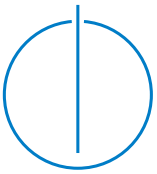
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Implementation of a method for hydraulic erosion

Hans Theobald Beyer



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

Implementation of a method for hydraulic erosion

Implementierung einer Methode zur hydraulischen Erosion

Author:	Hans Theobald Beyer
Supervisor:	Prof. Dr. Rüdiger Westermann
Advisor:	M. Sc. Florian Ferstl
Submission Date:	15.11.2015

I confirm that this Bachelor's Thesis in Informatics: Games Engineering is my own work and I have documented all sources and material used.

Munich, 15.11.2015 Hans Theobald Beyer

Abstract

The creation of realistic looking terrains is an important part in the movie industry as well as in computer games and simulations. Realistic and visually appealing landscapes help the consumer to immerse themselves in the virtual world. One of the greatest challenges of generating terrain are the effects of erosion. Those effects can be found everywhere in nature and are a vital part of natural looking terrain.

After a foundational research in the field of erosion algorithms, I wrote an highly adaptable algorithm to simulate hydraulic erosion on a heightfield. This algorithm excels through its various parameters, which allow several results for the same input. Through its interactivity it allows the user to erode the terrain according to its needs.

Das Generieren realistisch wirkender Landschaften ist ein wichtiger Bereich sowohl in animierten Filmen, als auch in Computerspielen und -simulationen. Naturnahe, optisch ansprechende Landschaften helfen dem Konsumenten in eine virtuelle Welt einzutauchen. Eine der größten Herausforderungen beim Generieren eines Terrains stellen dabei die Erosionseffekte dar. Da sie überall in der Natur sichtbar sind, sind sie unabdingbar für ein realistisch wirkendes Terrain.

Aufbauend auf einer grundlegenden Analyse der bestehenden Erosionsalgorithmen habe ich einen sehr anpassbaren Algorithmus zur Simulation von hydraulischer Erosion auf einem Höhenfeld geschrieben. Dieser Algorithmus zeichnet sich durch seine unterschiedlichen Simulationsparameter und somit eine Vielfalt möglicher Lösungen aus. Dank seiner Interaktivität erlaubt er dem Benutzer eine Landschaft seinen Anforderungen entsprechend möglichst naturnah zu erodieren.

Contents

1 Introduction

Beautiful landscapes are a feature in many computer games today and especially role-playing games often present huge areas of outside terrain. Those terrains need to look realistic and appealing on the one hand but on the other hand they often have to fulfill several requirements defined by the games mechanics. The latter can often be only achieved by hand-sculpting the terrain. Unfortunately it is very difficult and time-consuming to sculpt realistic looking terrain from hand since the human brain is familiarized with a lot of real landscapes and notices even slight deviations in style pretty fast.

One major aspect that formed the distinctive look of landscapes on our planet is erosion caused by wind and water. It is known in computer graphics that the results of such processes are crucial for a realistic looking landscape. Figure 1.1 shows an example of a procedurally generated mountain. Erosion marks are clearly visible.

To simplify the processes of generating and modeling landscapes and terrains erosion algorithms have been introduced. The first erosion simulation algorithms were developed more than 25 years ago and since then different approaches have been made to improve the results.

This thesis will give a short overview over previous work in this field of study and present a simple particle based algorithm for hydraulic erosion.

1.1 Definition of task

The goal of this bachelor's thesis is to provide an algorithm that simulates hydraulic erosion on an arbitrary heightmap on the basis of foundational research about the field of erosion algorithms.

The result shall be visually appealing and natural looking, however it is not goal to run a fully physically based fluid simulation. To create good looking results it will be necessary to make the algorithm adjustable to hightmaps with different characteristics.

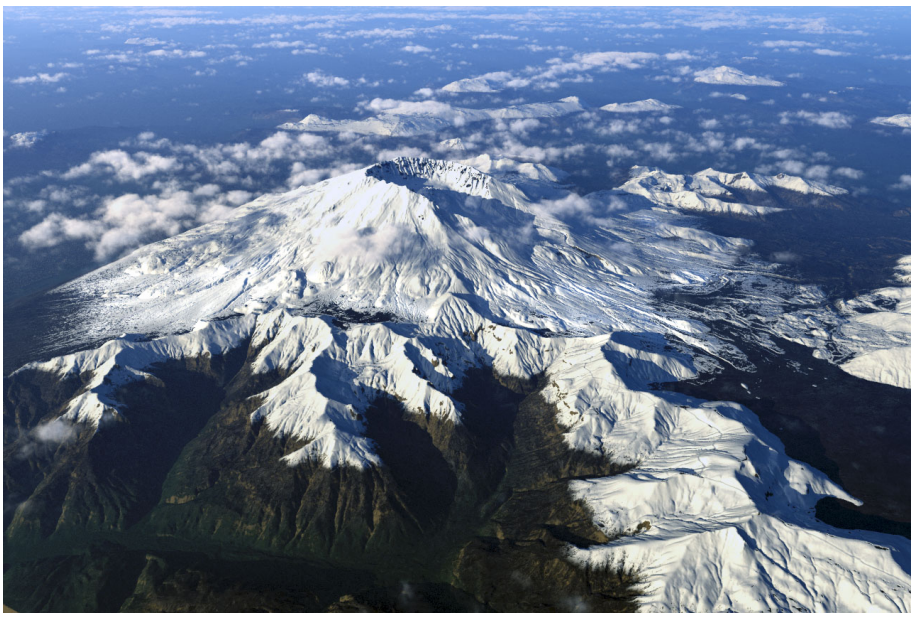


Figure 1.1: Procedurally generated and eroded mountain terrain. Generated and rendered with Terragen 3 [LLC].

