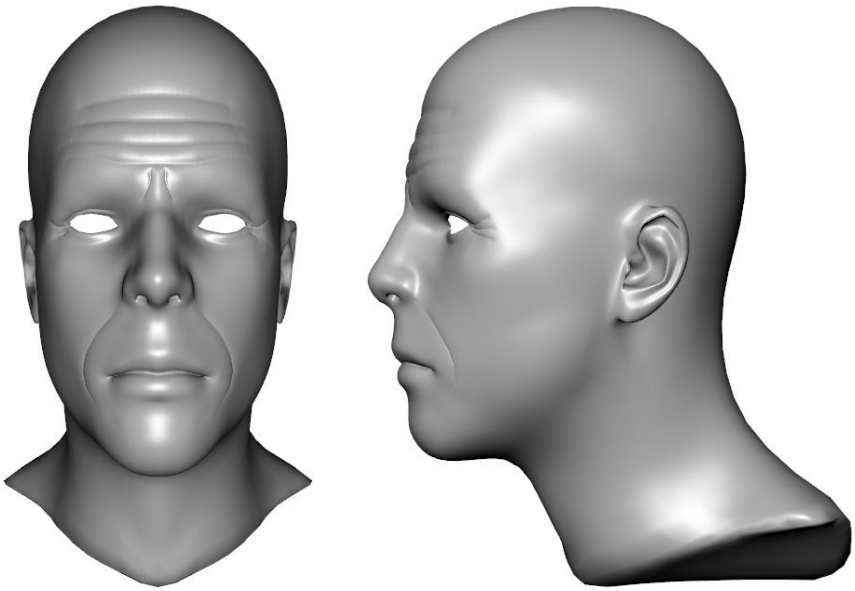
As the deformation-based modeling does not change the mesh topology, the same erosion factors can be used for any model obtained from the base model by a method, such as the approach presented in [MK14]. The method [MK14] was used to create several models derived from the base model captured in Figure 12.2. Figure 12.5 shows the result of the aging simulation when the erosion factor calculated for the base model was applied to the derived models.

To the best of our knowledge, there are no other approaches that would be targeted at aging simulation for models created by deformation-based modeling. The approach by Wang et al. [WWY06] is capable of deforming the model to imitate wrinkles, however, the results are not very visually plausible (see Figure 12.6). A more recent approach by Kim et al. [Kim+15] uses sketch-based modeling to simulate aging. Their approach can create plausible aged models (see Figure 12.7) but the wrinkle sketches have to be drawn for each input model, unlike our approach that



(a) First iteration of subdivision (b) Second iteration of subdivision

Figure 12.3: Local subdivision of the model in the areas affected by the wrinkles



(a) Front view (b) Side view

Figure 12.4: Result of the aging simulation of the base model

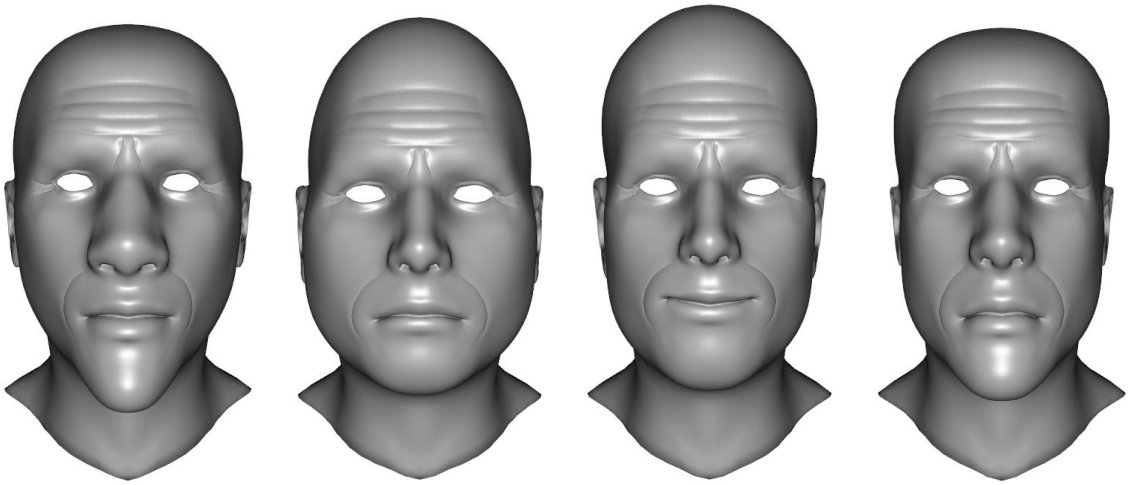


Figure 12.5: Results of the aging simulation of models derived from the base model using deformation-based modeling

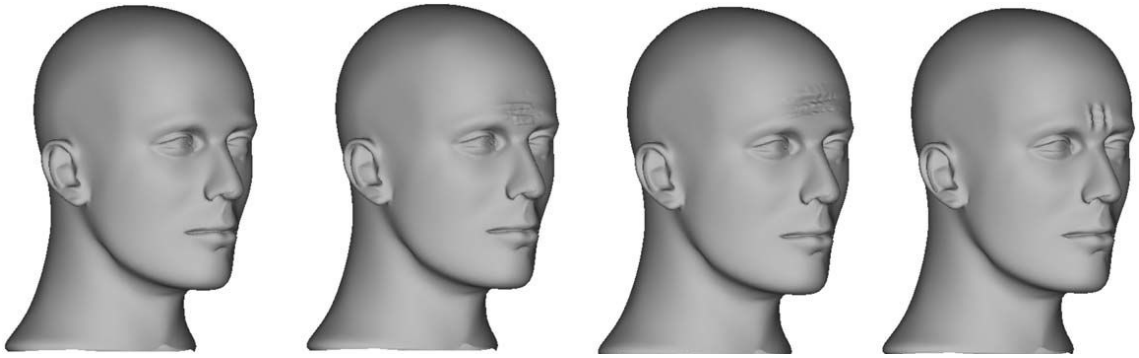


Figure 12.6: Input head model and forehead wrinkles created by the approach by Wang et al. [WWY06]

allows the replication of the computed erosion factors for models derived from a base model. Furthermore, the approach by Kim et al. [Kim+15] requires a wrinkled example model and simulates the wrinkles using normal maps, while our approach actually deforms the mesh to achieve better results when viewed from an angle.

