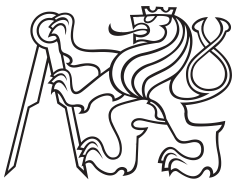
Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Weathering of rocks

Weathering simulation of sandstone formations

Vojtěch Cimbura

Supervisor: Ing. Jaroslav Sloup
Field of study: Open informatics
Subfield: Computer games and graphics
May 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Cimbura** Jméno: **Vojtěch** Osobní číslo: **474637**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Simulace zvětrávání hornin

Název bakalářské práce anglicky:

Weathering of rocks

Pokyny pro vypracování:

Seznamte se s fyzikálními a chemickými procesy zvětrávání hornin a metodami jejich simulace používanými v počítačové grafice [1-5]. Proveďte rešerši existujících metod a porovnejte je z hlediska simulovaných procesů zvětrávání. Vytvořte interaktivní aplikaci, která bude simulovat zvětráváni hornin a sedimentaci erodovaného materiálu.
Funkčnost a možnosti metody demonstrujte alespoň na třech různých scénách a výsledky porovnejte se skutečnými fotografiemi. Zhodnoťte pamětovou a operační složitost implementace.
Implementaci proveďte v C/C++ s využitím OpenGL.

Seznam doporučené literatury:

[1] McKay T. Farley: Fast Spheroidal Weathering with Colluvium Deposition. Master thesis, Brigham Young University, 2011.
[2] Matthew Beardall, Mckay Farley, Darius Ouderkirk, Jeremy Smith, Michael Jones,Parris K. Egbert: Goblins by Spheroidal Weathering. Eurographics Workshop on Natural Phenomena, pp. 7-14, 2007.
[3] Michael D. Jones, McKay Farley, Joseph Butler, Matthew Beardall: Directable weathering of concave rock using curvature estimation. IEEE Transactions on Visualization and Computer Graphics, vol.16, num.1, p.81-94, 2009.
[4] A. Peytavie, E. Galin, J. Grosjean, S. Mérillou: Arches: a framework for modeling complex terrains. Computer Graphics Forum, Vol. 28, No. 2, pp. 457-467, 2009.
[5] E. Galin, E. Guérin, A. Peytavie, G. Cordonnier, M.P. Cani, B. Benes, J. Gain: A review of digital terrain modeling. Computer Graphics Forum, Vol.38, No.2, pp. 553-577, 2019.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jaroslav Sloup, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2020** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to extend my sincere thanks to my supervisor, Ing. Jaroslav Sloup, for his guidenance and valuable advises and suggestions. I am also grateful to my family for their kind support and understanding. To my friends, thank you for listening, offering me advice, and supporting me through this entire process.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, 22. May 2020

.....

Abstract

This bachelor thesis discusses rock weathering simulation techniques in the field of computer graphics and suggests an implementation of an application dealing with this subject. An application producing realistic rock formations might find use in computer-generated animations and games, because modelling of specific rock formations and at the same time keeping the realism of the modelled object can be demanding.

Keywords: Computer graphics, Simulation, Weathering, Spheriodal weathering, Erosion

Supervisor: Ing. Jaroslav Sloup
Karlovo náměstí 13
121 35 Prague 2
Czech Republic

Abstrakt

Tato bakalářská práce pojednává o simulacích zvětrávání v rámci počítačové grafiky a současně navrhuje implementaci aplikace, která umožňuje simulaci zvětrávání skal. Aplikace umožňující automatickou generaci věrohodných a reálných skalních útvarů může najít uplatnění ve videoherním i filmovém průmyslu, protože ruční modelování skalních útvarů a zároveň zachování co největší realističnosti modelu je časově náročná a složitá činnost.

Klíčová slova: Počítačová grafika, Simulace, Zvětrávání, Sférické zvětrávání, Eroze

Překlad názvu: Simulace zvětrávání hornin

Contents

Figures

Tables

Chapter 1

Introduction

Landscape is unique on every place on Earth and changes with latitude, longitude and climate. People studied the impact of natural forces on the landscape throughout the years in order to predict the landscape development in the future. With the development of computer graphics, attempts to simulate the landscape in the virtual world has become a popular topic. Nowadays, it is possible to recreate many kinds of real landscape, which is particularly useful for movie industry or computer games. Several approaches to the landscape simulation were invented, however, there is still a small amount of landforms not easily reproducible, especially when automatic generation is required. A goblin, or hoodoo, is considered one of those. Goblins are usually formed by a combination of spheroidal weathering and erosion. Thanks to the goblin's unusual shape, the geological processes behind the forming of goblins were precisely investigated and algorithms capable of goblin creation were created. A comparison of a real goblin and goblins created thanks to an algorithm implementing spheroidal weathering and erosion can be seen in Figure 1.1.

Figure 1.1: Left: A goblin in Goblin Valley State Park, Utah, USA (from [1]), Right: Goblins created by spheroidal weathering (from [2]).

In Figure 1.2, there is a spectacular view on a rock formation called rock city, which was also created thanks to weathering and erosion. The photography is accompanied by a simulation attempt resembling a rock city. The goblin-like shape of individual rocks suggests that the forces applied are similar even though both rock formations are located on different continents.

Figure 1.2: Left: The Bohemian Switzerland National Park (from [3]), Right: A rock city created by spheroidal and cavernous weathering (from [2]).

It is advisable to get acquainted with the basic geological principles first. Therefore, Chapter 2 will discuss the geological processes behind the creation of spectacular rock formations. Majority of presented processes are associated with sandstone rocks, since those tend to weather the most. Chapter 3 introduces various existing methods and algorithms implementing the spheroidal weathering, as well as other simulation algorithms. Following Chapter 4 describes the problem analysis, suggests solution and introduces the application structure. Finally, the simulation results will be evaluated in Chapter 5 and the work concluded in Chapter 6.

Chapter 2

Geological principles of sandstone landforms

In this chapter we will focus on the geological processes behind forming sandstone landforms. Sandstone is generally known for its ability to create large diversity of shapes. The shape of the created landform depends on many factors, such as sandstone type, climate conditions, weathering and so on. The most essential factors will be further reviewed in this chapter. The weathering types are introduced in Section 2.1. More complex weathering types, such as Spheroidal weathering and Cavernous weathering, will be discussed in Sections 2.2 and 2.3. An easily noticeable geological activity, fracturing, will be discussed in Section 2.4. Finally, the gravitation force impact on the rock will be examined in Section 2.5.

Let's consider the difference between weathering and erosion. Generally, weathering is a process of break down of rocks, soil, and minerals. On the other hand, erosion involves the movement of rocks by water, wind, waves and gravity. The eroded material is then being transported and relocated, causing the exposure of rock. Usually, weathering and erosion go hand in hand - weathering disintegrates the rock, erosion transports the loose material. However, one should distinguish between those phenomenons.

2.1 Basic weathering types

The weathering causes can be divided into three main categories as described in a short article by Pamela J. W. Gore [4]. The categories are Physical and mechanical weathering, Chemical weathering and Biological weathering. Physical weathering disintegrates rock through mechanical processes, which are further described in Subsection 2.1.1. Chemical weathering decomposes rocks using chemical reactions. Biological weathering is considered as a weathering, where a living organism is present on the rock, affecting or damaging the rock structure. All three types are further discussed in this section.

2.1.1 Physical and mechanical weathering

Frost weathering, or frost wedging, is a type of physical weathering. Water freezes in tiny cracks or holes, creating stress within a rock. This phenomena occurs repeatedly and thus makes the holes larger. Frost weathering can occur anywhere where water is present and the temperatures are below -3 Celsius [17]. **Thermal expansion** is caused by repeatedly and regularly changing temperature. Heat causes the rock to expand, whereas cooling results into contraction. Furthermore, different minerals expand and contract at different rates resulting into stresses along mineral boundaries and eventually cause the crack to create along different material layers. In **Pressure release**, also known as unloading, rock breaks off along joints. Underlying rocks release pressure by expanding, which causes the uplifting mass to fracture parallel the surface (Figure 2.1).

Figure 2.1: Pressure release causes the rock to break off into leaves or sheets along joints (from [4]).

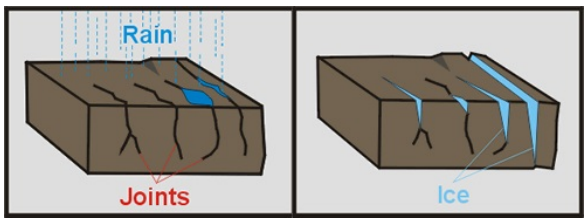


Figure 2.2: Frost wedging illustration (from [5]).

2.1.2 Chemical weathering

Hydrolysis is a chemical reaction, where water reacts with several minerals. Silicate minerals weather to form clay. Variety of metals undergo **Oxidation** - oxygen combined with iron-bearing minerals results into "rusting". This gives the affected rocks a reddish-brown coloration on the surface which crumbles easily and weakens the rock. **Dissolution** is a material dissolving process, usually happening in water. The most common material afflicted by dissolution is calcite, which is present in limestone and marble. The weathering rate of calcite is even higher when the water is acidic, around 4 pH. The most typical dissolution outcome are stalactites and stalagmites in caves (Figure 2.3). **Spheroidal weathering** and **Cavernous weathering** will be discussed in depth in Sections 2.2 and 2.3.

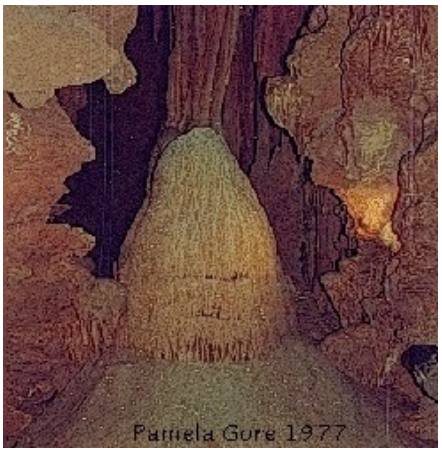


Figure 2.3: A stalagmite was formed due to dissolution (from [4]).

2.1.3 Biological weathering

Organisms can assist in breaking down rock into sediment or soil. Tree roots and other plants growing on rocks exert physical pressure or creates pathway for water. Also, lichens, mosses, micro-organisms or animals may disrupt the rock surface.

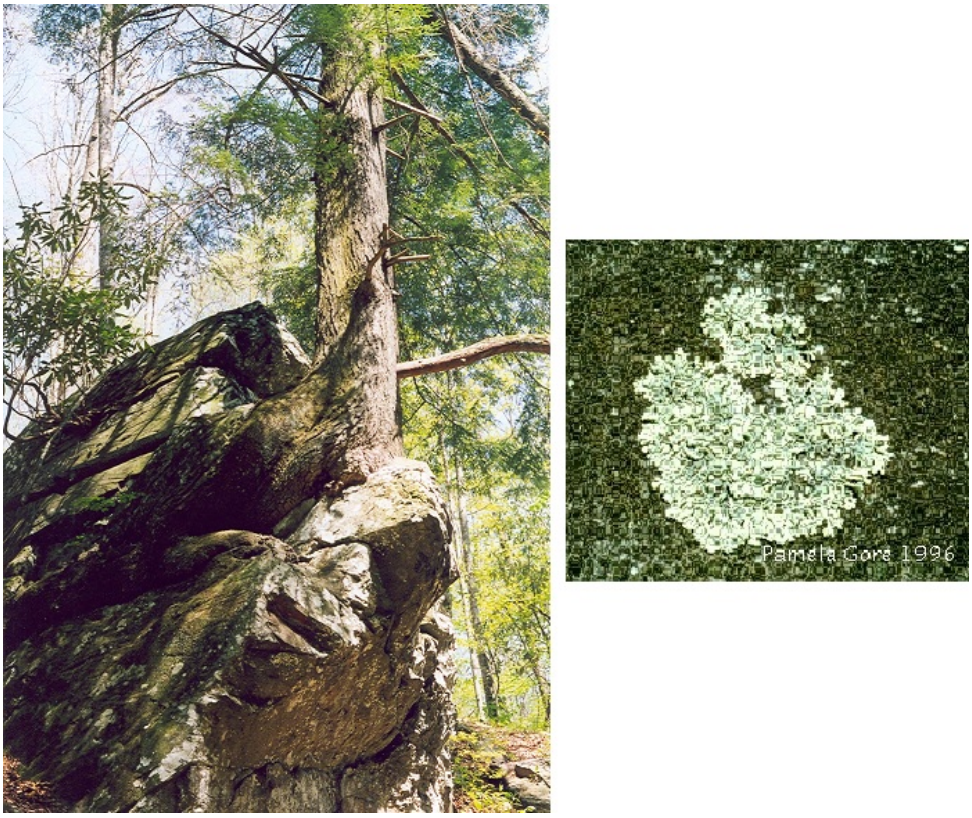


Figure 2.4: Left: Tree roots in rock, Anna Ruby Falls, GA, USA; Right: Closeup of lichen, Stone Mountain, GA, USA (both from [4]).

2.2 Spheroidal weathering

Spheroidal weathering is a chemical weathering process that results in the formation of concentric or spherical layers of highly decayed rock within weathered bedrock. Such formation can result into pillar-shaped rocks, called goblins. In order to describe the process closely, it is appropriate to be focused on more specific rock formation. In this case, the hoodoos, or goblins located in the Goblin Valley State Park in Utah, USA, will be considered. The process of forming goblins can be divided, according to a publication by M. Milligan [18], into four following phases.

The first phase consists of depositing various alternating materials, such as shale, siltstone or sandstone. Those materials are later transformed into rock. Material properties vary, however, the crucial property is the weathering rate, which describes material's resistance to weathering. Apparently, the weathering rate differs with layer. All the layers together form the geologic formation called Entrada sandstone.