1.2 Outline of the thesis

This first chapter introduces the topic and defines the task.

Chapter two defines and explains various terms which are used in the thesis.

The third chapter gives an overview over the history as well as current state of research of erosion algorithms.

In chapter four a comparison between the two most used approaches, the grid based and the particle based approach, is given with strengths and weaknesses.

Chapter five explains my implementation of an particle based erosion algorithm as well as failed attempts and improvements I made during development.

In chapter six the algorithm is evaluated concerning implementation, performance and results.

In the last chapter ideas and approaches for future work are given.

2 Terms and Definitions

This chapter describes important terms and definitions which are needed for the following chapters of the thesis.

Map

A map is a data structure that represents values on a grid. The values are saved in an array representing a two dimensional uniform grid with dimensions $size_x$ and $size_y$. Each value stored in the array is allocated to one grid point and every grid point has exactly one value. The value at point (x, y) is saved at array index $x + y \cdot size_x$.

Heightmap

A heightmap is a map containing float values used to describe terrain. The value at each grid point represents the elevation of the terrain at the coordinates (x, y), it can be seen as the grid points z-coordinate. The values usually are normalized to a range of $[0, 1]$.

Mesh

A mesh consists of triangles, defined by three vertices each. Those triangles represent flat surface parts of a 3D-Model.

3 Related Work

Procedurally generated terrain has been a topic of research nearly since the beginning of 3D computer graphics. The foundation for procedurally generated content were the fractal approaches given by Benoît Mandelbrot in [Man82] in 1982. After he recognized similarities between fractional Brownian motion (fBm) and the silhouette of mountains, he extended the process to two dimensions to create a Brownian surface, which resembled a mountainous landscape. In the same year the well known diamond-square algorithm, also known as 'random midpoint displacement', was introduced by Fournier et al. in [FFC82], however both methods were accused of being flawed since generated mountains and valleys showed similar features and the terrain has the same visual characteristics when mirrored upside-down. This particular feature can not be found in nature because of the effects of erosion.

After an approach to model terrain backwards from random generated stream networks [KMN88] the first algorithm was introduced by Musgrave et al. [MCM89] to simulate hydraulic erosion on a terrain in 1989. The introduced procedure adds water to the vertices of a terrain mesh generated from a heightmap. This water then flows to all lower neighbouring vertices. During this process sediment is taken from higher vertices and distributed to lower ones as it occures in nature.

In 1999 Chiba et al. introduced the first step to particle based approaches in their velocity field based algorithm [CMF99]. In their simulation they move water particles downhill along the velocity field resembling the motion of water. This vector field is obtained by the terrain and the particle movement.

A layered data representation was introduced by Beneš and Forsbach in 2001 in [BF01]. It allows to define various layers of material each with different properties like hardness or grain size. Each layer is affected differently by the erosion processes presented in the paper. The layers are represented as multiple heightmaps which are stacked on top of each other. This model is faster and requires far less memory than the slightly more accurate voxel based approaches. An example of an erosion model with a layered material representation can be seen in figure 3.1 from [Šta+08].

Beneš et al. presented an algorithm based on fluid simulation with the Navier-Stokes

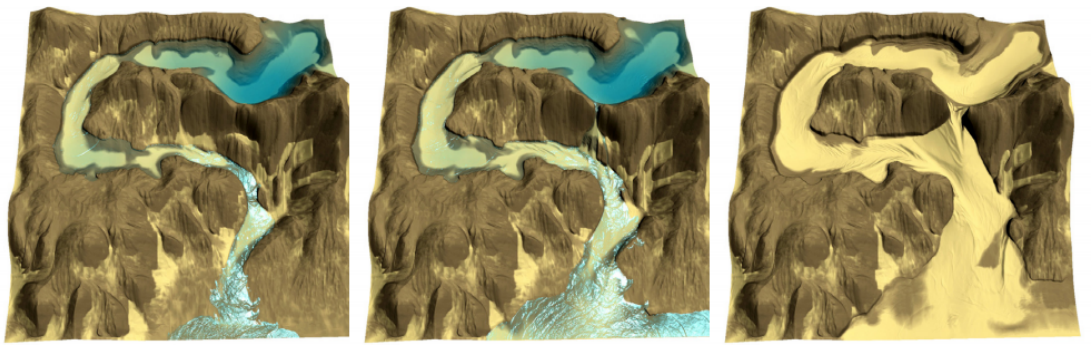


Figure 3.1: Example of hydraulic erosion on a uniform heightfield using a grid based fluid simulation. Image from [Šta+08].

equations in [Ben+06] in 2006. This model is the first real physics based approach and implemented in 3D. The environment is represented as a voxel grid which allows the creation of overhangs and cave systems. It is very detailed an excellent to simulate small scale erosion processes, however on large scale terrains those fine details are neglectable compared to a similar 2.5D implementation, which is much faster and requires less memory.

One year later Beneš [Ben07] introduced simultaneously with Mei et al [MDB07] a new hydraulic erosion simulation working in realtime. Both use shallow water equations for the water flow, but Beneš also uses a shallow water model with high viscosity for his grit representing layer. The fact that these simulations can run in real-time makes them importent for interactive terrain modeling.

Another approach was presented in 2009 by Krištof et al. [Kri+09]. The technique combines smoothed particle hydrodynamics (SPH) as an Lagrangian approach with an Eulerian erosion model. However the terrain is represented as an uniform heightfield and thus again no concave features like overhangs or caves are able to form. In their chapter '6. Conclusion and Future Work' they state that one future work is to extend the model to a full 3D representation.