12.7 Automatic Detection of Control Points

Our solution requires a manual definition of wrinkle endpoints. As our method is targeted to be used on models created by deformation-based modeling, it is only necessary to define the endpoints once for each base model. However, to be able to use the method on any face model more easily, a method for automatic detection of feature points would be helpful.

We have participated on the method proposed by Prantl et al. [Pra+17] that detects feature points and regions automatically based on curvature values of the input head model. The method is designed to detect feature points necessary

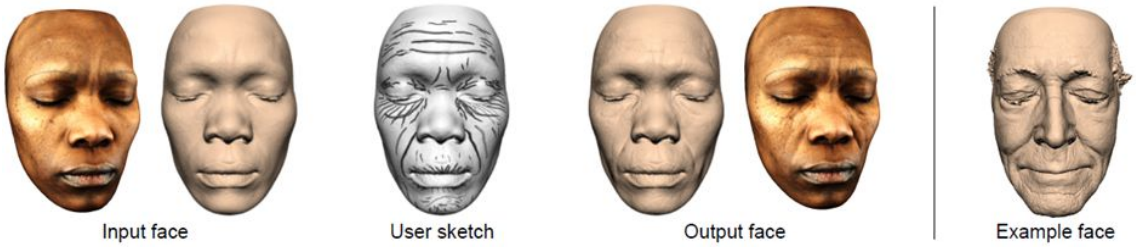


Figure 12.7: Sketch-based simulation of aging as proposed by Kim et al. [Kim+15]

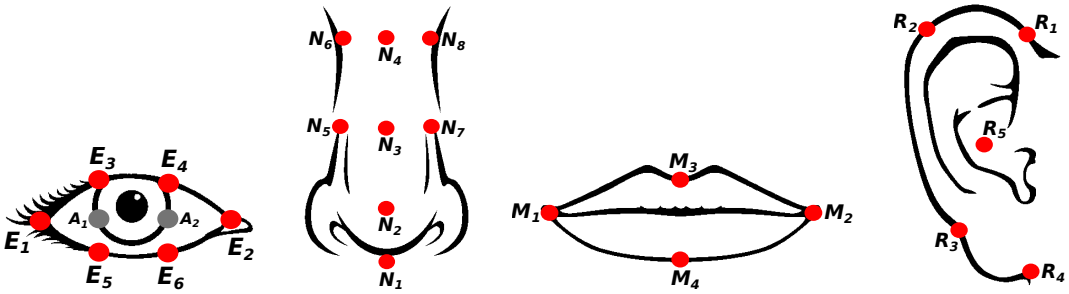


Figure 12.8: Feature points of eye, nose, mouth and ear

for deformation-based modeling, such as the approach proposed by Martínek and Kolingerová [MK14]. The feature points that the method is capable of detecting are captured in Figure 12.8. The comparison of manually inserted feature points and the points detected by the proposed approach are shown in Figure 12.9.

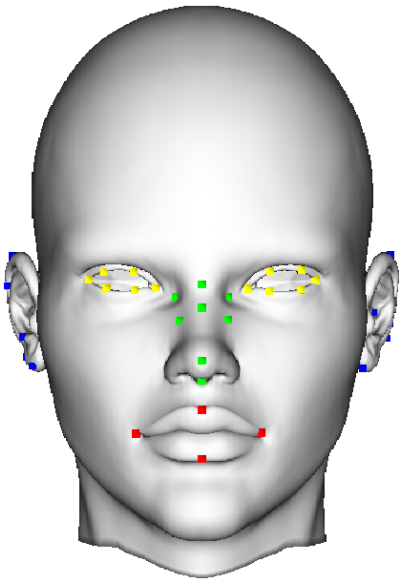
It can be seen that the positions of the detected and the manually inserted points slightly differ. However, as the intended use of the detected points is for the use in the deformation-based modeling, the accuracy of the detection is sufficient.

12.8 Method Summary and Future Work

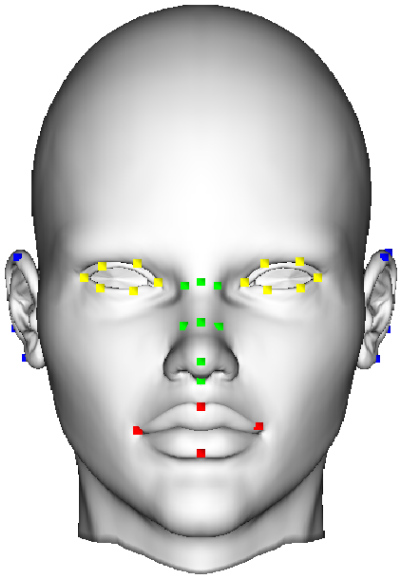
The method for the simulation of aging of 3D triangulated head models described in this Chapter uses user-defined wrinkle endpoints to detect wrinkle lines and imprints the wrinkles onto the mesh using an erosion-inspired deformation approach. The method is especially useful for deformation-based head modeling approaches, where new head models are created by applying deformations to a base mesh. These deformations do not change the topology of the mesh and so the same erosion factors can be used for the definition of the wrinkles as the ones that were used for the base model.