The second phase is called fracturing. There are multiple fractures or joints, that can be found within the sandstone. Fractures weaken the rock and as a result, the layers of the rock are susceptible to various weathering types.

Figure 2.5: A goblin in the Goblins National Park, Utah, USA (from [6]).

Next phase consists of spheroidal weathering. Spheroidal weathering decomposes rock through chemical reactions. Significantly faster weathering can be observed on non flat surfaces, for instance sharp edges and corners, which results into smooth and rounded rock surface. Mathematically, this can be modeled using the ratio of surface area to volume for a given portion of rock [10]. After the rock has been weakened by the weathering, portions of layers incline to erode.

The last phase, called transportation, is closely associated with the previous phase. As the sandstone erodes, water and wind remove the eroded material, exposing new parts of the rock to the spheroidal weathering. Eroded material, called colluvium, that has eroded and fallen down, can be seen in Figure 2.5 above.

2.3 Cavernous weathering

Cavernous weathering falls into the chemical weathering category. Cavernous weathering, also called Honeycomb weathering, results into creation of caverns or pits within a rock [19], as shown in Figure 2.6. The weathering rate is almost the same as spheroidal weathering, the difference is in the curvature representation. The cavernous weathering rate is proportional to negative curvature rather than positive curvature and it increases with negative curvature, because surface points tend to be protected from exposure by adjacent rock. The longer this region is protected from the sun, the longer

it holds moisture and leads to increased deposition of salt. Increased salt concentrations accelerate the breakdown of chemical cementation between rock particles and cause the protected rock to disintegrate more quickly.

Figure 2.6: Rock erosion caused by cavernous weathering, Altdahn Castle in the Palatinate Forest, Germany (from [7]).

One can conclude that caverns are likely to begin on shaded areas. These shaded areas might retain the water longer than not shaded areas, thus adding to the amount of salt deposited. Generally, the more salt is deposited, the more the rock weakened is, eventually enlarged into caverns.

2.4 Fractures within a rock

Fracturing is the underlying cause of earthquakes, and also contributes to the evolution of regional tectonic features. According to Ben A. Van der Pluijm and S. Marshak [20], a fracture is any surface of discontinuity within a rock, i.e. a surface across which the material is no longer bonded. Those fractures are usually formed by brittle deformation. Brittle deformation is the permanent change that occurs in a solid material due to the growth of fractures and/or due to sliding on fractures once they have formed. There are many types of fractures: Shear fractures (or faults) are fractures across which there has been displacement. Joints (or cracks), in contrast, are caused by tensile load. We will provide more information about cracks origin and development.

Tham et al [21] and Akesson et al [22] proposed that microcracks mainly occur along the mineral grains. This observation also implies that microcracks might be present anywhere within the rock. According to Y. Liu et al [8], during early loading, tension cracks are the dominant mode of cracking. When the cumulative damage increases, shear cracks become the main mode of cracking. A shear crack might join several tension cracks and lead to final rupture of the rock. The result of observations made by Liu et al on the crack propagation principle is shown in Figure 2.7.

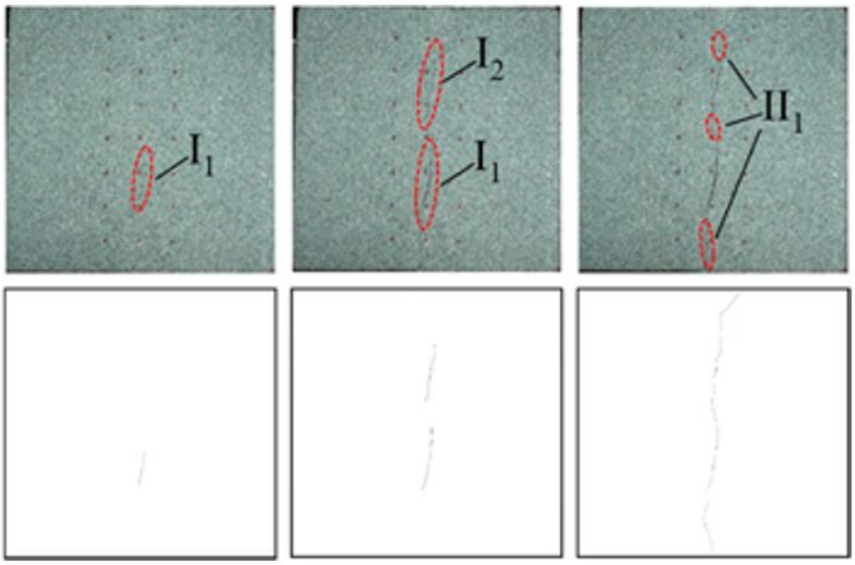


Figure 2.7: Macrocrack propagation under normal stress at 3.0 MPa, from left to right: a) first tension macrocrack I1; b) second tension macrocrack I2; c), tension macrocracks joined by shear cracks II, leading to final rupture (from [8]).

2.5 Gravitation impact on a rock

The process of forming landforms, such as arches, pedestral rocks or pillars has not been clearly explained for a long time. Geologists simply accepted that the formation is performed by weathering alone. Somehow, the influence of gravity has been overlooked and not been explored further. A few years back, in 2015, the team of geologists led by J. Bruthans presented a study about shaping the sandstone landforms [9].

The sandstone consists of several layers, where each layer can be formed by different materials. One can find sandstone layer, where the sand grains are connected by cements, which are keeping the grains together. On the other hand, there are rock layers without the cementation, sometimes called

'Locked sands'. The name originates in the structure. The pressure applied on those grains forced them to change shape and fit together well. According to J. Bruthans and co., as we decrease the pressure applied to this layer of the rock, the inner material tension decreases and grains are less resistant to weathering. As a result, the layer starts to erode quickly. Bruthans and co. also suggests that the effect of gravity loading stress plays an important role. In his study, several experiments have been performed and the results indicated, that the gravity effect on the sandstone rock shape was wrongly assumed to increase the landform's weathering rate. On the contrary, the higher pressure is applied on the rock, the more resistant the material is.

It is advisable to mention the relationship between fractures and gravitation. Various cracks within the sandstone cause the stress to transfer not evenly, making the force 'go around' the cracks. Then, the erosion will carry away all the erodible material. When the stress reaches a critical value (critical stress) it causes the remaining rock to stabilize. At this point, the rock is resistant to further erosion. This phenomena can form spectacular rock formations, such as the Delicate Arch in Utah, USA. The results of Bruthans study are presented in Figure 2.8.

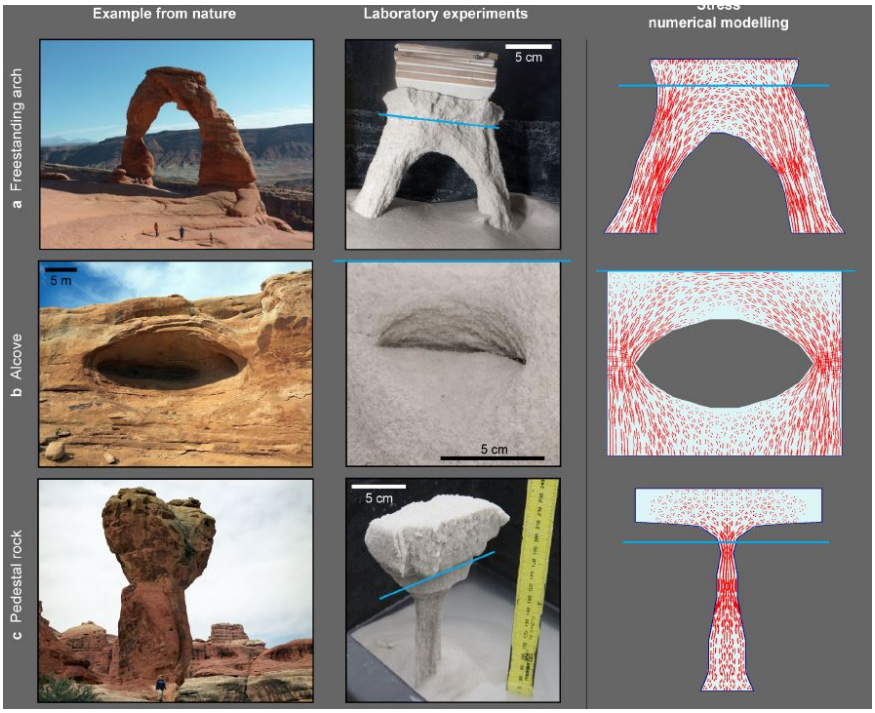


Figure 2.8: Result of experiments with sandstone rock formations. Top left picture shows the Delicate arch in Utah, USA (from [9]).

Chapter 3

Related work

Various approaches achieving plausible weathering simulation will be presented in this chapter. There are algorithms for generating terrain based on height maps, although those algorithms are not suitable for concave surfaces, because concave surfaces require multiple elevation values for a single grid point. Algorithms using procedural terrain generation based on voxel grid are used to implement physically based concave terrain, yet this approach is based on analytically differentiable equations, which are not suitable in many terrain features. Various algorithms and data structures for concave terrain generation were created instead, some of them are further presented in this chapter.

3.1 Data representation possibilities

With the knowledge of how weathering processes and erosion affects the rock, one can conclude that it is practically impossible to simulate the physics of all weathering processes. Such a computation is simply not possible to perform. Instead, we have basically two options:

Firstly, a real physical process approximation can be used. Simulation based on approximation will be presented in sections 3.2, 3.3 and 3.4. All of them use uniform voxel grid (Figure 3.1). The biggest advantage of using voxel grid is the ability to perform fairy simple computations above the grid, however, it is memory extensive to store all the voxels, especially when creating large

landscape. Another possibility is to use a material-stack data-structure, as proposed by A. Peytavie et al. [11]. Here the terrain is represented as a two dimensional grid of material stacks (see Figure 3.2), where overhangs, arches and caves can be created by adding air sections between material layers. This data representation is both computationally and memory efficient, but the weathering algorithm or fractures propagation cannot be easily implemented.

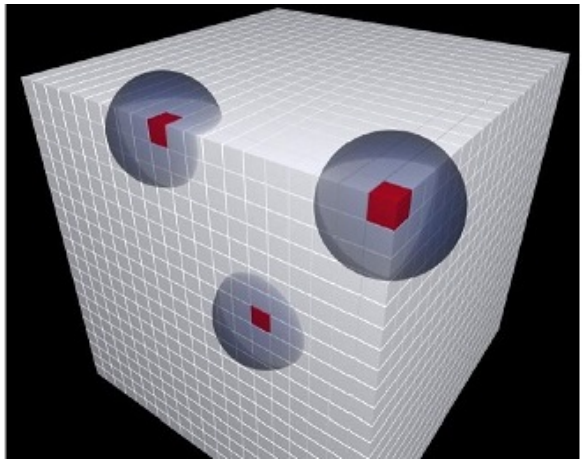


Figure 3.1: Voxel data-structure (from [10]).

Secondly, as Paris et al. suggests [23], the construction trees and implicit terrain models can be used, so the impact of physical forces can be omitted in order to obtain plausible results. Big advantage is the possibility to create plausible landscape up to 5x5km in reasonable computation time. However, this kind of implementation is not suitable for us, because we would need a comprehensive framework.

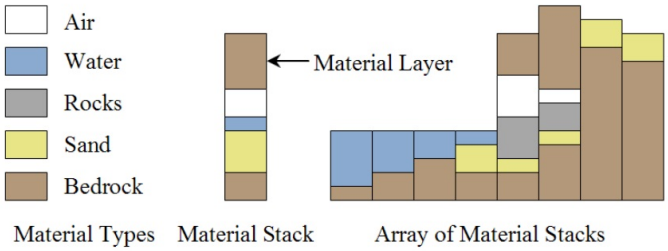


Figure 3.2: Material stack based data-structure (from [11]).

3.2 Spheroidal weathering simulation

This approach was presented in the article published by M. Beardall et al. [10]. The model is based on voxel grid, where spheroidal weathering algorithm is applied. At the beginning of the simulation, uniform size voxels form a uniform voxel grid. Each column in the grid is divided into regions specified by bedding plane layers. Furthermore, each voxel belongs to a layer which defines the voxel's resistance to weathering. The voxel's resistance to weathering is based on the input parameters, such as a bubble shape and size. A bubble of different size and shape with origin in the voxel's center is used to compute voxel's durability. The durability (a decimation rate) is set to the product of the voxel's resistance to weathering and the air to rock ratio for each bubble. For instance, voxels in corners have higher air to rock ratio than voxels exposed by one face only, which results into faster weathering of more exposed voxels. This refers to the spheroidal weathering, as described in Section 2.2.

As soon as the voxel's decimation reaches zero, the voxel is permanently removed from the grid. After a voxel erodes, all voxels that contain the destroyed voxel in their bubble are notified and their decimation rate is recomputed. Such operation is expensive, thus the amount of voxels for each bubble size and shape is stored in a lookup table to avoid recomputing.

The application architecture is shown in Figure 3.3. Firstly, the user specifies properties of each layer, such as resistance, height and the bubble parameters. A vertical joint map is specified too. Then the simulation starts, going through several cycles. After the simulation is finished, the result can be exported into a rendering program.