Very recently Skorkovská et al. [SKB15] did exactly that and extended the SPH approach to work on arbitrary triangle meshes, which now allows the creation of concave formations. That way not only overhangs could be generated, but also interesting structures like underground cave systems. The presented method requires far less memory than the volumetric based approaches in e.g. [Ben+06], but entail new challenges like inconsistencies in the mesh.

4 Comparison of grid based and particle based approaches

Over the years two distinctive methods for hydraulic erosion have been established. The cell based or grid based approach was introduced by Musgrave et al. [MCM89] and used in [Ben+06], [Ben07] and [MDB07]. In this approach the properties of water and terrain are stored in cells, which interact with their neighbouring cells. These cells can either be created in two dimensions as in the shallow water models in [Ben07] and [MDB07] or in three dimensions, as in [Ben+06].

The particle based approach breaks away from that grid and uses particles to represent the water. This approach was introduced first by Chiba et al. [CMF99] and used by the approaches using SPH models [Kri+09] and [SKB15]. It is important to note here, that only the simulated water is represented by particles. The representation of the terrain can still be on a grid basis.

In grid based techniques the available simulation space is split into cells. Every cell keeps track of how the fluid behaves inside it, namely how much water is distributed to neighbouring cells and how much water it gets from its neighbours. From this information a new state is calculated every time step. This method to look at defined spatial points to track the fluid is called an Eulerian viewpoint. Compared to the particle based simulation methods this one has a higher accuracy but at the cost of longer computation times. One disadvantage of a grid based method is that for every time step the whole simulation space has to be considered and every cell has to be calculated. It is therefore not scaleable if the water is momentarily only at a small area. Also the conservation of mass is not easy to accomplish in this method. If only a primitive grid based algorithm is used, it can lead to ravines only or mostly extending along the grid axes.

With particle based techniques, called Lagrangian methods, the simulated fluid is represented by particles which store their position, velocity and sometimes their mass. Other properties, like the carried sediment in erosion models, can be saved in the particle too. The particles are then moved according to their properties. In many

models the particles interact with each other. This method is usually faster then the Eulerian approach and it also conserves the mass of the fluid naturally as long as no particle is deleted. A great advantage is that the performance of the algorithm doesn't depend on the size of the simulation space. Areas in which no particles are located don't need to be updated since every particle is only interested in its closest surroundings. On the downside it is less accurate than the Eulerian approach for fluid simulations.

Since terrain erosion happens on a large scale, the accuracy of the method is neglectable. I choose to implement a particle based method, because the particle based approach loses its weakness if accuracy is not considered, while the cell based method loses its benefit.

5 Particle based Erosion on a Heightmap

The algorithm I implemented was inspired by [Vol]. It is a particle based approach in which single drops are placed onto the map, run downhill and move material depending on their carry capacity and speed of motion. The drops are simulated in 2.5D, which means the drop is considered to be always at ground level and it only saves its position in two dimensions. Also the drops do not interact and are not simulated with physical principals. The special feature of this algorithm is that the particles move the same distance every simulation step although they are not bound to the grid. The simulated time per step is not consistent. Therefore this simulation is not suitable to simulate a fluid visually, its purpose is to simulate the impacts of water on the terrain.

The goal of the algorithm is to provide optically appealing erosion marks from running water on small scale as well as large scale terrain.

5.1 Particles

Every drop stores the following information: Its position *pos* on the grid as a two dimensional float vector, its current flowing direction *dir* as a two dimensional normalized float vector, its speed of motion *vel* as float value, the amount of water *water* it contains and the sediment *sediment* it carries both as float values.

The goal is to move a drop along the path of least resistance from its initial position to a local minimum in an efficient amount of steps. To move the drop in every iteration step the gradient g of the heightmap at the current position pos_{old} is interpolated with a simple bilinear interpolation considering gradients of the four surrounding grid points. Those gradients are depending on the position of the drop. E.g. for a position $pos_{old} = (x + u, y + v)$ with $u, v \in [0, 1[$ the gradients of surrounding points $P_{x,y}$, $P_{x+1,y}$, $P_{x,y+1}$, $P_{x+1,y+1}$ are defined as

$$g(P_{x,y}) = \begin{pmatrix} P_{x+1,y} - P_{x,y} \\ P_{x,y+1} - P_{x,y} \end{pmatrix}$$

$$g(P_{x+1,y}) = \begin{pmatrix} P_{x+1,y} - P_{x,y} \\ P_{x+1,y+1} - P_{x+1,y} \end{pmatrix}$$
$$g(P_{x,y+1}) = \begin{pmatrix} P_{x+1,y+1} - P_{x,y+1} \\ P_{x,y+1} - P_{x,y} \end{pmatrix}$$
$$g(P_{x+1,y+1}) = \begin{pmatrix} P_{x+1,y+1} - P_{x,y+1} \\ P_{x+1,y+1} - P_{x+1,y} \end{pmatrix}$$

If a bilinear interpolation is performed with these values, the gradient g in position pos_{old} can be defined as

$$g(pos_{old}) = \begin{pmatrix} (P_{x+1,y} - P_{x,y}) \cdot (1 - v) + (P_{x+1,y+1} - P_{x,y+1}) \cdot v \\ (P_{x,y+1} - P_{x,y}) \cdot (1 - u) + (P_{x+1,y+1} - P_{x+1,y}) \cdot u \end{pmatrix}$$

In [Vol] the gradient is always set to the interpolated middle point of the cell $(x + 0.5, y + 0.5)$. For accuracy reasons I decided to interpolate the gradient according to the drops position. I deliberately did not calculate the gradient in a continuous function to avoid looking up grid points outside of the cell the drop is located in.