The proposed solution concentrates on the creation of distinct wrinkles that alter the shape of the head when viewed from an angle. The solution could be enhanced by subtle wrinkles by applying an appropriate texture.

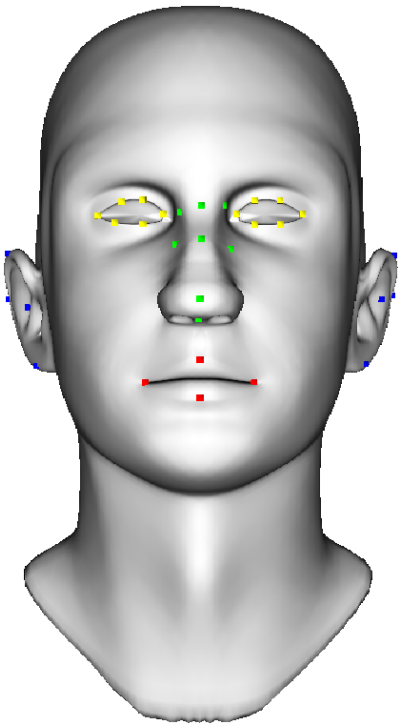
As future work, it would be possible to reduce the necessary user interaction by detecting the wrinkle endpoints automatically. A possible approach is to use an



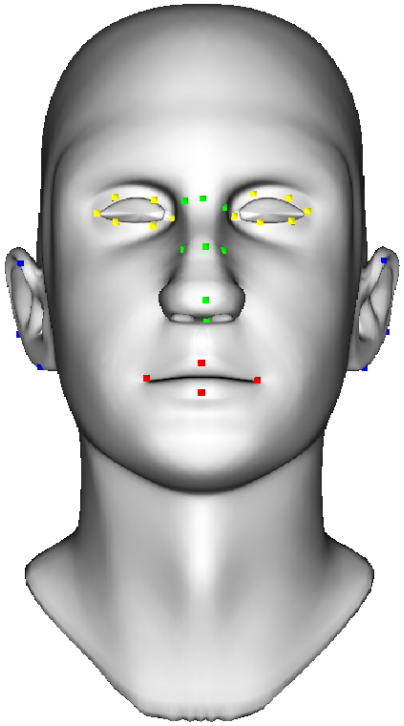
(a) Model 1; manually inserted feature points



(b) Model 1; automatically detected feature points



(c) Model 2; manually inserted feature points



(d) Model 2; automatically detected feature points

Figure 12.9: Comparison of manually inserted and automatically detected feature points

automatic detection of feature points, such as the approach mentioned in Section 12.7 and to use the detected feature points to estimate the wrinkle positions.

Chapter 13

Summary of Contributions

This thesis presented our contributions to the solution of the problem of erosion simulation in the field of computer graphics. We have proposed several approaches for weathering and hydraulic erosion simulation. Our main focus was to create methods that would be fast and simple to use and would produce visually plausible models of both eroded terrains and also complex general objects. We chose to represent the eroded objects as triangular meshes. This data structure meets our requirements due to its adaptability and possibility to model any complex features of the eroded object. Its use is also widespread in the field of computer graphics and so the use of this data structure in the proposed methods simplifies the use of the methods in wide range of scenarios.

In Chapter 8, we have explored the capabilities and disadvantages of hydraulic erosion simulation using triangular meshes as the representation of the eroded object and Smoothed particle hydrodynamics (SPH) for the fluid simulation [SKB15]. The method focuses on the hydraulic erosion of simple terrains but also shows the capabilities of the method to work on fully 3D models that could not be represented using simpler data structures, such as the height field.

In Chapter 9, we have improved the base method by taking into account the influence of local shape of the eroded objects [SKV19]. The speed of the erosion is driven by the local mean curvature value. This results in protruded regions of the object being eroded faster while the gaps are eroded at much slower rate. This simple and fast method is capable of producing visually plausible eroded scenes not only for hydraulic erosion simulations but it is also capable of simulating weathering effects. The method runs at almost interactive frame rates without any special effort being put to optimization of the method.

The use of the triangular mesh data structure brings new problems to the erosion simulation. Strong erosion or deposition can cause splitting or merging of two parts of the mesh, creating a topology inconsistency. The inconsistency impedes the correct erosion simulation and has to be repaired by a mesh repair technique. In Chapter 10, we have presented an accurate mesh repair method [SKB18] to solve this problem. The proposed method uses a local geometry-based approach to repair intersecting meshes directly through a careful classification of the cases that could

result from a numerical imprecision of the floating point arithmetic. The method requires input meshes to be free of nearly degenerate triangles. This condition cannot be satisfied directly when working with meshes changing due to erosion but can be enforced by removing the nearly degenerate triangles during preprocessing step before applying the repair method.

Another topic of our interest is the representation of multiple material erosion scenes, as it is necessary for the simulation of a complex realistic landscape. We have examined several possible approaches to efficient material description in [SK15] and in [SK16] and we have described our findings in Chapter 11. Our approach is based on binary space partitions (BSP). The BSP subdivides the space into convex regions made of homogenous material. This data structure is then used during the erosion simulation to dynamically assign the material properties to the eroded regions of the mesh. As the definition of the BSP tree by hand would be troublesome, we have proposed a method for the automated detection of the BSP splitting planes from the input volumetric data. An isosurface extraction method is used to extract a triangular mesh that is used in the BSP construction algorithm.

In Chapter 12, we have shown a possible application of erosion principles to other real-life problems. The proposed approach [Sko+17] uses a simple erosion-inspired approach to simulate aging on 3D head models. The method requires no training data and generates distinct wrinkles based on control points defined by the user. The method is particularly useful for the use in deformation-based modeling, where new models are created from the base model using deformations that do not damage the mesh topology or connectivity. For such models, the control points for the wrinkle generation only have to be manually selected for the base model and can be then used for all the derived models without any additional effort from the user.