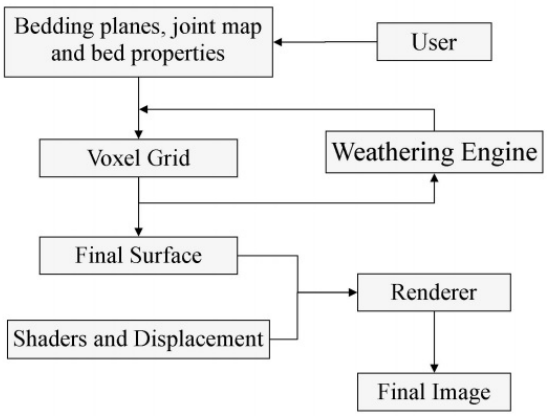


Figure 3.3: The architecture of goblin simulation program presented in 3.2 (from [10]).

3.3 Spheroidal and Cavernous weathering simulation

Another approach to spheroidal weathering was presented by Jones et al. [2]. Their approach is based on similar principle as described in Section 3.2, however, efficiency gains were achieved by reducing the amount of voxels, which are being updated during the weathering simulation. Instead of performing computation above all voxels in the grid, the computation includes only 'active' voxels. As an active voxel can be considered a voxel that is exposed by at least one face to air. See Figure 3.4.

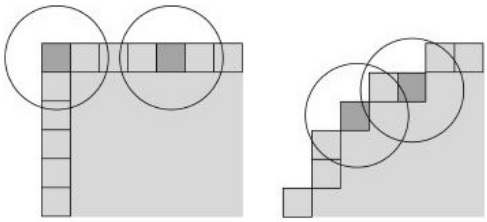


Figure 3.4: The spheroidal weathering process; Left: Before; Right: After the simulation (from [2]).

The biggest extension compared to the first method is the cavernous weathering implementation. Considering spheroidal weathering, voxels on edges containing in its bubble more air undergo faster weathering, whereas in cavernous weathering, voxels containing bigger rock portion tend to weather faster. See Figure 3.5 for 2D voxel grid representation with negative curvature.

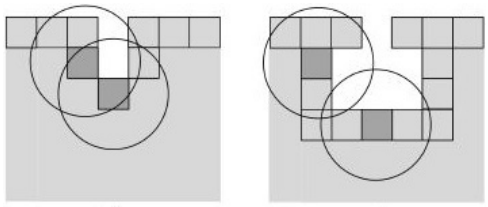


Figure 3.5: Cavernous weathering principle; Left: Before; Right: After the simulation (from [2]).

Furthermore, colluvium deposition algorithm has been implemented. Colluvium distribution process contains 2 phases, which are repeated until the colluvium unit reaches a stable position. The first phase simulates sliding sleep slopes until the unit reaches the angle of repose. Second phase is activated, when the unit stops sliding and detaches from the surface. There, the iteration in the $-y$ direction starts until the colluvium reaches a non-air voxel. At this point, the process either ends - the unit reached a stable spot - or the unit starts to slide again and enters the first phase. Another similar approach to material deposition will be discussed in Section 3.5.

3.4 Spheroidal weathering on GPU

Spheroidal weathering simulation accelerated by GPU was presented as a master thesis by McKay T. Farley [12].

Major improvement in efficiency has been achieved by caching mechanism that stores the number of air voxels in the bubble around each voxel, so instead of calculating the decimation at each simulation step, the value is available in a lookup table. Therefore, each voxel has a index to the lookup table, which corresponds to the number of air voxels. For example, a rock voxel containing 5 air voxels in it's bubble has stored an index of 5, so there is no need to visit all neighbour voxels per each simulation step. This caching technique is similar to the caching mechanism presented in Section 3.2. Dealing with fully decimated voxels now requires updating the cached air voxel count for neighboring voxels within the bubble of decimated voxel. The update is simply incrementing the air voxel count by one for each neighbouring voxel.

Another efficiency gain can be achieved by using the GPU to compute the simulation steps. In this case, the OpenCL architecture has been used. OpenCL is single-instruction multiple-data architecture (SIMD). Generally, OpenCL is a framework allowing parallel computing on CPU and GPU and so allows to utilize the tremendous parallel processing cores on today's graphics cards. Parallel computing on the GPU can achieve highest efficiency if all the cores are performing the same task, but with different values. For instance, when the instruction to be executed is the same on each core for each step, the ideal situation for parallel computing occurred. On the other hand, if there is any branch in the code causing two cores to process differently, the operation will be executed on the first core and then on the second core instead of in parallel. See figure 3.6, where Core 1 has different instruction set causing a slowdown and cores 2,3,4 work in parallel. Simply, more branching leads to longer computational time.

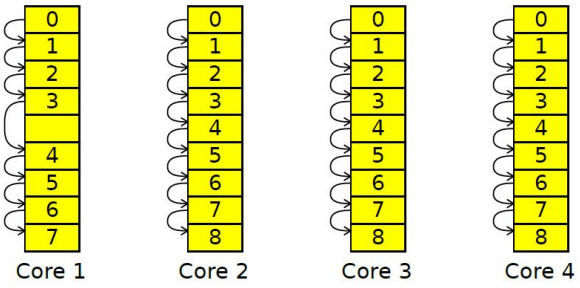


Figure 3.6: Code path with Core 1 executing different instructions set (from [12]).

3.5 Material stack based simulation with rockfall

A. Peytavie et al. [11] proposed a material stabilization simulation. This simulation uses material stack data-structure, which is suitable to simulate the rockfall. Erosion simulation was implemented as well, however, this section will focus on the rockfall simulation only.

In order to be able to simulate rockfall, we need to define the angle of repose between two neighbouring material stacks as the angle between the horizontal plane and the contact plane. The angle of repose characterization has been simplified by processing each material type separately, i.e. first it is necessary to sort the layers by merging materials of the same kind together. Then the materials are sorted in the order: sand, rock, air. It is possible to use a unique angle of repose for each material type. The stabilization process is triggered when the angle of repose is higher than a predefined threshold. Then the material is moved from the too high material stack to the neighbouring stacks. The moved material height, which has been moved to a neighbouring stack, is a weighted average proportional to the height difference between the two stacks. It is essential to mention that only a small amount of material is moved to the neighbouring stacks to avoid oscillation in the algorithm. The stabilization process is repeated until all the material layers are stabilized.

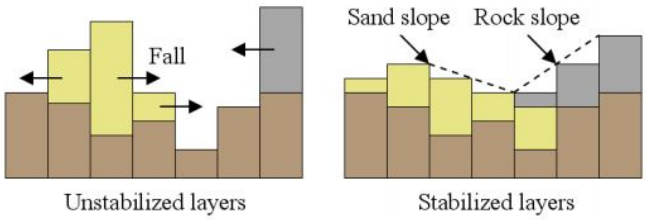


Figure 3.7: Material layers stabilization according to the angle of repose for different materials (from [11]).

Chapter 4

Analysis and Solution

The aim of this bachelor thesis is to simulate weathering and erosion impact on a rock. The applied simulation algorithms should be physically based as much as possible. The user should be able to choose between different rock types, choose the size of simulated rock or the amount of factors affecting the simulation. There has to be the possibility to pause the simulation and continue in the simulation if required, as well as the possibility to export the simulation results into a third party programs, where the model can be modified and textures applied. Furthermore, the user should be able to run the simulation above the same grid with different parameters. For this purpose, configuration files with the generated rock parameters should be available. This way the user has the opportunity to use the application as a tool that helps to create plausible and physically based rocks without the need of tedious modelling process.

This chapter discusses the solution proposal and the chosen simulation approach, as well as implementation details. Some application features will be described in Sections from 4.1 through 4.3, such as details of voxel grid generations in 4.1, configuration files in 4.2 and finally possibility to export currently rendered data in 4.3. Simulation algorithms are introduced in further sections. Spheroidal weathering algorithm will be described in Section 4.4, algorithm used for eroded material distribution in Section 4.5, gravitation force impact on the rock in 4.6 and finally, fracture creation and propagation is presented in Section 4.7.

4.1 Application structure

The application is written in C++ and OpenGL 3.1. Simple shaders, camera movements and rendering were implemented. In our application we will be rendering many voxels (cubes) with identical mesh but with different transformations. Therefore, the most efficient way to render many similar objects is to render them using a single call. This technique is called instanced rendering [24]. So instead of iterating over all the voxels and drawing them individually, we will prepare all the model matrices for all the voxels and send them to the GPU in one call. This way the performance bottleneck caused by sending data to the GPU for each voxel will be avoided.

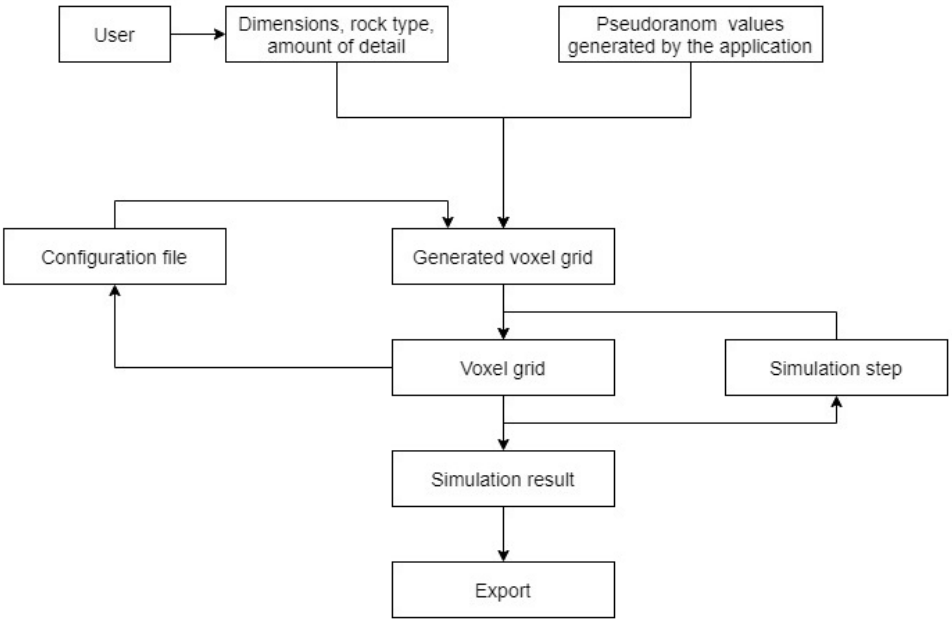


Figure 4.1: The application flow visualized.

Application loop and event flow is shown in Figure 4.1. Here, each simulation step will consist of application of our simulation algorithms. Those algorithms will be applied in this order: Fractures propagation algorithm, spheroidal weathering algorithm, algorithm implementing the gravitation force impact and lastly the eroded material deposition algorithm. The simulation step can be executed manually or automatically using the automatic simulation option. While running the simulation manually, the user runs simulation one simulation step per click. On the other hand, the automatic simulation can be run and stopped by the user at any time. There is a possibility to customize the time passed between simulation steps. This value can be changed while the automatic simulation is running and is particularly useful when it comes to smaller voxel grids, where one simulation step can take a few milliseconds.

4.1.1 Voxel grid

Voxel data structure has been used despite the memory requirements, because the spheroidal weathering algorithm can be implemented above the voxel grid easily compared to other data structures. The voxel grid is created by rendering a simple uniform cube on uniform intervals, eventually forming a grid of dimension $width \times height \times depth$. As an area designed for the eroded material to fall and spread, a 5 voxel wide area is added on each side of the rock. Each voxel has assigned multiple values, where the most essential values will be further introduced.

Voxel type: There are four voxel types when creating the voxel grid. The Rock voxels are voxels which form the rock shape. Air voxels are not rendered by the application, because they represent an empty cell in the voxel grid. Ground voxels are used only for the interpretation of a firm surface beneath the rock formation and are not changed throughout the simulation whatsoever. The last type is an Eroded voxel, which replaces the Rock voxel as soon as the Rock voxel erodes. Moreover, the Eroded voxel works as a stack of eroded material (colluvium) that changes it's height according to the material amount stored in this voxel. The material amount stored depends on rules introduced in Section 4.5.

Weathering rate: A rock is formed by layers of voxels, where each layer's weathering rate is assigned a float value based on normal (Gaussian) distribution with mean of $\mu = 0.7$ and standard deviation $\sigma = 0.1$. A voxel within the layer takes the weathering rate value from the parent layer and then this value is changed according to a normal distribution with parameters $\mu = 0.0$ and $\sigma = 0.03$ in order to achieve a non uniform weathering rate within the layer. Visualizing the voxel's weathering rate is achieved by sending the weathering rate value to the vertex shader while rendering each voxel. This value is then recomputed to a RGB color in the vertex shader, later set as a fragment color in the fragment shader, making each layer and voxel visually distinctive. Red color represents the most resistant material - 1.0, whereas blue color represents the least resistant material. Colors in between are interpolated.

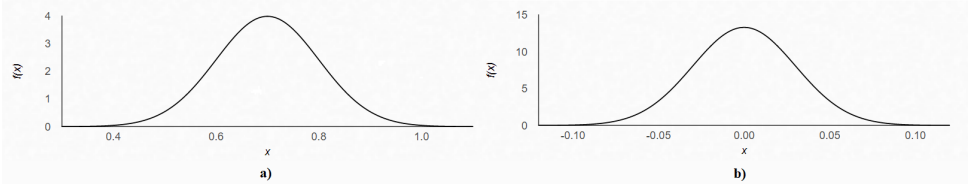


Figure 4.2: a) Gaussian distribution for layer weathering rates, b) Gaussian distribution for voxels within the layer.

Durability: Voxel's durability is a float value in interval $[0.0, 1.0]$ and is considered for Rock voxels only. Durability represents 'health points' of each voxel, so if the durability reaches zero, the voxel is changed into Eroded voxel. The durability is lowered in each simulation step based on rules presented in Section 4.4. Additionally, the user has the option to visualize the durability of voxels. While having this option enabled, the durability is being sent to the vertex shader. Here the outgoing color is multiplied by the durability, which results into darkening of the voxel's color.