This two dimensional vector g is then used to determine the new direction of motion dir_{new}. This direction is a blended value between g and dir_{old} defined by a parameter $p_{inertia}$ with a value between 0 and 1. 1 means g is not taken into account and the direction never changes and 0 means the previous direction dir_{old} is ignored and the new direction is along the negative gradient. This calculation is described in equation 5.1.

$$dir_{new} = dir_{old} \cdot p_{inertia} - g \cdot (1 - p_{inertia}) \tag{5.1}$$

In [Vol] a random direction is chosen if the new directions magnitude is below a threshold. This happens most often when a drop is initialized inside a flat cell. My algorithm only chooses a random direction if the direction otherwise would be 0. The direction of motion is then normalized.

The new position pos_{new} of the drop is calculated by simply adding the direction of motion dir_{new} onto the current position pos_{old} as described in equation 5.2. Since the direction dir_{new} always has a magnitude of 1, the drop moves exactly one unity during each step, regardless of its speed vel. This is because one step does not represent a fixed time span, but the way from position pos_{old} to position pos_{new}. If the drop would move a distance according to its speed vel, a fast drop can jump over one or more cells. That causes inconsistencies in the terrain in form of small hills if the drop should have eroded the part it jumped over, or small pits if the drop should have deposited

sediment in that cell. If a drop is slowing down it moves a very small distance and a lot of steps have to be calculated inside one single cell. Allthough that would probably be more accurate, the differences in the result are small enough to be ignored for the benefit of performance. A step size of one is the highest distance a drop can travel without the possibility of jumping over another cell.

$$pos_{new} = pos_{old} + dir_{new} \tag{5.2}$$

Now the difference in height between the new and the old position h_{dif} is calculated following equation 5.3 with h_{new} as the height of the terrain at position pos_{new} and h_{old} as height at position pos_{old}. The height values are again interpolated from the height values at the surrounding grid points.

$$h_{dif} = h_{new} - h_{old} \tag{5.3}$$

This height difference is now used to determine whether the drop moved downhill or uphill. If h_{dif} is positive and thus the new position pos_{new} is higher than the old position pos_{old}, sediment carried by the drop is deposited at pos_{old} to fill the pit the drop apparently ran through. If the drop carries enough sediment, the pit is filled, if it does not, it drops all its sediment. If h_{dif} is negative, p_{new} is lower than p_{old} and the new carry capacity c of the drop is calculated by means of h_{dif}, the current velocity vel, the amount of water the drop contains $water$ and the parameter $p_{capacity}$ as described in 5.4. If the height difference converges to 0 the capacity also would converge to 0, which leads to less erosion and more deposition in flatter areas. Sometimes however it is more aesthetic to erode flatter terrain too. For those cases the value $p_{minSlope}$ is a minimum value for h_{dif}. $p_{minSlope}$ can be used to prevent the capacity from falling too close to 0.

$$c = max(-h_{dif}, p_{minSlope}) \cdot vel \cdot water \cdot p_{capacity} \tag{5.4}$$

If the drop carries more sediment than it has capacity for, it drops a percentage of the sediment surplus defined by $p_{deposition}$ at position p_{old} as seen in equation 5.5. In my first attempts the drop always dropped all surplus sediment. That resulted in spikes everywhere a drop suddenly lost speed.

If the drop carries less sediment than its capacity c allows, it takes a percentage of its remaining capacity defined by $p_{erosion}$ from the map at position p_{old} (5.6). With $p_{erosion}$ the speed of erosion can be adjusted down. Important here is that the drop never takes more sediment then the height difference h_{dif}. Otherwise drops would be albe to dig holes, which is not wanted.

$$(sediment - c) \cdot p_{deposition} \tag{5.5}$$

$$min((c - sediment) \cdot p_{erosion}, -h_{dif}) \tag{5.6}$$

After that process the speed of motion is adjusted (5.7) and water is evaporated from the drop (5.8). The new speed of motion vel_{new} is calculated as the geometric mean of the squared old speed vel_{old} and the height difference h_{dif} as in [Vol]. vel_{old} is squared to give the speed inheritance more weight than the slope. h_{dif} is multiplied by the parameter $p_{gravity}$, which allows to adjust the gravity.

$$vel_{new} = \sqrt{{vel_{old}}^2 + h_{dif} \cdot p_{gravity}} \tag{5.7}$$

$$water_{new} = water_{old} \cdot (1 - p_{evaporation}) \tag{5.8}$$

This process is repeated until the drop moves out of the map or dies in a pit. To ensure a drop is not moved around endlessly, a maximum of $p_{maxPath}$ steps per drop is given.

5.2 Erosion and Deposition

If sediment is taken from the map and added to the drop, all n grid points P_i within the radius p_{radius} are taken into account. p_{radius} determines the area in which the drop erodes terrain. This covers the fact that no thermal erosion and no sediment slippage are simulated. If p_{radius} is 1, very thin ravines occur. In nature those would fill up with sediment braking from the walls and falling down into the ravines and smoothe their sides.

Every grid point looses sediment according to its weight w_i which is given to P_i. w_i is linear decreasing with its distance between P_i and the drop position pos as described in equation 5.9.

$$w_i = \frac{max(0, p_{radius} - (|P_i - pos|))}{\sum_{k=0}^{n} max(0, p_{radius} - (|P_k - pos|))} \tag{5.9}$$

All weights are normalized by dividing them through their overall sum to ensure the right amount of sediment is potentially taken.

When a grid point loses sediment according to its weight w_i, w_i is multiplied with the erosion factor. This factor determines how hard or solid the terrain at any given point in 3D space is. The value of the erosion factor is between 0 and 1. If it is 0, no erosion happens at all. If it is 1, the full erosion is happening. For values over 1 more

sediment than calculated would be removed and the results most certainly are neither realistic nor optically appealing.