The proposed methods address the open problems in erosion simulation in computer graphics. The methods are capable of simulating hydraulic erosion and weathering on terrains with complex concave features, such as caves, tunnels or overhangs, as well as on general objects, such as statues or other man-made objects. We have proposed possible solutions to the problems that are caused due to the use of the triangular mesh data structure. We have also shown an example of application of erosion approaches in other fields of research.

For future work, an obvious goal would be to merge all the proposed approaches into a complex unified framework that could be easily used to simulate weathering and hydraulic erosion on objects represented as triangular meshes. Another possible avenue could be to optimize the methods and to implement them on the GPU to achieve better performance.

Chapter 14

Conclusion

Erosion simulation is a very important topic of the modern computer graphics. Erosion simulation allows to create plausible terrains or eroded objects without the need to employ exhaustive software modeling tools. The interactive state-of-the-art approaches are not capable of simulating fully 3D phenomena, while the fully 3D solutions cannot be run at interactive frame rates.

We have proposed several methods for the hydraulic erosion and weathering simulation. Our solutions combine the eroded objects represented as a triangular mesh with a fluid simulated as a particle system. This approach is capable of simulating fully 3D scenes containing features such as caves or overhangs with lower memory requirements than the existing volumetric solutions, while running at almost interactive rates.

This thesis supports the validity of our choice of the triangular mesh as the representation of the eroded objects. The data structure allowed us to create fast and simple methods for erosion simulations, however, the use of this data structure also brought new challenges. The main challenges caused by the use of triangular meshes were the correct handling of topology changes due to heavy erosion or deposition and also the need to be able to represent an object composed of multiple materials. We have proposed approaches that address these problems in the context of erosion simulation.

The erosion simulation approach used in our experiments is not physically exact, however, it gives satisfactory and visually plausible results. If more accurate solution is needed, the physics calculations can be easily replaced with more physically-based approaches.

Bibliography

[ABA02] C. Andújar, P. Brunet, and D. Ayala. "Topology-reducing Surface Simplification Using a Discrete Solid Representation". In: *ACM Trans. Graph.* 21.2 (Apr. 2002), pp. 88–105. ISSN: 0730-0301.

[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.

[Abd+14] X. Abdikerem, L. Wang, A. Jin, and M. Geni. "Numerical modeling and simulation of wind blown sand morphology under complex wind-flow field". In: *Journal of Applied Mathematics* 2014 (2014).

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

[ACK13] M. Attene, M. Campen, and L. Kobbelt. "Polygon mesh repairing: An application perspective". In: *ACM Comput. Surv.* 45.2 (Mar. 2013), 15:1–15:33. ISSN: 0360-0300.

[Ada+07] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. "Adaptively sampled particle fluids". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301.

[All+03] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. "Anisotropic polygonal remeshing". In: *ACM Transactions on Graphics (TOG).* Vol. 22. 3. ACM. 2003, pp. 485–493.

[Ama06] T. Amada. "Real-Time Particle-Based Fluid Simulation with Rigid Body Interaction". In: *Game Programming Gems 6.* Ed. by M. Dickheiser. Charles River Media, 2006, pp. 189–205.

[Att10] M. Attene. "A lightweight approach to repairing digitized polygon meshes". In: *The Visual Computer* 26.11 (2010), pp. 1393–1406. ISSN: 0178-2789.

[BA05] B. Beneš and X. Arriaga. "Table Mountains by Virtual Erosion." In: *Proceedings of the Eurographics Workshop on Natural Phenomena, NPH 2005.* Ed. by P. Poulin and E. Galin. Eurographics Association, 2005, pp. 33–39. ISBN: 3-905673-29-0.

[BB09] T. Brochu and R. Bridson. "Robust Topological Operations for Dynamic Explicit Surfaces". In: *SIAM Journal on Scientific Computing* 31.4 (June 2009), pp. 2472–2493. ISSN: 1064-8275.

[Bea+07] M. Beardall, M. Farley, D. Ouderkirk, C. Reimschussel, J. Smith, M. Jones, and P. Egbert. "Goblins by Spheroidal Weathering". In: *Proceedings of the Third Eurographics Conference on Natural Phenomena.* NPH'07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 7–14. ISBN: 978-3-905673-49-4.

[Ben+06] B. Beneš, V. Těšínský, J. Hornyš, and S. K. Bhatia. “Hydraulic erosion”. In: *Computer Animation and Virtual Worlds* 17.2 (2006), pp. 99–108. ISSN: 1546-427X.

[Ben07] B. Beneš. “Real-Time Erosion Using Shallow Water Simulation.” In: *VRIPHYS*. Eurographics Association, 2007, pp. 43–50. ISBN: 978-3-905673-65-4.

[BF01] B. Beneš and R. Forsbach. “Layered Data Representation for Visual Simulation of Terrain Erosion”. In: *Proceedings of the 17th Spring conference on Computer graphics*. SCCG ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 80–. ISBN: 0-7695-1215-1.

[BF02] B. Beneš and R. Forsbach. “Visual simulation of hydraulic erosion”. In: *Journal of WSCG* (2002), pp. 79–86.

[BK01] M. Botsch and L. Kobbelt. “A Robust Procedure to Eliminate Degenerate Faces from Triangle Meshes.” In: *VMV*. 2001, pp. 283–290.

[BK05] S. Bischoff and L. Kobbelt. “Structure Preserving CAD Model Repair”. In: *Computer Graphics Forum* 24.3 (2005), pp. 527–536. ISSN: 1467-8659.

[BLW14] J. Bronson, J. A. Levine, and R. Whitaker. “Lattice Cleaving: A Multimaterial Tetrahedral Meshing Algorithm with Guarantees”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.2 (Feb. 2014), pp. 223–237. ISSN: 1077-2626.