4.1.2 Voxel grid scale

Let's suppose we have a uniform voxel grid with uniform voxel size. We would like to add more detail to this uniform grid by creating a grid of the same size, but with more voxels. One can accomplish that by decreasing the voxel's size and modifying the voxel positions by the same factor, making the voxel grid more dense. However, if the spheroidal weathering is applied on both uniform and denser grid, the resulting shape will be significantly different after the same simulation step count. The same stands for other algorithms, such as deposition algorithm and gravitation force impact.

There are four grid detail levels that can be chosen before the simulation starts in our application. The detail level defines the number of voxels generated at one length unit. To counter problems listed in the previous paragraph, it is necessary to let the denser voxel grid weather faster by the same factor which has been used for decreasing the voxel's size. The same factor is used for simulation algorithms as well. Considering fractures propagation, a denser grid will result into more detailed fractures. The grid detail level is visualized in Figure 4.3, where the dimensions of both voxel grids are equal. Both grids were captured after 35 simulation steps with gravitation force impact applied during the simulation.

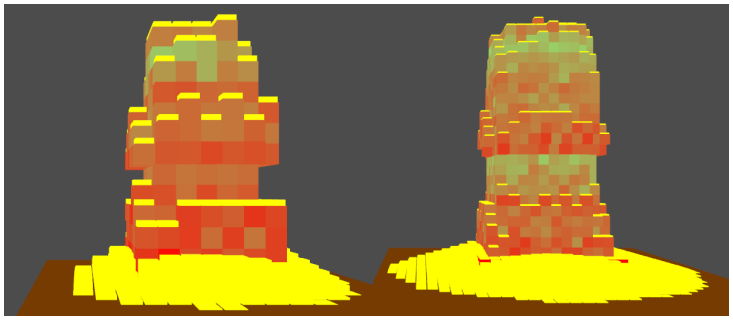


Figure 4.3: Left: A voxel grid with default density, Right: A 2x denser voxel grid.

4.2 Configuration files

The ability to recreate certain shape with different parameters is an indispensable functionality for all simulations. Therefore, an configuration file should contain all simulation parameters needed for an simulation reproduction. Such a file is usually created in a human readable form with a purpose of further modifications by the user. The user then needs to consider that any modification within the configuration file might cause an unexpected behaviour. To counter that, the application producer should provide a configuration file specification, containing listed simulation variables and their ranges.

Our application provides the user with configuration files in JSON format. A configuration file can be created arbitrarily during the simulation. When creating, the user specifies the desired target location and name of the file in a popup dialog. A notification is displayed in the Application Log as soon as the configuration file is saved. Configuration file specification is displayed in Figure 4.4. Any modification made to the file needs to be within specified limits. The file can be modified using any software that supports JSON file format. The configuration file can be loaded from the application menu.

```
"dimensions": limited to create max 10 M voxels,
"layer scale": float value from {0.2, 1/3, 0.5, 1.0},
"layers":[float values in range <0.0, 1.0>],
"rock city":{
    "heights": [unsigned integers],
    "x coords":[unsigned integers],
    "x sizes": [unsigned integers],
    "z coords":[unsigned integers],
    "z sizes": [unsigned integers]
},
"seed":unsigned integer,
"type":unsigned integer from {1,2,3}
```

Figure 4.4: Limits of variables located in configuration file.

4.3 Data export

A comprehensive framework supporting textures application and models creation would be necessary when it comes to applying textures and direct model export. One can use third party software instead. Firstly, exported data type needs to be determined and a third party software supporting the data type needs to be chosen. Simulations using a voxel grid might usually produce volumetric data, so a third party software supporting volumetric data import should be used. A suitable software meeting our requirements is called ParaView, since it offers a VTK file format that can be used for volumetric data import. Another requirement is the possibility to apply textures and modify the rock shape. Unfortunately this process cannot be performed in ParaView, however, exporting data from ParaView is possible. A modeling program, such as Blender or Maya, can be used afterwards for applying textures and modifying the rock shape.

Our application supports exporting the simulation result into third party modelling programs. Currently rendered rock can be saved into a file generated by the application and the user will specify the target location of this file. The VTK file format is used to export the rendered data, because it is supported by ParaView. One can easily import the generated file into ParaView and visualize the simulation output with lightning, since our simulation program does not support any lightning models. Examples of visualized data in ParaView are shown in Figures 4.5 and 4.6.

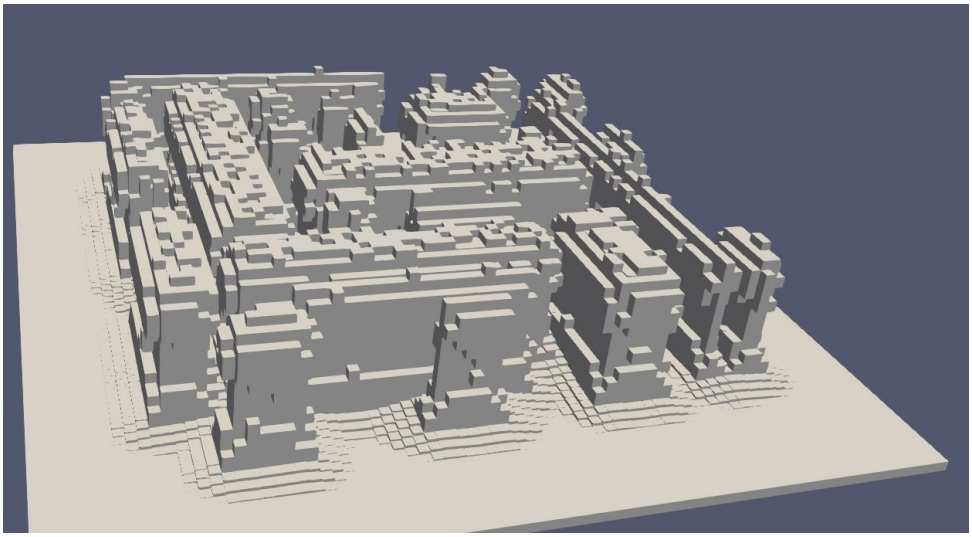


Figure 4.5: A rock city exported to ParaView

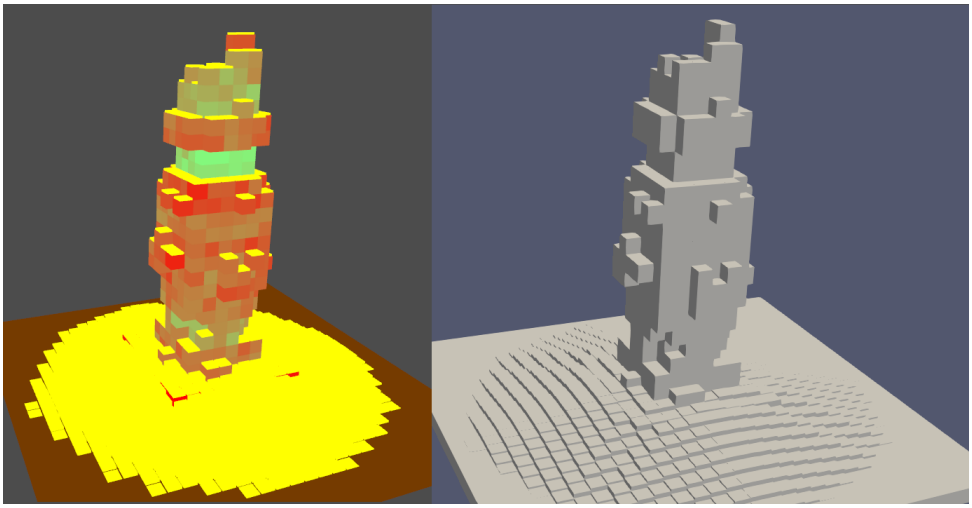


Figure 4.6: Left: A goblin after the simulation is finished, Right: The same goblin exported to ParaView.

4.4 Spheroidal weathering implementation

The spheroidal weathering algorithm can be implemented intuitively and the most easiest on the voxel grid, compared to other possible data representations. Previously introduced methods in Chapter 3 proposed several possibilities to make the algorithm more efficient, such as a caching mechanism storing the number of air voxels in the surrounding of the currently processed voxel (Section 3.4) or the reduction of voxels that will not be influenced by the spheroidal weathering in the simulation step whatsoever (Section 3.3). GPU acceleration of the algorithms would be a valuable improvement, but unfortunately it will be not further considered, because the implementation would be demanding and it would take considerable amount of time to implement the algorithm correctly.

A spheroidal weathering algorithm based on 3.2 and 3.3 was implemented in our application. Spheroidal weathering impact on the voxel grid is computed every simulation step. In order to speed up the algorithm, only voxels exposed to the surface are considered, as suggested in [12] and explained in Section 3.3. While processing a voxel, it's neighbours are visited in two phases. The first phase I consists of visiting voxels neighbouring by a face (6 voxels). The second phase II consist of visiting voxels neighbouring by an edge (12 voxels). Each phase have different impact on the currently processed voxel - phase I voxels have greater impact than phase II voxels, because an imaginary circumscribed sphere of the currently processed voxel occupies a greater part of the volume of the phase I voxels. If there is an Air voxel, the currently processed voxel is exposed to the air and thus it should lose some durability,

depending on the phase. As soon as the durability reaches zero, the voxel erodes.

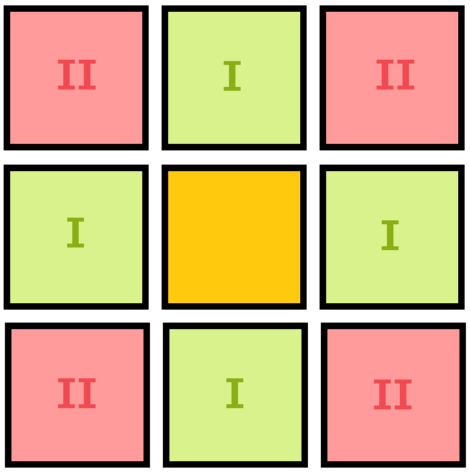


Figure 4.7: A 2D view in a $Width \times Depth$ plane on Phase I and II voxels, yellow color represents the currently processed voxel.

The durability is lowered by the following formulae 4.1, where New stands for the new voxel's durability, Old for old voxel's durability, $Rate$ for voxel's weathering rate and $AirRatio/RockRatio$ is the percentage of air in the surrounding of the currently processed voxel. Adding the weathering rate into the durability computation ensures that voxels with higher durability tend to weather slower than voxels with lower durability.

$$New = Old - (1 - Rate) * (AirRatio/RockRatio) \tag{4.1}$$

4.5 Material deposition implementation

Material deposition algorithm described in Section 3.5 is implemented above the material-stack based data-structure. A crucial property is the angle of repose, which defines the maximum slope used for the material movement and sedimentation. Since our implementation will be based on a voxel grid, the deposition algorithm needs to be adjusted accordingly. We will deal with one eroded material type only, which means that only one angle of repose needs to be defined. Furthermore, the material stabilization process seems be the most computational time consuming process of the simulation, because stabilizing one layer can trigger the stabilization process on already processed material stacks, causing the algorithm to recompute the unstable stacks until

the whole grid is stabilized. In short, several algorithm passes will be required to stabilize the eroded material.

The eroded material distribution principle was taken from Section 3.5 and modified afterwards. The value of the angle was set to 30 degrees. Before the deposition algorithm starts, it is crucial to merge all possible 'floating' Eroded voxels. If a floating Eroded voxel is detected, the amount of material it holds will be iteratively lowered to the first non-floating Eroded voxel. The type of a floating Eroded voxel is changed to Air voxel afterwards. After the material merging is completed, the deposition algorithm can begin.

Let $h1$ and $h2$ denote the height of two neighbouring Eroded voxels, d denote the distance between centers of those voxels and α the angle of repose. Let $\triangle h = h1 - h2$ denote the difference between heights of those voxels. Then we define $M = 0$ if $\triangle h < d.tan(\alpha)$ and $M = \triangle h - d.tan(\alpha)$ otherwise, where M is the height of material to be moved in order to maintain the angle of repose. Material moves to a neighboring voxel if the angle between the tops of those two voxels is greater than the angle of repose. The height of material moving from the central voxel to a neighboring voxel is defined as a weighted average proportional to the height difference between the voxels. To avoid osculations in the algorithm, the amount of material to be moved is set as a small constant amount a. This process is repeated iteratively until all the material layers are stabilized.

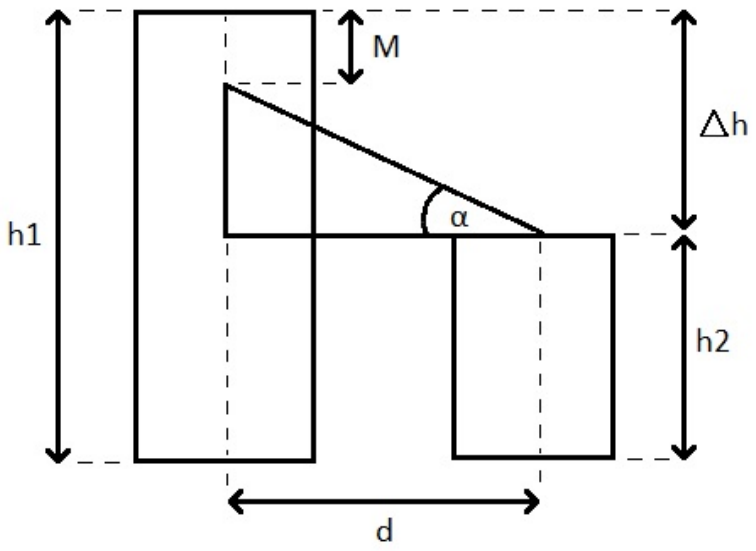


Figure 4.8: Deposition algorithm principle visualized.