The function computing the erosion factor takes the x, y and z coordinates of the grid point and returns a factor between 0 and 1. This is helpful if you have parts of your map, that should not be eroded, e.g. rocks in a sandy area or different layers of material. In this case the function could take a boolean map and erode just those grid points to which the corresponding value in the boolean map is true.

If sediment is deposited, it is distributed along the four grid points surrounding the current position. The distribution is calculated via bilinear interpolation. In this case no radius is used, because a depositing drop wants to probably fill a small pit of the size of one cell. Distributing sediment to more surrounding grid points would result in lifting the pit up, but not filling it.

5.3 Application of the changes

To avoid overly thin ravines, I started blurring the terrain after the erosion. With this approach details in the erosion effect as well as the terrain map itself were lost. After that I started to just blur only eroded parts of the terrain. That way details in the terrain map survived but the erosion effect still lacked of detail.

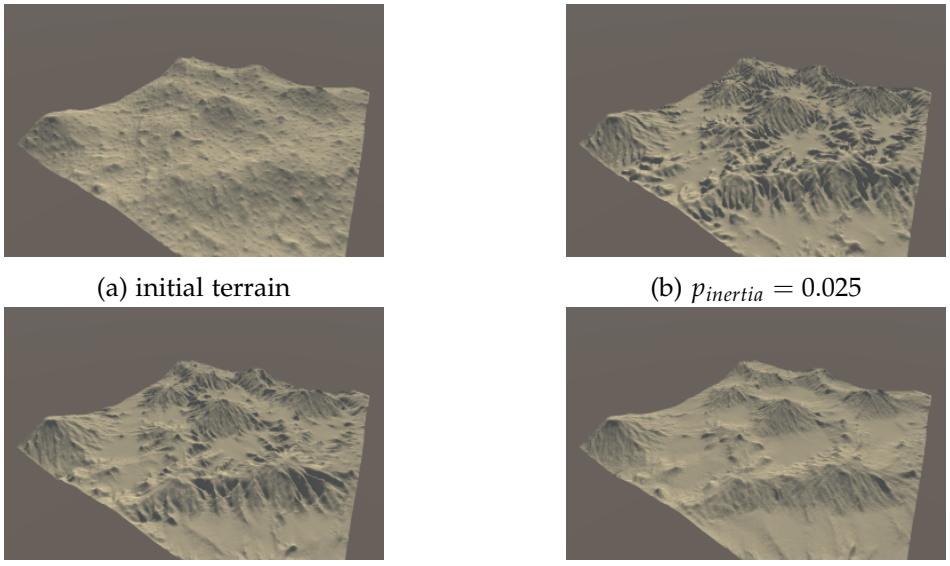
In the final algorithm the changes on the terrain are tracked during the whole process of erosion. That way the change map can be blurred before it is applied to the terrain map. As a last step the blurred change map and the unblurred change map are blended into one map with a factor b between 0 and 1. If $b = 0$, the blurred change map is discarded. if $b = 1$, only the blurred change map is applied. For values between 0 and 1 the change map is computed as described in equation 5.10. This effect is used in combination with a low p_{radius} since a high p_{radius} value prevents thin ravines itself.

$$map = b \cdot map_{blurred} + (1 - b) \cdot map_{unblurred} \tag{5.10}$$

5.4 Parameters

In the following section all parameters of the algorithm are explained and examples are given that show their effect on the terrain. The blend factor b mentioned in 5.3 is 0 for all examples in the following section.

Inertia



(a) initial terrain (b) $p_{inertia} = 0.025$

(c) $p_{inertia} = 0.1$ (d) $p_{inertia} = 0.4$

Figure 5.1: Example terrain eroded with different $p_{inertia}$ values. The terrain has a resolution of 512x512 samples. 300,000 drops were simulated in (b), (c) and (d).

The parameter $p_{inertia}$ determines the inertia of the simulated drops. Its value needs to be between 0 and 1. The closer $p_{inertia}$ gets to 0, the more valley and ravine like structures are formed. This effect occurs as a result of the drops flowing exactly downhill and thus digging existing valleys deeper. If the parameter exceeds a certain value, ravine like structures are only emerging on steep terrain parts. Instead hills are sharpened while lower parts of the terrain are flattened.