[BPK05] S. Bischoff, D. Pavic, and L. Kobbelt. “Automatic Restoration of Polygon Models”. In: *ACM Trans. Graph.* 24.4 (Oct. 2005), pp. 1332–1352. ISSN: 0730-0301.

[BR04] B. Beneš and T. Roa. “Simulating Desert Scenery”. In: *WSCG (Short Papers)*. 2004, pp. 17–22.

[Bri08] R. Bridson. *Fluid Simulation For Computer Graphics*. Ak Peters Series. A K Peters, 2008. ISBN: 9781568813264.

[BS09] C. Braley and A. Sandu. “Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods”. In: *Computer* (2009).

[Béz+10] R. Bézin, A. Peyrat, B. Crespin, O. Terraz, X. Skapin, and P. Meseure. “Interactive hydraulic erosion using cuda”. In: *Computer Vision and Graphics*. Springer, 2010, pp. 225–232.

[Chr+14] A. N. Christiansen, M. Nobel-Jørgensen, N. Aage, O. Sigmund, and J. A. Bærentzen. “Topology Optimization Using an Explicit Interface Representation”. In: *Struct. Multidiscip. Optim.* 49.3 (Mar. 2014), pp. 387–399. ISSN: 1615-147X.

[Cig+08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. “MeshLab: an Open-Source Mesh Processing Tool”. In: *Eurographics Italian Chapter Conference*. Ed. by V. Scarano, R. D. Chiara, and U. Erra. The Eurographics Association, 2008. ISBN: 978-3-905673-68-5.

[CK10] M. Campen and L. Kobbelt. “Exact and Robust (Self-)Intersections for Polygonal Meshes”. In: *Computer Graphics Forum* 29.2 (2010), pp. 397–406. ISSN: 1467-8659.

[Cla+13] P. Clausen, M. Wicke, J. R. Shewchuk, and J. F. O’brien. “Simulating Liquids and Solid-liquid Interactions with Lagrangian Meshes”. In: *ACM Trans. Graph.* 32.2 (Apr. 2013), 17:1–17:15. ISSN: 0730-0301.

[CMF98] N. Chiba, K. Muraoka, and K. Fujita. “An erosion model based on velocity fields for the visual simulation of mountain scenery”. In: *The Journal of Visualization and Computer Animation* 9.4 (1998), pp. 185–194. ISSN: 1099-1778.

[Cor+16] G. Cordonnier, J. Braun, M.-P. Cani, B. Benes, E. Galin, A. Peytavie, and E. Guérin. “Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion”. In: *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics*. EG ’16. Lisbon, Portugal: Eurographics Association, 2016, pp. 165–175.

[Cor+17] G. Cordonnier, E. Galin, J. Gain, B. Benes, E. Guérin, A. Peytavie, and M.-P. Cani. “Authoring Landscapes by Combining Ecosystem and Terrain Erosion Simulation”. In: *ACM Trans. Graph.* 36.4 (July 2017), 134:1–134:12. ISSN: 0730-0301.

[Cre+14] B. Crespin, R. Bézin, X. Skapin, O. Terraz, and P. Meseure. “Generalized maps for erosion and sedimentation simulation”. In: *Computers & Graphics* 45 (2014), pp. 1–16. ISSN: 0097-8493.

[Dav11] C. M. Davenport. *Incompressible Navier-Stokes equations reduce to Bernoulli’s Law*. 2011. URL: `http://home.comcast.net/~cmdaven/navier.htm` (visited on 05/06/2012).

[DBG14] F. Da, C. Batty, and E. Grinspun. “Multimaterial Mesh-based Surface Tracking”. In: *ACM Trans. Graph.* 33.4 (July 2014), 112:1–112:11. ISSN: 0730-0301.

[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*. Springer-Verlag, 1996, pp. 61–76.

[Dor+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. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 225–234. ISBN: 0-201-48560-5.

[Dur64] R. Durstenfeld. “Algorithm 235: random permutation”. In: *Communications of the ACM* 7.7 (1964), p. 420.

[EM90] H. Edelsbrunner and E. P. Mücke. “Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms”. In: *ACM Trans. Graph.* 9.1 (Jan. 1990), pp. 66–104. ISSN: 0730-0301.

[Eng00] G. A. Engines. *The Volume Library - The Tooth Dataset*. Accessed: 2016-01-13. 2000.

[Enr+02] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. “A hybrid particle level set method for improved interface capturing”. In: *Journal of Computational Physics* 183.1 (2002), pp. 83–116.

[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. New York, NY, USA: ACM, 2001, pp. 23–30. ISBN: 1-58113-374-X.

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. “On Visible Surface Generation by a Priori Tree Structures”. In: *SIGGRAPH Comput. Graph.* 14.3 (July 1980), pp. 124–133. ISSN: 0097-8930.

[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. New York, NY, USA: ACM, 2001, pp. 15–22. ISBN: 1-58113-374-X.

[FTB16] N. Faraj, J.-M. Thiery, and T. Boubekeur. “Multi-material adaptive volume remesher”. In: *Computers & Graphics* 58 (2016). Shape Modeling International 2016, pp. 150 –160. ISSN: 0097-8493.

[FY56] R. A. Fisher and F. Yates. “Statistical Tables for Biological, Agricultural and Medical Research”. In: *Vet. Rec* 68.1015 (1956).

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

[HH01] Y. Hatano and N. Hatano. “Dune Morphology and Sand Transport”. In: *Forma* 16.1 (2001), pp. 65–75. ISSN: 0911-6036.

[Hib10] A. Hibbs. *Navier-Stokes Equation*. 2010. URL: `http://www2.warwick.ac.uk/fac/sci/physics/pendulum/navierstokes/` (visited on 05/06/2012).