4.6 Gravitation force impact implementation

Unfortunately, the research yielded challenging simulation techniques only, considering the stresses within the rock. Such simulation would require to study the stresses propagation within the rock closely, which is beyond the scope of this thesis. The results of studies by J. Bruthans et al. [9], introduced in Chapter 2, will be examined instead. The most crucial observation from their study is that with increased pressure applied, the rock will become more resistant. This observation is crucial for us since we would like to eliminate the floating parts of the rock as much as possible.

Before the gravitation force algorithm starts, all floating Rock voxels need to be detected. A simple Breath First Search was implemented to achieve this goal. The algorithm goes through all voxels in the lowest layer and at each voxel it tries to expand all neighbouring Rock voxels. After the search ends, any leftover Rock voxels which were not visited during the search will have their durability decreased to zero.

The algorithm starts at the top of the rock formation and gradually advances to the bottom of the rock, layer by layer. While processing a layer, all voxels within the layer are counted and the load applied on them is summed up. Then the algorithm proceeds to the next layer and counts the voxels within the layer. The load from the higher layer is then evenly distributed between voxels within the currently processed layer. As the load applied on a voxel increases, the voxel becomes more resistant to weathering. The weathering rate of each voxel is increased by the product of voxel's initial weathering rate and the load applied on the voxel. Furthermore, the value needs to be adjusted according to the rock height. The adjustment consists of dividing the value by the height of the rock formation. Such adjustment ensures that the weathering rate of voxels at the bottom will not be enormously high. The whole formulae is shown below in 4.2, where $Rate$ denotes the voxel's weathering rate, $Load$ denotes the load applied on the voxel by the gravitation force and $Height$ denotes the height of the rock.

$$Rate = Rate + Load * Height \tag{4.2}$$

The user has can turn this functionality on or off in the simulation menu, although it is recommended keeping this functionality enabled to counter the floating voxels problem. The following Figures 4.9 and 4.10 show the differences between keeping the gravitation force impact turned on and off.

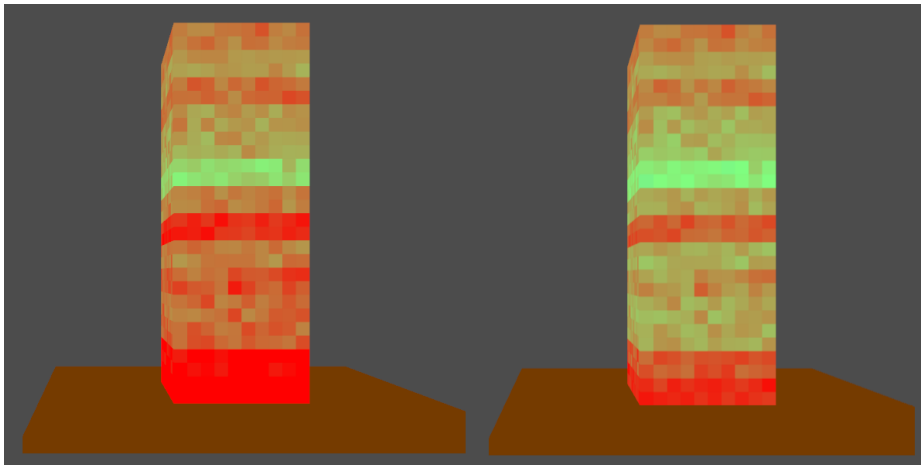


Figure 4.9: Generated voxel grid in initial state, Left: gravitation force applied, Right: without gravitation force.

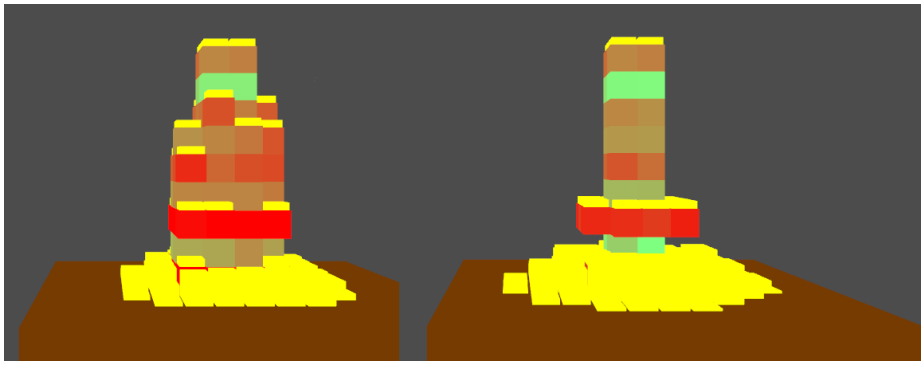


Figure 4.10: A goblin after 25 simulation steps, Left: simulation with gravitation force applied, Right: simulation without gravitation force.

4.7 Fractures propagation implementation

Fractures propagation causing displacement of the rock is related to the stresses within the rock. There will not be any advanced stress simulation implemented in our application, so different approach needs to be selected. Based on study by Liu et.al.[8], introduced in Chapter 2, shear fractures are created in two steps. Firstly, a tension macrocracks are created and with the cumulative damage increasing, shear cracks may occur. Our approach will be rather based on this study. Also, our fractures implementation will be limited by the voxel grid - one cannot simply visualize fractures precisely using an uniform voxel grid. Therefore, fractures needs to be visualized by deforming the voxel according to the six possible fracture positions on the voxel's faces.

The fractures propagation algorithm is divided into two parts as well. The first algorithm phase consists of defining whether a tension macrocrack will be generated based on the fracture creation probability, defined by the user. If a voxel with a tension macrocrack should be generated, a voxel exposed to the surface by at least one face is found and the macrocrack is generated on one of it's faces. The second algorithm phase immediately follows the first phase. All voxels with a tension macrocrack are searched and their distance is computed. If their distance is lower than given threshold, a shear macrocrack can appear, connecting both tension macrocracks. In order to find the path of the shear macrocrack, A* path finding algorithm was used to find the shortest path between those voxels. Voxels euclidean distance was used as the heuristic function for the A* algorithm instead of Manhattan distance, because we would like to achieve more natural fracture development. After A* finishes, the shortest path returned by the algorithm is then backtracked and a continuous fracture along the path is formed. The user can switch fractures propagation functionality on or off.

After a shear crack is created, the size of the fracture increases with each simulation step. As soon as the size of the crack is bigger than the voxel on which this crack is generated, the voxel will erode. Moreover, voxels with cracks incline to erode faster than voxels without cracks. In the next Chapter 5, our application was able to re-create an arch, or a small tunnel within the rock, which has been created thanks to a fracture leading through the rock. This fracture later ensured that the spheroidal weathering enlarged the path along the fracture.

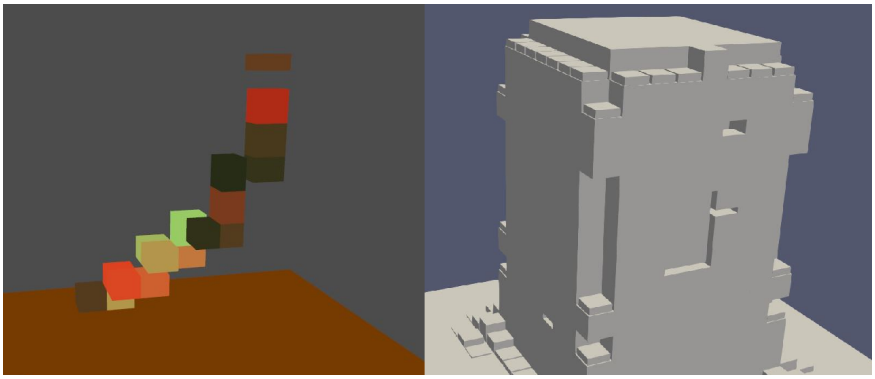


Figure 4.11: Left: Generated fracture in a fracture-only rendering mode, Right: the whole rock exported to ParaView with visible fracture.

Chapter 5

Results

This chapter introduces the simulation results accompanied with real world rocks. Our simulation modes, or scenes, will be discussed and compared to the real world rocks on photos.

Firstly, the goblin simulation will be presented in Section 5.1, followed by simulation of an overhang in Section 5.2. The most unexpected result will be introduced in Section 5.3 - an arch simulation. Our last simulation mode will try to generate a rock city, in Section 5.4.

A laptop with the following parameters listed in Table 5.1 was used for our tests.

Model	DELL G5 5587
OS	MS Windows 10 Home 64-bit
Processor	Intel Core i7-8750H
GPU	NVIDIA GeForce GTX 1060 6GB
RAM	16GB

Table 5.1: Computer specifications.

5.1 Goblin

A voxel grid of dimensions $9 \times 19 \times 9$ and a scale of 0.5 (i.e. 2x denser grid) was used to simulate one of the most iconic goblins in Goblin Valley State park, Utah. The gravitation force impact was enabled throughout the whole simulation and fractures creation and propagation was disabled. Following four Figures show our results.

An inaccuracy can be observed at the top of the goblin, because our application uses the same weathering rate for the whole layer and the user cannot change manually the weathering rate for specific voxels within the layer. The voxel grid generation was introduced in Chapter 4, Section 4.1.

The application used 43.2 MB RAM and approximately 20% of GPU usage while rendering the scene with the simulation running.

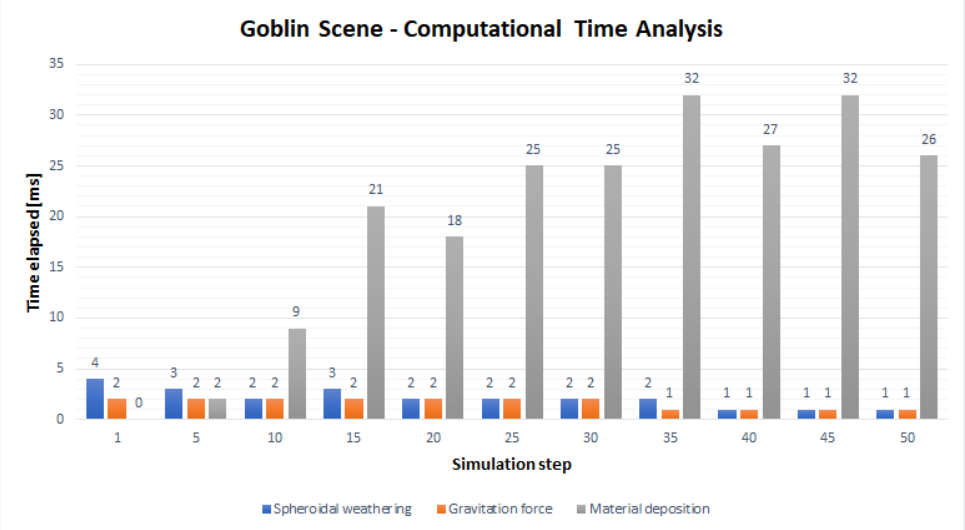


Figure 5.1: Elapsed time measured at each 5^{th} step.

Spheroidal weathering	106 ms
Gravitation force	80 ms
Material deposition	1058 ms
Whole simulation	1499 ms

Table 5.2: Goblin - Elapsed time of each algorithm during the simulation.

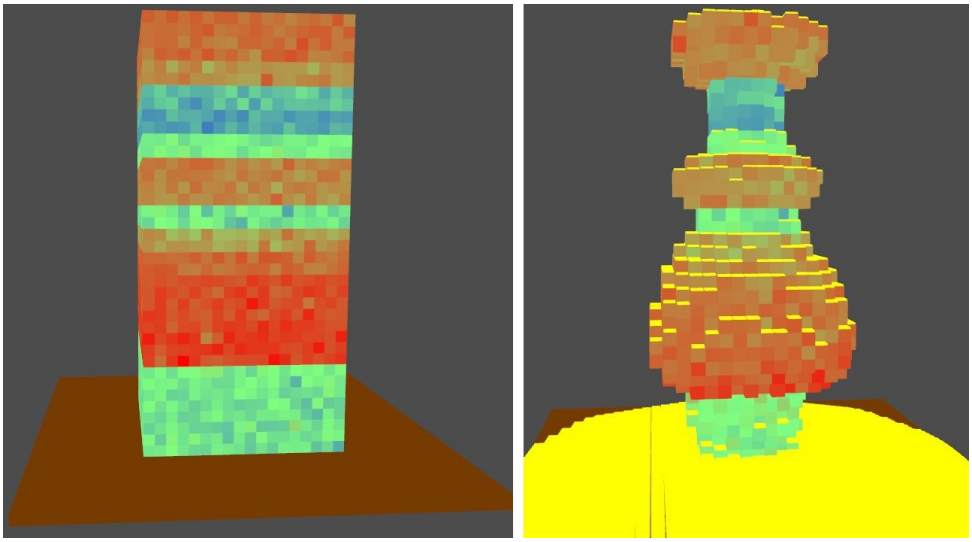


Figure 5.2: Left: The generated voxel grid, Right: The rock after 50 simulation steps.