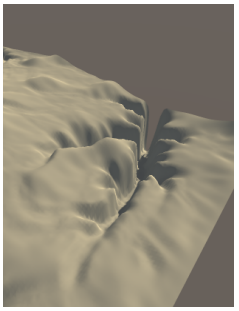


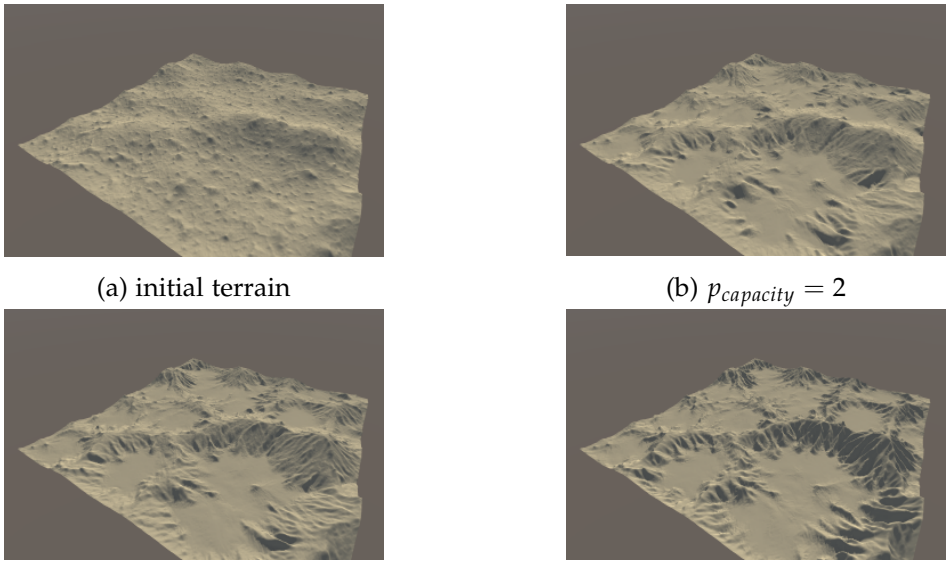
Figure 5.2

A low $p_{inertia}$ value can also lead to a drain valley effect, seen in figure 5.2. This effect describes the appearance of valleys growing from the edge of the heightmap in seemingly flat areas. They are caused by small pans at the edge of the heightmap that won't get filled up with sediment, because drops coming there leave the heightmap in the next

step and take their sediment with them. If such a valley emerges, every drop that follows the valley takes sediment and leaves the map without depositing it at the end of the valley.

One way to avoid this effect is to generate a larger map than needed, erode it and then cut out a part in the middle of the map. Since those valleys only occur at the edges, they will be cut off. In my approach the terrain height can not sink below 0. That way drain valleys can occur, but they will not give unrealistically deep ravines if the lower level of the terrain is held close to 0.

Carry capacity



(a) initial terrain

(b) $p_{capacity} = 2$

(c) $p_{capacity} = 8$

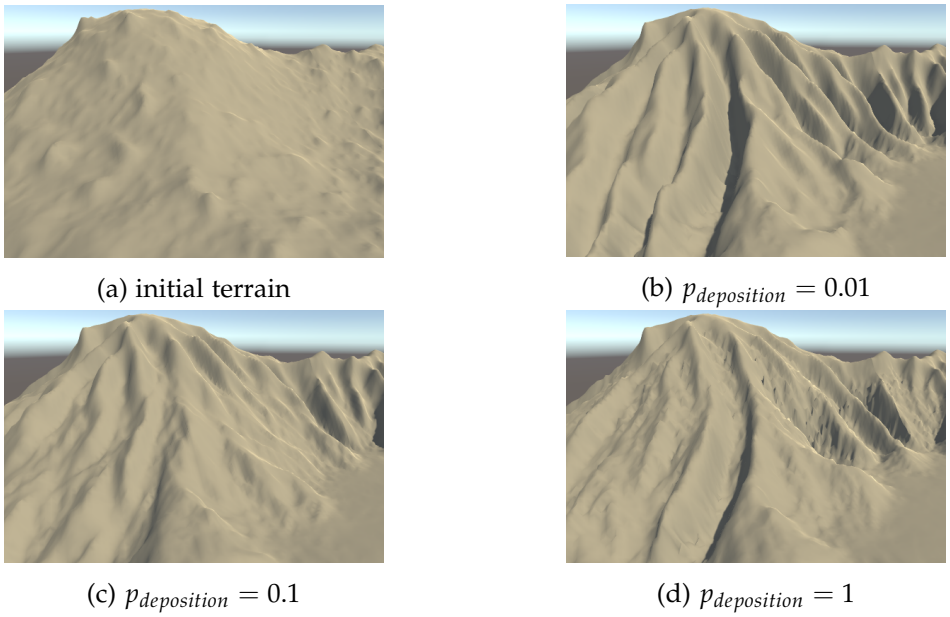
(d) $p_{capacity} = 32$

Figure 5.3: Example terrain eroded with different $p_{capacity}$ values. The terrain has a resolution of 512x512 samples. 300,000 drops were simulated in (b), (c) and (d).

The parameter $p_{capacity}$ determines the amount of sediment a drop can carry as used in equation 5.4. A higher value results in more sediment being eroded on steeper ground and deposited in lower regions. Thus each drop has a higher impact on the result. That leads to a ruggy terrain with more ravines. For the same erosion level with

lower carry capacity more drops are needed, but the result looks smoother.

Deposition speed



(a) initial terrain

(b) $p_{deposition} = 0.01$

(c) $p_{deposition} = 0.1$

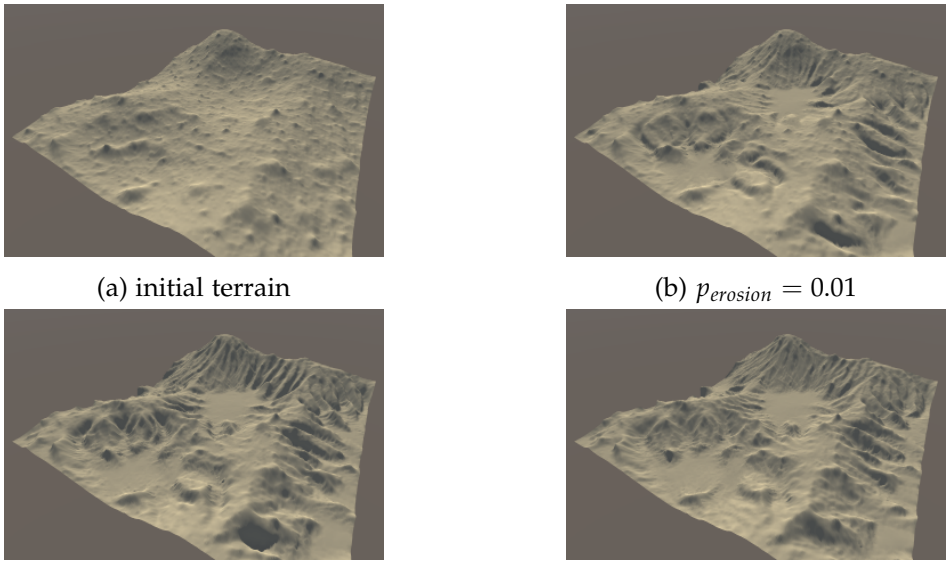
(d) $p_{deposition} = 1$

Figure 5.4: Example terrain eroded with different $p_{deposition}$ values. The terrain has a resolution of 256×256 samples. 75,000 drops were simulated in (b), (c) and (d).