[Hoe09] R. Hoetzlein. *FLUIDS v.2 - A Fast, Open Source, Fluid Simulator*. 2009. URL: `http://www.rchoetzlein.com/eng/graphics/fluids.htm` (visited on 04/30/2012).

[Jon+10] M. D. Jones, M. Farley, J. Butler, and M. Beardall. “Directable Weathering of Concave Rock Using Curvature Estimation”. In: *IEEE Transactions on Visualization and Computer Graphics* 16 (2010), pp. 81–94. ISSN: 1077-2626.

[JSZ18] M. Jiang, R. Southern, and J. J. Zhang. “Energy-based dissolution simulation using SPH sampling”. In: *Computer Animation and Virtual Worlds* 29.2 (2018). e1798 cav.1798, e1798. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.1798`.

[Ju04] T. Ju. “Robust repair of polygonal models”. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 888–895. ISSN: 0730-0301.

[Kim+15] H.-J. Kim, A. C. Öztireli, I.-K. Shin, M. Gross, and S.-M. Choi. “Interactive Generation of Realistic Facial Wrinkles from Sketchy Drawings”. In: *Computer Graphics Forum*. Vol. 34. 2. Wiley Online Library. 2015, pp. 179–191.

[KLN91] M. Karasick, D. Lieber, and L. R. Nackman. “Efficient Delaunay Triangulation Using Rational Arithmetic”. In: *ACM Trans. Graph.* 10.1 (Jan. 1991), pp. 71–91. ISSN: 0730-0301.

[KM90] M. Kass and G. Miller. “Rapid, Stable Fluid Dynamics for Computer Graphics”. In: *SIGGRAPH Comput. Graph.* 24.4 (Sept. 1990), pp. 49–57. ISSN: 0097-8930.

[Kri+09] P. Krištof, B. Beneš, J. Křivánek, and O. Šťava. “Hydraulic erosion using smoothed particle hydrodynamics”. In: *Computer Graphics Forum*. Vol. 28. 2. Wiley Online Library. 2009, pp. 219–228.

[KW06] P. Kipfer and R. Westermann. “Realistic and interactive simulation of rivers”. In: *Proceedings of Graphics Interface 2006*. GI ’06. Quebec, Canada: Canadian Information Processing Society, 2006, pp. 41–48. ISBN: 1-56881-308-2.

[LC87] W. E. Lorensen and H. E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930.

[Lev+05] M. Levoy, J Gerth, B Curless, and K Pull. *The Stanford 3D Scanning Repository*. 2005. URL: `http://graphics.stanford.edu/data/3Dscanrep/` (visited on 07/20/2015).

[LHN05] S. Lefebvre, S. Hornus, and F. Neyret. *GPU Gems 2. Chapter 37: Octree Textures on the GPU*. 2005.

[Lie94] P. Lienhardt. “N-dimensional generalized combinatorial maps and cellular quasi-manifolds”. In: *International Journal of Computational Geometry & Applications* 4.03 (1994), pp. 275–324.

[LMS11] F. Löffler, A. Müller, and H. Schumann. “Real-time Rendering of Stack-based Terrains.” In: *VMV*. 2011, pp. 161–168.

[Luc77] L. B. Lucy. “A numerical approach to the testing of the fission hypothesis”. In: *Astronomical Journal* 82 (Dec. 1977), pp. 1013–1024.

[LW04] S. Lo and W. Wang. “A fast robust algorithm for the intersection of triangulated surfaces”. English. In: *Engineering with Computers* 20.1 (2004), pp. 11–21. ISSN: 0177-0667.

[Man82] B. B. Mandelbrot. *The Fractal Geometry of Nature*. San Francisco: W.H. Freeman, 1982. ISBN: 0-7167-1186-9.

[MB12] M. K. Misztal and J. A. Bærentzen. “Topology-adaptive Interface Tracking Using the Deformable Simplicial Complex”. In: *ACM Trans. Graph.* 31.3 (June 2012), 24:1–24:12. ISSN: 0730-0301.

[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. San Diego, California: Eurographics Association, 2003, pp. 154–159. ISBN: 1-58113-659-5.

[McL+13] D. McLaurin, D. Marcum, M. Remotigue, and E. Blades. “Repairing unstructured triangular mesh intersections”. In: *International Journal for Numerical Methods in Engineering* 93.3 (2013), pp. 266–275. ISSN: 1097-0207.

[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. Washington, DC, USA: IEEE Computer Society, 2007, pp. 47–56. ISBN: 0-7695-3009-5.

[Mei75] G. H. Meisters. “Polygons Have Ears”. In: *The American Mathematical Monthly* 82.6 (1975), pp. 648–651. ISSN: 00029890.

[MF97] T. M. Murali and T. A. Funkhouser. “Consistent Solid and Boundary Representations from Arbitrary Polygonal Data”. In: *Proceedings of the 1997 Symposium on Interactive 3D Graphics*. I3D ’97. Providence, Rhode Island, USA: ACM, 1997, 155–ff. ISBN: 0-89791-884-3.

[MK14] P. Martínek and I. Kolingerová. “Deformation method for 3d identikit creation”. In: *Computer Graphics Theory and Applications (GRAPP), 2014 International Conference on*. IEEE. 2014, pp. 1–8.

[MKM89] F. K. Musgrave, C. E. Kolb, and R. S. Mace. “The Synthesis and Rendering of Eroded Fractal Terrains”. In: *SIGGRAPH Comput. Graph.* 23.3 (July 1989), pp. 41–50. ISSN: 0097-8930.

[MMW01] T. Miao, Q. Mu, and S. Wu. “Computer simulation of aeolian sand ripples and dunes”. In: *Physics Letters A* 288.1 (2001), pp. 16 –22. ISSN: 0375-9601.