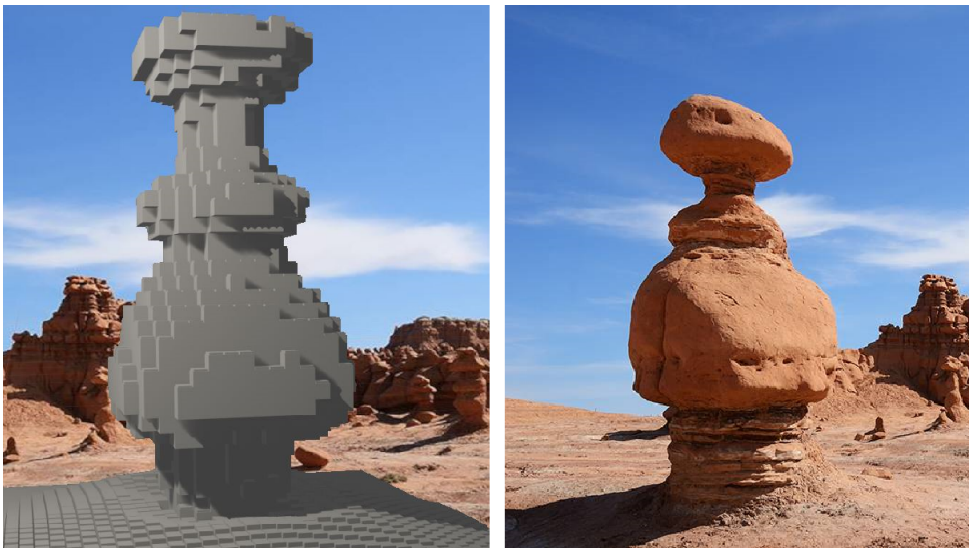


Figure 5.3: Left: The rock visualised in Blender, Right: The reference picture (from [1], edited).

One can notice that our simulation did not re-create the rounded shape of the goblin precisely. This imprecision is generally caused by two factors: The first one refers to the square voxel's shape, the ability to render a smooth shape according to voxel's decimation, specifically. The second reason is that the more detailed grid causes to generate voxels with unique weathering rate within the layer, which might eventually result into different shape. However, a bigger scale also results into longer computational time - for comparison, simulation of the goblin with 5x bigger scale took 47 935ms, which is almost 32x slower than simulation with 2x bigger scale. Also, achieving the same shape took 3 extra simulation steps while using the 5x bigger scale. The difference between two different scales applied is shown in Figure 5.4.

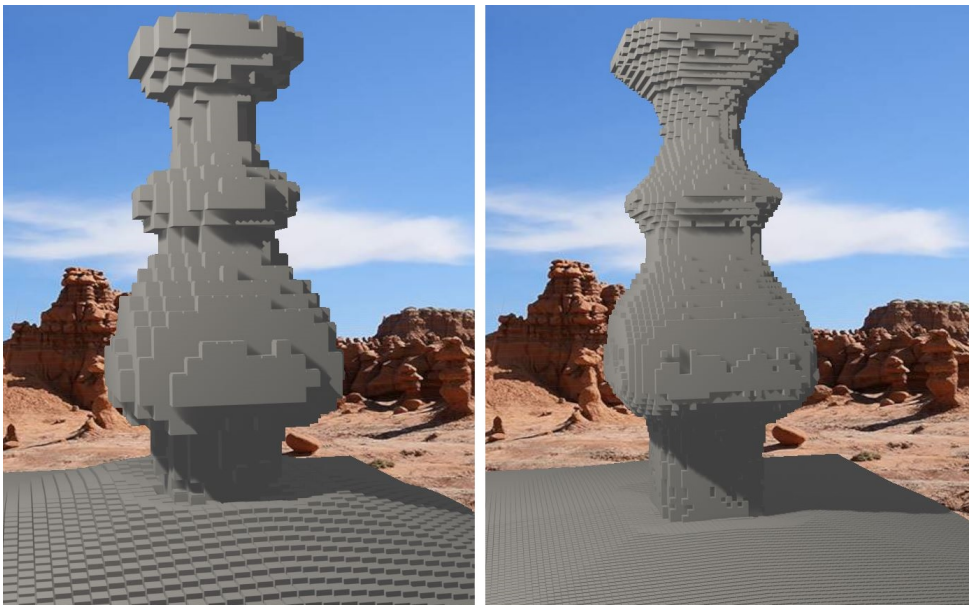


Figure 5.4: The same goblin created in our application and visualized in Blender, Left: 2x bigger scale, Right: 5x bigger scale.

5.2 Overhang

Some types of cliff overhangs suggest weathering and erosion have been the cause of the current rock shape. A cliff called Overhanging Point, Bruce Peninsula National Park, Ontario, Canada will be our reference rock formation for the simulation of weathering and erosion on the rock. The overhang, photographed from below, is shown in Figure 5.5.

Figure 5.5: Overhanging Point, Bruce Peninsula National Park, ON, Canada (from [13]).

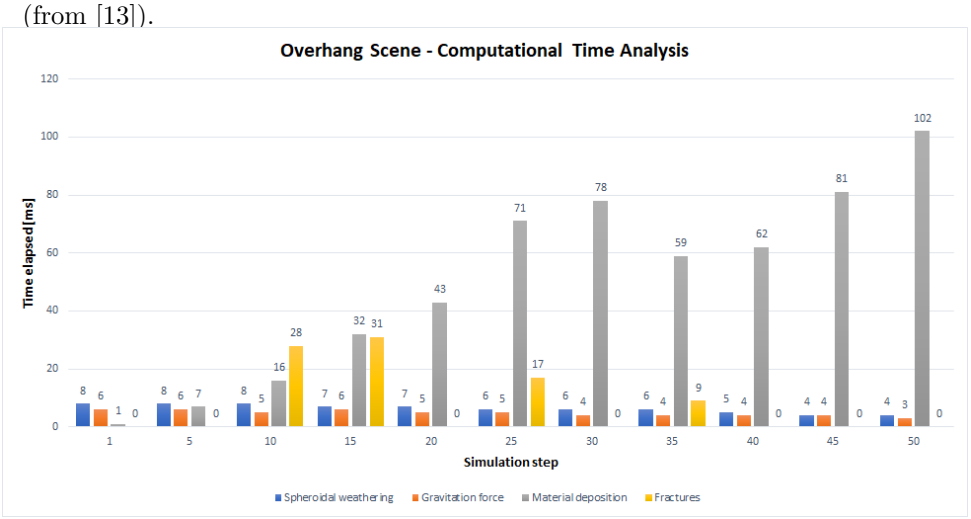


Figure 5.6: Overhang - elapsed time measured at each 5^{th} step.

A $10 \times 13 \times 30$ voxel grid was generated. All four implemented algorithms were applied during the simulation and their computational time is visualised

Spheroidal weathering	322 ms
Gravitation force	239 ms
Material deposition	2708 ms
Fractures	201 ms
Whole simulation	3881 ms

Table 5.3: Overhang - Elapsed time of each algorithm during the simulation.

in Graph 5.6. One can observe that the fractures creation and propagation algorithm took the least time from the whole simulation. One of the reason is that a fracture had a probability of creation equal to $\frac{4}{5}$. Moreover, a shear fracture is not generated each simulation step, therefore A* and backtracking are not applied each simulation step. The longest time spent on a fractures propagation algorithm is equal to 797ms at 29^{th} simulation step. At the initialization, the application used 52.3MB of RAM and approximately 26% GPU usage while the simulation was running.

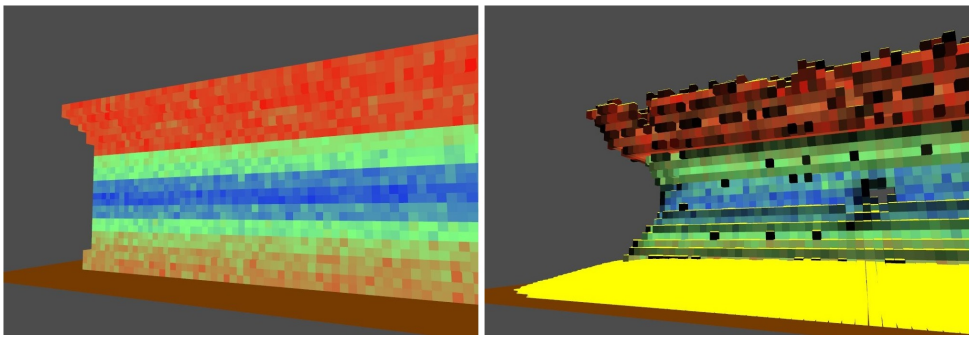


Figure 5.7: Left: The generated voxel grid, Right: The rock after 50 simulation steps.

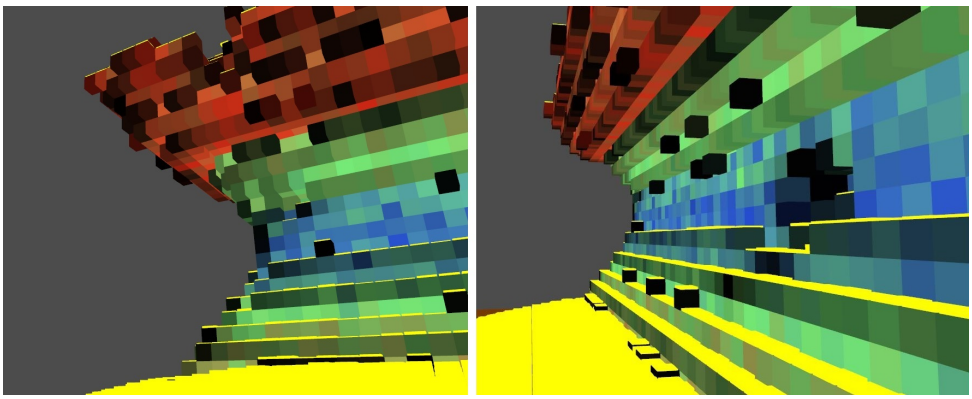


Figure 5.8: Closeups of the simulation results, rendered in our application.

Following Figures 5.9 and 5.10 compare our simulation and the real world rocks. Figure 5.9 compares the most famous overhang at Overhanging Point cliff with our simulation result, whereas Figure 5.10 compares the Overhanging Point cliff photographed from bottom with our interpretation. Unfortunately, the definition of our material deposition algorithm did not produce identical eroded material distribution, however, our deposition algorithm computes general material distribution according to one angle of repose without any other factors influencing the material deposition. An angle of repose adjustment might improve the similarity, but on the other hand, this adjustment would produce unrealistically big amount of material stored on relatively small area elsewhere in the voxel grid.

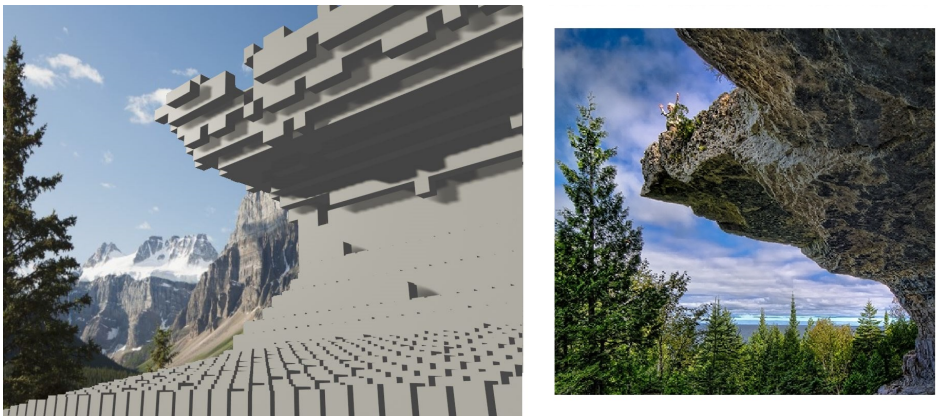


Figure 5.9: Left: Closeup of the rock visualised in Blender, Right: The reference picture (from [14], edited).

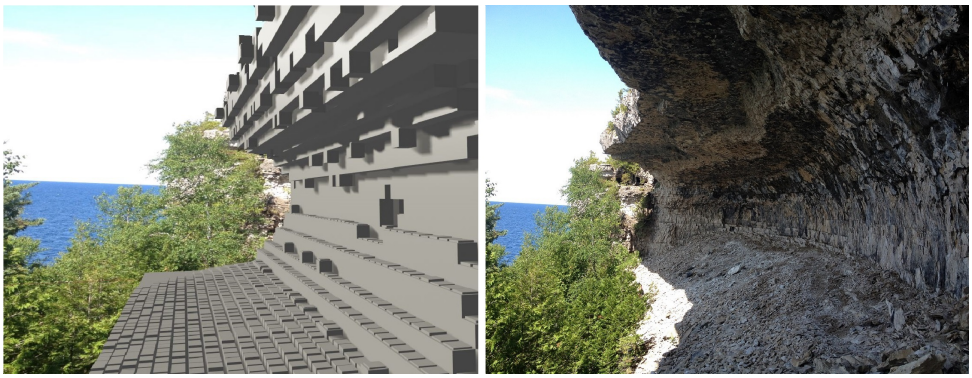


Figure 5.10: Left: Closeup of the rock visualised in Blender, Right: The reference picture (from [13], edited).

5.3 Arch

Weathering and erosion are able to produce monumental rock formations, such as the Delicate Arch in Utah, USA. Our application did not have a mode to simulate arches, however, a block-shaped grid can be generated in the Goblin scene and fractures creation and propagation algorithm might create ideal conditions for an arch-shaped rock. In our case, a crack leading through the grid causes gradual material erosion along the fracture, resulting into enlargement of the crack. Coincidentally, similar process of the creation of arches was presented in the study by J. Bruthans [9]. In this case, $8 \times 13 \times 20$ voxel grid was generated with a scale of 0.5 (i.e. 2x denser grid). Graph 5.11 shows the elapsed time of each algorithm at each 5^{th} step from a total of 52 simulation steps. Our results are presented in Figures 5.12 and 5.13. The application used 42MB RAM and approximately 25% of GPU usage.