The parameter $p_{deposition}$ limits the sediment that is dropped if the sediment carried by a drop exceeds the drops carry capacity c as described in equation 5.5. The value is between 0 and 1. Since the drop loses water over time through evaporation, it happens, that the capacity falls below the amount of currently carried sediment. For high values of $p_{deposition}$ that leads to visible sediment deposition on the flow path. This effect is visible in ravines on steep ground. A value close to 1 can result in spike artifacts in the ravines as seen in figure 5.4d. A value close to 0 leads to more ravine formation since nearly no sediment is dropped except for cases when a drop reaches a pit and tries to fill it up. The least ravine formation happens with a value around 0.1.

Erosion speed

Similar to the deposition speed, the parameter $p_{erosion}$ determines how much of the free capacity of a drop is filled with sediment in case of erosion as described in equation 5.6. The value is between 0 and 1. With a high erosion speed, a drop quickly fills its capacity and after that most likely only deposits sediment. With a low value, the drops pick up sediment for a longer path, which results in stronger ravine formation. If the value falls below 0,1 the ravine formation gets less again, since nearly nothing is eroded at all.



(a) initial terrain

(b) $p_{erosion} = 0.01$

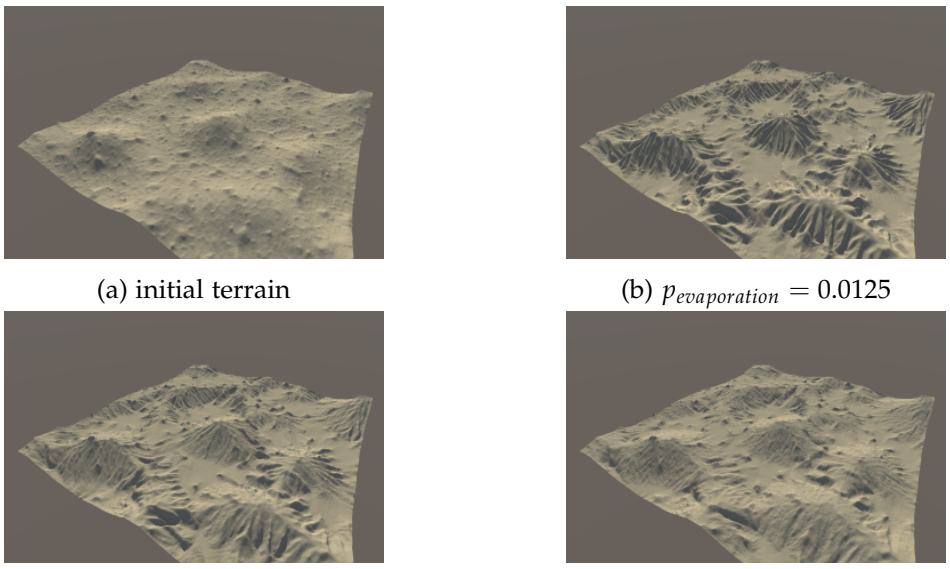
(c) $p_{erosion} = 0.1$

(d) $p_{erosion} = 0.9$

Figure 5.5: Example terrain eroded with different $p_{erosion}$ values. The terrain has a resolution of 256x256 samples. 75,000 drops were simulated in (b), (c) and (d).

Evaporation speed

The parameter $p_{evaporation}$ determines how fast the drops evaporate. Again its value is between 0 and 1. A faster evaporation leads to shorter paths of the drops in which they influence the terrain. That means that a slower evaporation increases the formation of



(a) initial terrain

(b) $p_{evaporation} = 0.0125$

(c) $p_{evaporation} = 0.05$

(d) $p_{evaporation} = 0.1$

Figure 5.6: Example terrain eroded with different $p_{evaporation}$ values. The terrain has a resolution of 512x512 samples. 300,000 drops were simulated in (b), (c) and (d).

ravines drastically. With values over 0.5 the water evaporates so fast, that nearly no changes are visible at all.

Erosion radius

The erosion radius p_{radius} determines the radius in which sediment is taken from the rock layer. The smaller p_{radius} is, the deeper and more distinct the ravines will be. Raising the erosion radius also increases the computational time needed for each drop drastically.

As seen in figure 5.7b values lower then 3 lead to unrealistically rugged terrain if the changes are applied unblurred, but interesting effects can be achieved, when the result is partially or fully blurred.