[Mon92] J. J. Monaghan. “Smoothed particle hydrodynamics”. In: *Annual review of astronomy and astrophysics* 30 (1992), pp. 543–574.

[NT03] F. S. Nooruddin and G. Turk. “Simplification and Repair of Polygonal Models Using Volumetric Techniques”. In: *IEEE Transactions on Visualization and Computer Graphics* 9 (2003), pp. 191–205.

[NVI] NVIDIA. *Nvidia FleX*. `http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/flex/index.html`. Online; Accessed: 01/04/2018.

[OF02] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces (Applied Mathematical Sciences)*. 2003rd ed. Springer, Nov. 2002. ISBN: 0387954821.

[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. Washington, DC, USA: IEEE Computer Society, 2000, pp. 427–. ISBN: 0-7695-0868-5.

[Par65] E. Partheniades. “Erosion and deposition of cohesive soils”. In: *Journal of Hydraulics Division of the American Society of Agricultural Engineers 91* (1965), pp. 105–139.

[Pey+09] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou. “Arches: a framework for modeling complex terrains”. In: *Computer Graphics Forum*. Vol. 28. 2. Wiley Online Library. 2009, pp. 457–467.

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

[Pra+17] M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. “Curvature-Based Feature Detection for Head Modeling”. In: *Procedia Computer Science* 108 (2017). International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland, pp. 2323 –2327. ISSN: 1877-0509.

[Pur09] V. Purchart. “Modelování písčitého terénu pro virtuální realitu”. MA thesis. University of West Bohemia, Pilsen, Czech Republic, 2009.

[Ren+18] B. Ren, T. Yuan, C. Li, K. Xu, and S. Hu. “Real-Time High-Fidelity Surface Flow Simulation”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.8 (Aug. 2018), pp. 2411–2423. ISSN: 1077-2626.

[Rus04] S. Rusinkiewicz. “Estimating Curvatures and Their Derivatives on Triangle Meshes”. In: *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2Nd International Symposium*. 3DPVT ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 486–493. ISBN: 0-7695-2223-8.

[Si14] H. Si. *TetGen example: Dragon*. 2014. URL: `http://wias-berlin.de/software/tetgen/examples.dragon.html` (visited on 07/20/2015).

[SK15] V. Skorkovská and I. Kolingerová. "Multiple Material Meshes for Erosion Simulation". In: *Proceedings of SIGRAD 2015, June 1st and 2nd, Stockholm, Sweden*. Selected as one of the three best papers of the conference. Linköping University Electronic Press, Linköpings universitet, 2015, pp. 5–8. ISBN: 978-91-7685-855-4.

[SK16] V. Skorkovská and I. Kolingerová. "Complex multi-material approach for dynamic simulations". In: *Computers & Graphics* 56 (2016), pp. 11 –19. ISSN: 0097-8493.

[SKB15] V. Skorkovská, I. Kolingerová, and B. Benes. "Hydraulic Erosion Modeling on a Triangular Mesh". In: *Surface Models for Geosciences*. Ed. by K. Růžičková and T. Inspektor. Lecture Notes in Geoinformation and Cartography. Springer International Publishing, 2015, pp. 237–247. ISBN: 978-3-319-18406-7.

[SKB18] V. Skorkovská, I. Kolingerová, and B. Benes. "A Simple and Robust Approach to Computation of Meshes Intersection". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP*. INSTICC. SciTePress, 2018, pp. 175–182. ISBN: 978-989-758-287-5.

[Sko+17] V. Skorkovská, M. Prantl, P. Martínek, and I. Kolingerová. "Erosion-Inspired Simulation of Aging for Deformation-Based Head Modeling". In: *Procedia Computer Science* 108 (2017). International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland, pp. 425 –434. ISSN: 1877-0509.

[Sko12a] V. Skorkovská. "Modeling of Erosion Impacts on the Terrain". Master thesis. Pilsen, Czech Republic: University of West Bohemia, 2012.

[Sko12b] V. Skorkovská. "Modeling of Erosion Impacts on the Terrain". In: *Studenská vědecká konference* (2012).

[SKV19] V. Skorkovská, I. Kolingerová, and P. Vaněček. "A Unified Curvature-Driven Approach for Weathering and Hydraulic Erosion Simulation on Triangular Meshes". In: *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP*. Selected as the best student paper of the conference. INSTICC. SciTePress, 2019, pp. 122–133. ISBN: 978-989-758-354-4.

[SSP07] B. Solenthaler, J. Schläfli, and R. Pajarola. "A unified particle model for fluid-solid interactions: Research Articles". In: *Comput. Animat. Virtual Worlds* 18.1 (Feb. 2007), pp. 69–82. ISSN: 1546-4261.

[Sta+08] O. Stava, B. Beneš, M. Brisbin, and J. Křivánek. "Interactive Terrain Modeling Using Hydraulic Erosion". In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 201–210. ISBN: 978-3-905674-10-1.

[TJ10] L. A. Tychonievich and M. D. Jones. "Delaunay deformable mesh for the weathering and erosion of 3D terrain". In: *Vis. Comput.* 26.12 (Dec. 2010), pp. 1485–1495. ISSN: 0178-2789.

[Váš+16] L. Váša, P. Vaněček, M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. "Mesh Statistics for Robust Curvature Estimation". In: *Computer Graphics Forum* 35.5 (2016), pp. 271–280. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12982`.

[Van+11] J. Vanek, B. Benes, A. Herout, and O. Stava. "Large-Scale Physics-Based Terrain Editing Using Adaptive Tiles on the GPU". In: *IEEE Computer Graphics and Applications* 31.6 (Nov. 2011), pp. 35–44. ISSN: 0272-1716.

[Wan11] C. C. Wang. "Computing on rays: A parallel approach for surface mesh modeling from multi-material volumetric data". In: *Computers in Industry* 62.7 (2011), pp. 660–671.