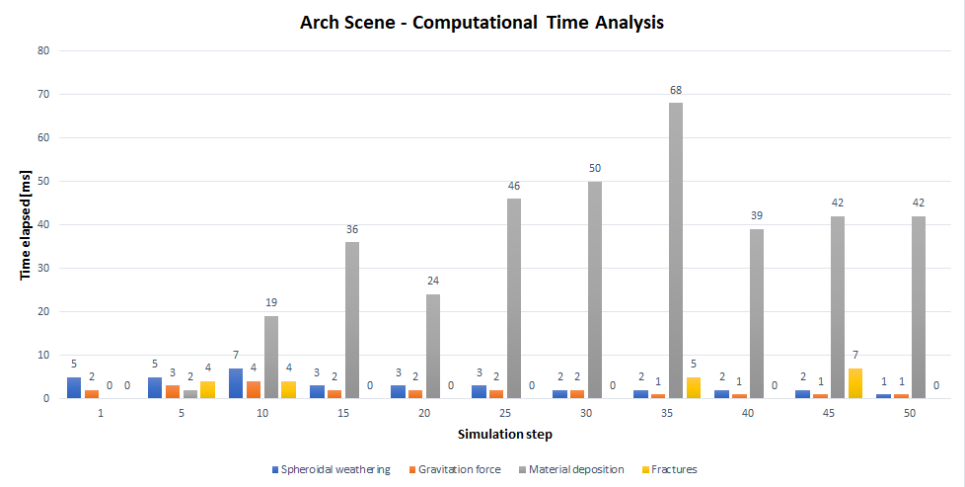


Figure 5.11: Arch - elapsed time measured at each 5^{th} step.

Spheroidal weathering	149 ms
Gravitation force	99 ms
Material deposition	1691 ms
Fractures	110 ms
Whole simulation	2383 ms

Table 5.4: Arch - Elapsed time of each algorithm during the simulation.

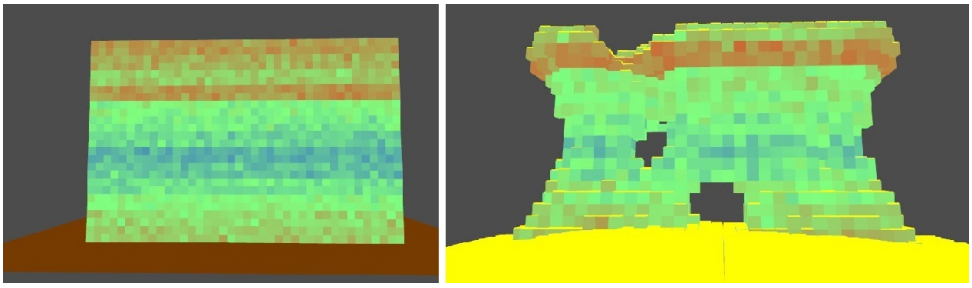


Figure 5.12: Left: The generated voxel grid, Right: The rock after 50 simulation steps.

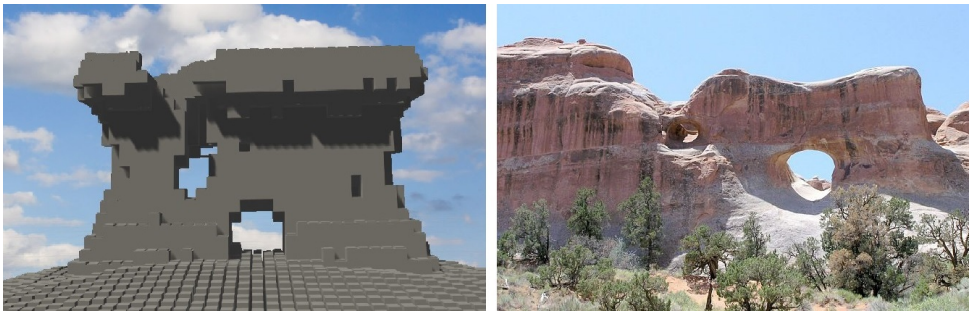


Figure 5.13: Left: The rock visualised in Blender, Right: The reference picture (from [15], edited).

5.4 Rock City

The most difficult scene to re-create is the rock city. While simulating the rock city, the emphasis was given to the crack generation. The fractures generation at rock city scene is increased 10 times in comparison with previous simulation scenes. Our results are the least precise since from all previous simulation results since the shape of individual rocks varies a lot as well as the crack locations on the rock surface.

Spheroidal weathering	284 ms
Gravitation force	183 ms
Material deposition	1990 ms
Fractures	2435 ms
Whole simulation	5685 ms

Table 5.5: Rock city - Elapsed time of each algorithm during the simulation.

The generated grid had dimensions of $10 \times 10 \times 12$, 5x bigger scale was used for this scene. Time needed to finish the simulation is presented in Graph 5.14 and our simulation results are shown in Figures 5.15 and 5.16. Our application used 103.3MB RAM and approximately 29% GPU usage while rendering the rock city scene.

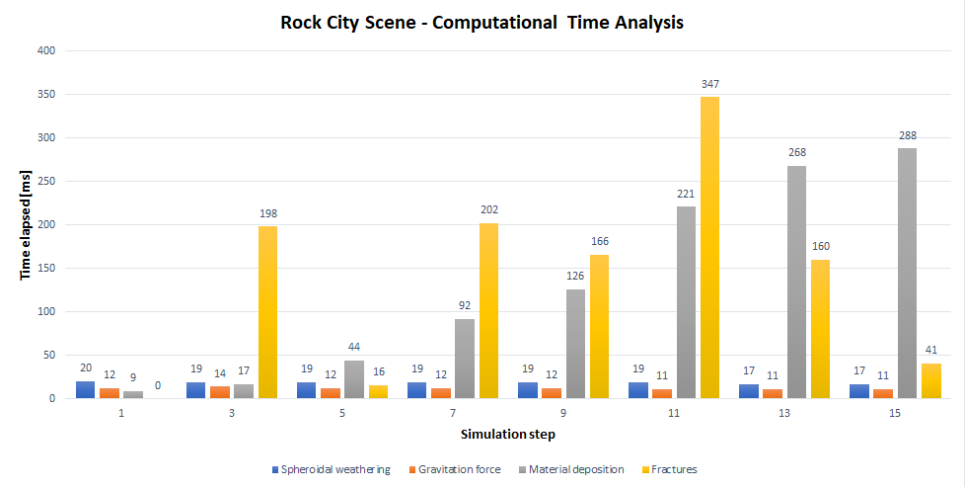


Figure 5.14: Rock city - elapsed time measured at each odd step.

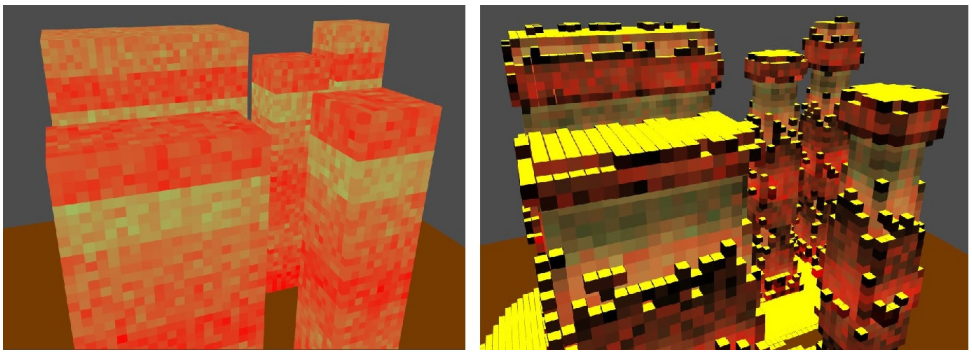


Figure 5.15: Left: The generated voxel grid, Right: The rock after 15 simulation steps.

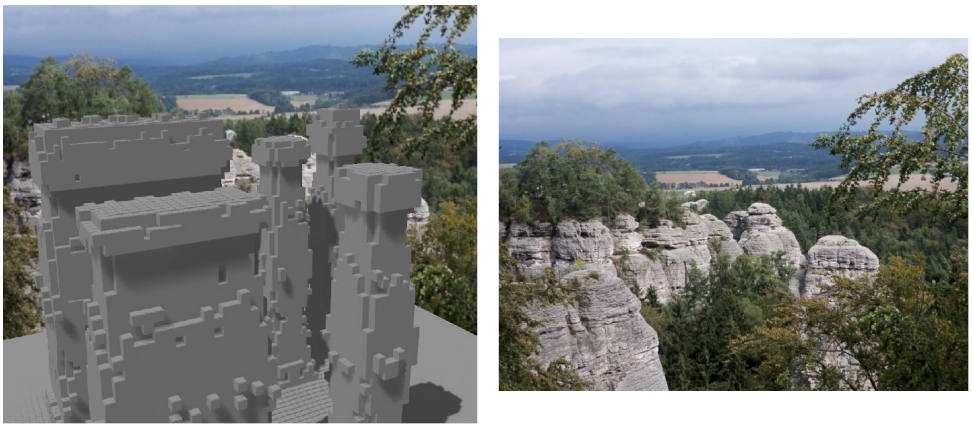


Figure 5.16: Left: The rock visualised in Blender, Right: The reference picture (from [16], edited).

5.5 Memory requirements and complexity evaluation

In comparison with other spheroidal weathering simulation methods, our method runs acceptably fast. On the other hand, previous simulation methods were tested on less powerful machines than our machine used for testing, so the comparison might be inaccurate. However, our simulation uses an algorithm implementing gravitation force impact on the rock, as well as the fractures creation and propagation algorithm, which have not been implemented in any method introduced in Chapter 3. The drawback of our algorithms is that they are not paralleled. The complexity of our algorithms is presented in Table 5.6, where, for fractures, the complexity refers to A* algorithm.

Algorithm	**Time complexity**
Spheroidal weathering	$O(N)$
Gravitation force	$O(N)$
Material deposition	$O(N)$
Fractures	Worst case $O(b^d)$

Table 5.6: Time complexity of our algorithms, N is the number of voxels, b is the branching factor and d is the length of the path.

A comparison of our implementation of spheroidal weathering algorithm and an implementation presented by Jones et al. [2], is shown in the following Table 5.7. It is crucial to mention that the computational time of their spheroidal weathering implementation uses a surrounding area with radius from 3 to 7 to compute the durability change of each voxel, whereas our implementation computes the durability change by visiting neighbouring voxels.

Implementation	**Time per step, approximately 24000 voxels**
Our method	2ms
Method from [2]	Varying from 0.2s through 3s

Table 5.7: Spheroidal weathering algorithm - computational time comparison.

Considering the memory requirements, our application is more memory extensive than other simulations. For a comparison, our application uses 1228.1MB RAM to render a scene with 10 253 250 voxels, whereas the simulation presented by M. Beardall et al. [10] uses 793 MB to render a scene with 10M voxels. To defend our 33% higher memory requirements, it is necessary to mention that their simulation uses only a spheroidal weathering algorithm. Our application uses three more algorithms requiring different variables per voxel, which eventually results into higher memory requirements.

The reader might have noticed that in Tables from 5.2 through 5.5 the computational time of the whole simulation does not match the sum of the computational time of each algorithm. The difference is caused by model matrices update of visible voxels after a simulation step is finished. The model matrices are not updated after each voxel is modified in order to save computational time, since one voxel might change several times during one simulation step.

Chapter 6

Conclusion

The goal of this thesis was to create an application that is able to simulate spheroidal weathering impact and eroded material sedimentation using C++ and OpenGL. In order to achieve more realistic simulation results, gravitation force impact was added to the simulation. Moreover, the problem of fractures creation and propagation within the rock was addressed and added to the simulation. Emphasis was placed on physical reality during the application development. Such requirement is crucial in simulation of real-world objects or events, however, maintaining physical reality and at the same time keeping the simulation running in real-time proved problematic and challenging.

6.1 Future work and possible improvements

The spheroidal weathering algorithm works as expected and also produces similar results as results of methods presented in Chapter 3. However, improvements can be suggested in order to speed up the algorithm. Caching mechanism can be used to avoid recomputing the voxel's durability loss if there were no changes to the neighbouring voxels. On the other hand, the time savings thanks to such caching mechanism might be lost due to the gravitation force propagation algorithm, because any changes in the voxel grid are propagated throughout the whole grid.

The deposition algorithm behaviour is strictly defined by the angle of repose and surrounding voxels. Increasing the amount of moved material may cause oscillation in the algorithm and lead to even longer computational time, if not an infinite loop. However, an improvement might be suggested. The wind impact might be also an factor while defining where the eroded material will be moved. Seasonal wind then may cause the eroded material to accumulate in the lee of the rock and the wind-exposed rock will hold less material.

Gravitation force propagation throughout the grid has been implemented and discarded several times, because the results have not been satisfying or the behaviour of the gravitation force was unrealistic. A simpler solution to gravitation force propagation was introduced in Section 4.6. Understanding the stresses withing the rock seems to be the key to implementing the gravitation force more realistically.

The fractures propagation technique has been optimized with A* search algorithm, nonetheless the fracture creation is a slow process, where every voxel along the path needs to be processed and investigated in order to find possible and valid fracture direction. The voxel grid also limits the fracture direction down to six, causing unnatural sharp edges to appear along the fracture. The fractures realism is also limited by the stresses within the rock - an implementation of realistic tension may create more realistic fractures.

A voxel grid with a bigger scale increases the precision of simulation in comparison with a voxel grid with lower scale on identical rock dimensions. Using even bigger scales in our application would require parallelism of our algorithms.

Finally, the data export quality might be significantly improved by implementing the marching cubes algorithm, which might be used in this case to obtain relatively smooth surface in comparison with the sharp edged voxel grid.

Bibliography

[1] Mark Herreid. Sandstone rock formation (hoodoo) at goblin valley state park in utah. Photography, Online, https://www.shutterstock.com/cs/image-photo/sandstone-rock-formation-hoodoo-goblin-valley-134001290.

[2] Michael Jones, McKay Farley, Joseph Butler, and Matthew Beardall. Directable weathering of concave rock using curvature estimation. *IEEE transactions on visualization and computer graphics*, 16:81–94, 03 2010.

[3] V. Fukal. Národní park České Švýcarsko. Photography, Online, https://www.kctul.cz/destinace/img/ceske_svycarsko_1.jpg.

[4] Pamela J. W. Gore. *Weathering.* Georgia Perimeter College, 1995. https://web.archive.org/web/20130510224332/http://facstaff.gpc.edu/~pgore/geology/geo101/weather.htm.

[5] Earlham College. Frost weathering visualization. Online, https://legacy.earlham.edu/~debowke/frost%20wedging.htm.

[6] T.N. Hien. Hoodoo at goblin valley state park, 2019. Photography, Online, https://neihtn.wordpress.com/tag/goblin-valley-state-park/.

[7] Matlok. Altdahn castle in the palatinate forest, germany, 2019. Photography, Online, https://commons.wikimedia.org/wiki/File:Wabenverwitterung_auf_Burg_Altdahn.jpg.

[8] Yixin Liu, Jiang Xu, and Gang Zhou. Relation between crack propagation and internal damage in sandstone during shear failure. *Journal of Geophysics and Engineering*, 15(5):2104–2109, 06 2018. https://doi.org/10.1088/1742-2140/aac85e.

[9] Jiri Bruthans, Jan Soukup, Jana Vaculíková, Michal Filippi, Jana Schweigstillová, Alan Mayo, David Mašín, Gunther Kletetschka, and Jaroslav Řihošek. Sandstone landforms shaped by negative feedback between stress and erosion. *Nature Geoscience*, 7, 07 2014.

[10] Matthew Beardall, McKay Farley, Darius Ouderkirk, Jeremy Smith, Michael Jones, and Parris Egbert. Goblins by spheroidalweathering. *Natural Phenomena*, pages 7–14, 01 2007.

[11] Adrien Peytavie, Eric Galin, Jérôme Grosjean, and Stéphane Mérillou. Arches: a framework for modeling complex terrains. *Computer Graphics Forum*, 28, 04 2009.

[12] McKay Farley. Fast spheroidal weathering with colluvium deposition. In *Master of Science Thesis*, 2011.

[13] Unknown. Overhanging point along the bruce trail. Photo taken: 30 July 2014, `https://commons.wikimedia.org/wiki/File:Overhanging_Point_along_the_Bruce_Trail.JPG`.

[14] Tony Paine. Overhanging point along the bruce trail. Photo taken: 2010, `https://tonypainephoto.com/artwork/1911571_Overhanging_Point_Bruce_Peninsula.html`.

[15] Daniel Mayer. Tunnel arch in arches national park. Taken: June 2005, `https://commons.wikimedia.org/wiki/File:Tunnel_Arch_in_Arches_NP.jpeg`.

[16] Jan Karhánek. Hrubá skála, Český ráj. `http://hany.info/cz/galerie/cesky-raj-hruba-skala-hruboskalsko-trosky-piskovec-veze-skalni-mesto.html`.

[17] T. C. Hales and Joshua J. Roering. Climatic controls on frost cracking and implications for the evolution of bedrock landscapes. *Journal of Geophysical Research: Earth Surface*, 112(F2), 2007.

[18] M. Milligan. *The Geology of Goblin Valley State Park*. Public Information Series. Utah Geological Survey, 1999.

[19] A. V. Turkington. Cavernous weathering in sandstone: lessons to be learned from natural exposure. *Quarterly Journal of Engineering Geology and Hydrogeology*, 31(4), 11 1998.

[20] S Marshak and BA van der Pluijm. *Earth Structure*. W. W. Norton & Company, 2004.

[21] L. Tham, L. Li, Y. Tsui, and P.K.K. Lee. A replica method for observing microcracks on rock surfaces. *International Journal of Rock Mechanics and Mining Sciences*, 40:785–794, 07 2003.

[22] Urban Akesson, Jan Hansson, and Jimmy Stigh. Characterisation of microcracks in the bohus granite, western sweden, caused by uniaxial cyclic loading. *Engineering Geology - ENG GEOL*, 72:131–142, 03 2004.

[23] Axel Paris, Eric Galin, Adrien Peytavie, Eric Guérin, and James Gain. Terrain amplification with implicit 3d features. *ACM Transactions on Graphics*, 38, 09 2019.

[24] Joey de Vries. Learn opengl tutorial. `https://learnopengl.com/Advanced-OpenGL/Instancing`.

Appendix A

User interface

Our application uses ImGUI library to provide the user interface and ImGUI add-ons provide the file explorer. All parts of UI will be discussed below. The camera movement is explained in Table A.1.

Hotkey	Description
W	Move forward
S	Move backwards
A	Move left
D	Move right
Space bar	Lock / Unlock camera view

Table A.1: Camera movement in the scene.

While the application is running, the user is provided an Activity Log window, where all the events and messages will be logged. The application log is shown in Figure A.1.

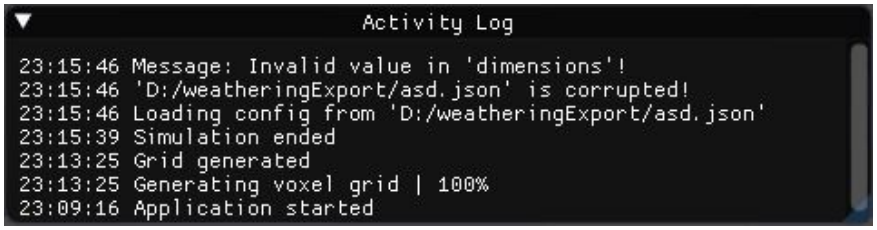


Figure A.1: The Application Log window.

The Main Menu is shown in Figure A.2 and is divided into three parts. The first part (1) consists of the scene selection, where the simulation type is specified. The user can choose one from three listed options. The second part (2) specifies the voxel grid scale and rock dimensions. There are four supported scales, available in the combo box: Default scale, 2x, 3x and 5x bigger scale. Rock dimension are unsigned integers in the following order: x, y, z, or rather $Width, Height, Depth$. Button (3) launches the simulation with the settings specified in (1) and (2). Button (4) opens a File explorer window, where the user can browse configuration files in their system. Button (5) closes the application.

Figure A.2: The Main Menu window.

A File browser window is captured in Figure A.3. This file browser is a part of the ImGui add-ons library. The layout of the window is similar to the file browser in Windows OS, therefore the functionality is very similar. The current path to the displayed folder is at the top of the window (1). Majority of the window is an area, where all the files and folders on the current path are listed (2). Below is a text field (3), where the opened or saved file name needs to be typed. Button (5) opens the currently selected file. Alternatively, double-click on a file can be used. Button (6) discards the dialog window.

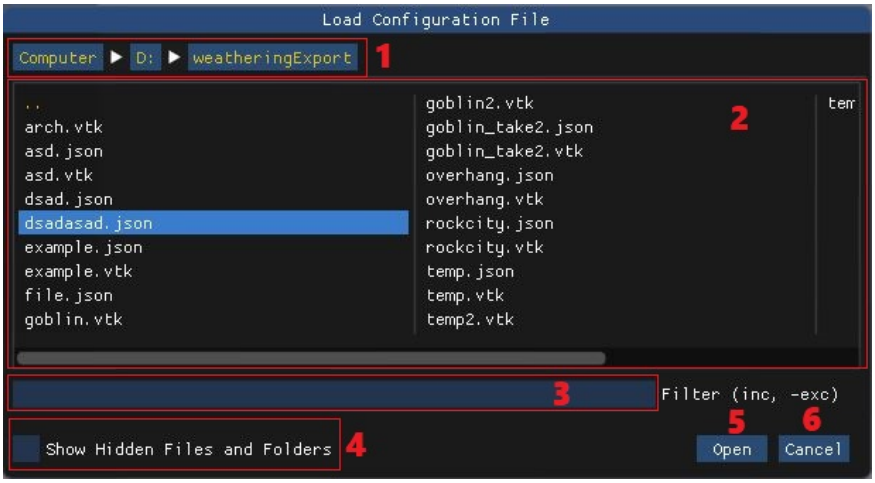


Figure A.3: File browser window.

Simulation tools are displayed while the simulation runs (Figure A.4).

- In the area (1), current simulation scene parameters are displayed.
- Toggle button (2) enables the rendering of voxel's durability.
- Toggle button (3) enables the fractures creation and propagation algorithm and simultaneously opens a fractures settings (10), where the fractures-only rendering mode can be enabled and the probability of a fracture creation can be adjusted.
- Toggle button (4) enabled the gravitation force algorithm.
- Toggle button in area (5) enables the automatically running simulation. If this toggle button is not ticked, a button SIMULATION STEP is displayed. This button launches one simulation loop. If the Automatic simulation toggle button is ticked, automatic simulation menu is displayed (11). Here the user can adjust the wait-time between the simulation steps by browsing available values in the combo-box.
- Button (6) opens a File explorer window and the user specifies, where a file with exported data from the simulation should be saved.
- Button (7) opens a File explorer and the user specifies, where the configuration file with current simulation parameters should be saved.
- Button (8) discards the current scene and returns to the Main Menu.
- Finally, Button (9) closes the application.

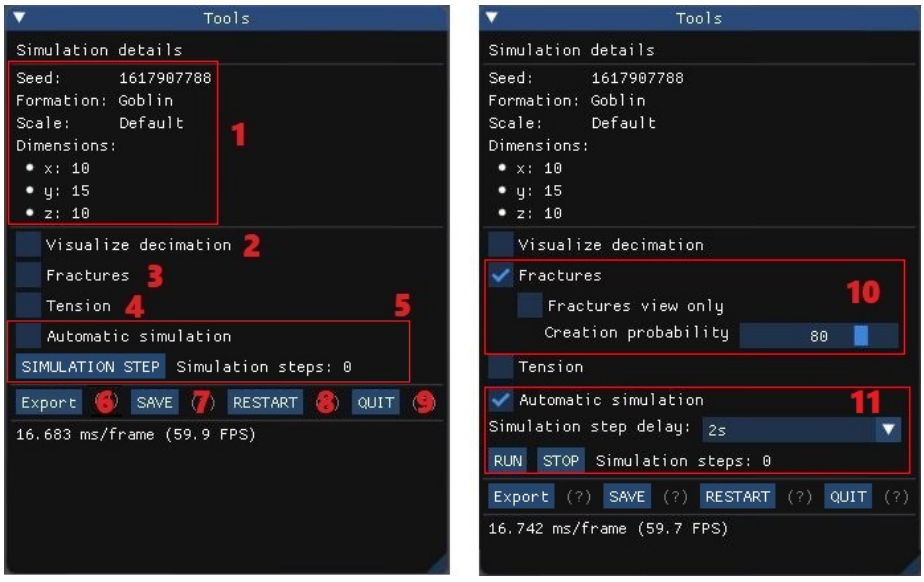


Figure A.4: Tools window is displayed when a simulation scene is rendered.

Appendix B

Third party programs references, application usage

B.1 Used third party programs and libraries

JSON parser, used for configuration files. Available on 'github.com/nlohmann/json'.

Dear ImGUI, a graphical user interface suitable for OpenGL. Available on 'github.com/ocornut/imgui'

ImGUI add-ons, add-on widgets for GUI library Dear ImGui. Used for File browser dialogues. Available on 'github.com/gallickgunner/ImGui-Addons'

ParaView, a third party software suitable for visualisation of volumetric data and export into modelling software. Available on www.paraview.org

Blender, a modelling software. Available on www.blender.org

PGR framework, providing basic OpenGL functionality. Available on cent.felk.cvut.cz/courses/PGR/framework.html

Overleaf, an easy to use, online, collaborative LaTeX editor. Available on www.overleaf.com

B.2 Usage and source code

Our application was developed in Visual Studio 2017 with MS Windows 10 and tested on several machines using Windows 10 operating system. However, a bug was found while testing the application on Windows 7 and Windows 8/8.1 operating systems. While in the simulation scene and with the movement Enabled (Space bar key), the camera movement does not response to any mouse movement. The issue was tracked and it was found that the callback for mouse movement detection `glMouseFunc` and `glPassiveMouseFunc` stalls the rendering pipeline, causing the `glTimerFunc` not to redraw the frames. This issue does not appear on any Windows 10 operating system and probably refers to changes in the architecture. Finally, the bug does not have impact on the simulation results whatsoever, it only disables the camera movement in the scene. Therefore, any computer running MS Windows 10 is recommended to use while running our application.

In order to compile our application using the provided source code, it is necessary to download and include a PGR framework first (link is provided in Appendix B.1). After opening the Visual Studio project, the project properties need to be modified in order to include all PGR-framework related dependencies. Then the source code can be successfully compiled for a 32bit version. In case of issues while compiling a Release 32-bit version, the PGR-framework libraries need to be re-compiled.

The doxygen-like style source code documentation can be found within the attachments. A HTML and LaTeXversions are provided for this purpose. Note that majority of included classes listed in the documentation are GUI related and are part of the ImGui library and ImGui add-ons.

Appendix C

List of attachments

- This thesis as a PDF file
- An executable application
- A folder containing the source code of our application
- Doxygen documentation of the source code
- Screenshots of the application