[Wei+16] P. Wei, M. Zhang, W. Jiang, and D. Nie. "A New Model for Sand-Ripple Scattering Based on SSA Method and Practical Ripple Profiles". In: *IEEE Transactions on Geoscience and Remote Sensing* 54.4 (Apr. 2016), pp. 2450–2459. ISSN: 0196-2892.

[WH12] N. Wang and B.-G. Hu. "Real-Time Simulation of Aeolian Sand Movement and Sand Ripple Evolution: A Method Based on the Physics of Blown Sand". English. In: *Journal of Computer Science and Technology* 27.1 (2012), pp. 135–146. ISSN: 1000-9000.

[WN14] K. K. Warszawski and S. S. Nikiel. "A proposition of erosion algorithm for terrain models with hardness layer". In: *Journal of Theoretical and Applied Computer Science* 8 (1 2014), pp. 76–84. ISSN: 2299-2634.

[Woj+07] 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*. Eurographics Association, 2007, pp. 15–22.

[Woj+09] C. Wojtan, N. Thürey, M. Gross, and G. Turk. "Deforming meshes that split and merge". In: *ACM Trans. Graph.* 28.3 (July 2009), 76:1–76:10. ISSN: 0730-0301.

[WS03] Z. Wu and J. M. Sullivan. "Multiple material marching cubes algorithm". In: *International Journal for Numerical Methods in Engineering* 58.2 (2003), pp. 189–207. ISSN: 1097-0207.

[WWY06] Y. Wang, C. C. Wang, and M. M. Yuen. "Fast energy-based surface wrinkle modeling". In: *Computers & Graphics* 30.1 (2006), pp. 111–125.

[ZBH11] A. Zaharescu, E. Boyer, and R. Horaud. "Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multiview Reconstruction". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.4 (2011), pp. 823–837. ISSN: 0162-8828.

[ZHB10] Y. Zhang, T. J. Hughes, and C. L. Bajaj. "An automatic 3D mesh generation method for domains with multiple materials". In: *Computer methods in applied mechanics and engineering* 199.5 (2010), pp. 405–415.

Appendix A

Professional Activities

Publications

Impacted Journals

[SK16] V. Skorkovská and I. Kolingerová. "Complex multi-material approach for dynamic simulations". In: *Computers & Graphics* 56 (2016), pp. 11 –19. ISSN: 0097-8493.

[Váš+16] L. Váša, P. Vaněček, M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. "Mesh Statistics for Robust Curvature Estimation". In: *Computer Graphics Forum* 35.5 (2016), pp. 271–280. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12982`.

International Conferences (WoS and Scopus)

[SKB15] V. Skorkovská, I. Kolingerová, and B. Benes. "Hydraulic Erosion Modeling on a Triangular Mesh". In: *Surface Models for Geosciences*. Ed. by K. Růžičková and T. Inspektor. Lecture Notes in Geoinformation and Cartography. Springer International Publishing, 2015, pp. 237–247. ISBN: 978-3-319-18406-7.

[SKB18] V. Skorkovská, I. Kolingerová, and B. Benes. "A Simple and Robust Approach to Computation of Meshes Intersection". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP*. INSTICC. SciTePress, 2018, pp. 175–182. ISBN: 978-989-758-287-5.

[Sko+17] V. Skorkovská, M. Prantl, P. Martínek, and I. Kolingerová. "Erosion-Inspired Simulation of Aging for Deformation-Based Head Modeling". In: *Procedia Computer Science* 108 (2017). International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland, pp. 425 –434. ISSN: 1877-0509.

[SKV19] V. Skorkovská, I. Kolingerová, and P. Vaněček. "A Unified Curvature-Driven Approach for Weathering and Hydraulic Erosion Simulation on Triangular Meshes". In: *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications* -

Volume 1: GRAPP. Selected as the best student paper of the conference. INSTICC. SciTePress, 2019, pp. 122–133. ISBN: 978-989-758-354-4.

[Pra+17] M. Prantl, V. Skorkovská, P. Martínek, and I. Kolingerová. "Curvature-Based Feature Detection for Head Modeling". In: *Procedia Computer Science* 108 (2017). International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland, pp. 2323 –2327. ISSN: 1877-0509.

International Conferences (other)

[SK15] V. Skorkovská and I. Kolingerová. "Multiple Material Meshes for Erosion Simulation". In: *Proceedings of SIGRAD 2015, June 1st and 2nd, Stockholm, Sweden*. Selected as one of the three best papers of the conference. Linköping University Electronic Press, Linköpings universitet, 2015, pp. 5–8. ISBN: 978-91-7685-855-4.

Student Publications

[Sko12a] V. Skorkovská. "Modeling of Erosion Impacts on the Terrain". Master thesis. Pilsen, Czech Republic: University of West Bohemia, 2012.

[Sko12b] V. Skorkovská. "Modeling of Erosion Impacts on the Terrain". In: *Studenská vědecká konference* (2012).

Stays Abroad

- Universidad de Las Palmas de Gran Canaria, Spain, February-June 2011
- Purdue University, Indiana, USA, October 2011 (1 week), October 2012 (1 week), October 2013 (2 weeks)

Participation in Scientific Projects

- Interactive Geometrical Models for Simulation of Natural Phenomena and Crowds. Project leader Ivana Kolingerová. Funded by The Ministry of Education, Youth and Sports, project code LH11006. (2011 - 2013)
- Project NTIS (New Technologies for Information Society), European Center of Excellence. Funded by The European Regional Development Fund (ERDF), project code CZ.1.05/1.1.00/0.2.0090. (2013 - 2014)
- Advanced Computing and Information Systems. Project code SGS-2013-029. (2013 - 2015)
- Advanced Graphical and Computing Systems. Project code SGS-2016-013. (2016 - 2018)