(a) initial terrain

(b) $p_{radius} = 2$

(c) $p_{radius} = 4$

(d) $p_{radius} = 6$

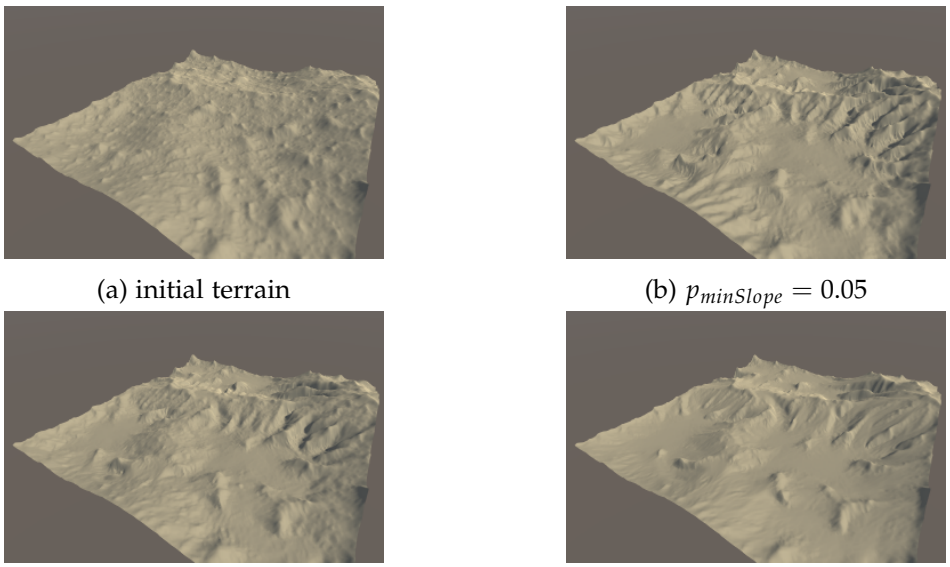
Figure 5.7: Example terrain eroded with different p_{radius} values. The terrain has a resolution of 256x256 samples. 75,000 drops were simulated in (b), (c) and (d).

Minimal slope

The minimal slope $p_{minSlope}$ is the minimum level of height difference that is taken for the calculation of the carry capacity of each single drop. Increasing the value ensures that the carry capacity does not fall below a certain line. Higher values lead to faster erosion but also stronger ravine forming. Terrains eroded with a low minimal slope need more drops but give smoother, more realistic looking terrains. Especially for detailed features and small scale maps a low $p_{minSlope}$ is recommended as seen in figure 5.8d.

Maximum path steps

The parameter $p_{maxPath}$ determines the maximum number of steps a drop is allowed to take before it is deleted. Obviously a drop has less impact on the terrain map if the maximum path steps are low. On the other hand the drop loses its ability to erode



(a) initial terrain

(b) $p_{minSlope} = 0.05$

(c) $p_{minSlope} = 0.01$

(d) $p_{minSlope} = 0.0001$

Figure 5.8: Example terrain eroded with different $p_{minSlope}$ values and different amounts of drops. The terrain has a resolution of 256x256 samples. 75,000 drops were simulated in (b) and (c); 225,000 drops were simulated in (d).

noteworthy amounts of sediment since evaporation decreases the amount of *water*, which means a sinking amount of the carry capacity c. So with increasing $p_{maxPath}$ the impact of a drop increases logarithmically.

Gravity

The gravity factor $p_{gravity}$ is used to determine the speed of motion of the drop. Increased $p_{gravity}$ leads to higher speed, which causes more carry capacity c. Overall a higher gravity factor leads to faster erosion, but there are no differences in the appearance of the terrain.

6 Evaluation

This chapter describes the evaluation of the developed algorithm in terms of performance and results.

6.1 Implementation

The algorithm was implemented in C# with the Unity3D Engine (version 5.0.0f4) [Tec], which also rendered all pictures in this work. I connected the droplet simulation with Unitys update function. This way only a previously determined amount of drops is simulated before rendering the terrain again. As a result it is possible to watch the algorithm proceed and stop at any point, when the terrain is eroded enough. Also various small functions like blurring the heightmap and saving the heightmap to a file are provided. This real-time interactivity helps to easily create visually appealing heightmaps.

6.2 Performance

In this section the performance of the algorithm is evaluated in means of time. The measurements were made on a system with an Intel Core i5-3470 CPU with 3.20 GHz and 8 GB RAM.

In table 6.1 the algorithm simulated different amounts of drops on four different sized heightmaps and measured the time. This process was repeated 10 times for every setting, the values in the table are the average times. The slightly higher times on big maps are due to the fact that a drop is far more likely to drop off a small map than from a large one. This is confirmed by the counts for drops falling off the map in 100,000 drops on sizes 256×256 ($\sim$16,500 drops), 512×512 ($\sim$12,000 drops) and 1024×1024 ($\sim$5,000 drops).

Also the first drops simulated always took longer than later ones. This is caused by ravines, that are formed over time. They provide a fast way for a drop to its destiny,

whether its outside of the map or a local minimum on the map to fill a pit and die. The first few drops have to form that raviens first.

terrain size drops	5,000	10,000	20,000	50,000	100,000
256×256	1.83s	2.27s	7.66s	19.63s	32.02s
512×512	2.08s	2.49s	7.79s	19.75s	35.89s
1024×1024	2.29s	2.56s	8.60s	20.90s	40.65s
2048×2048	2.38s	2.66s	9.33s	22.47s	43.81s

Table 6.1: Time in seconds needed for different amounts of drops on different sized heightmaps. The following parameters were used: $p_{inertia} = 0,3$, $p_{capacity} = 8$, $p_{deposition} = 0,2$, $p_{erosion} = 0,7$, $p_{evaporation} = 0,02$, $p_{minSlope} = 0,01$, $p_{gravity} = 10$, $p_{radius} = 4$, $p_{maxPath} = 64$;

Most parameters have no noticeable effect on the time, but p_{radius} extends the computational time per drop exponentially, since the drop interacts on every erosion step wit ${p_{radius}}^2$ grid points. Measured times can be seen in table 6.2.

p_{radius}	1	2	4	6	8
Time	0.48s	0.83s	2.27s	5.51s	9.69s

Table 6.2: Time in seconds needed for 10000 drops on a 256×256 sized heightmap. The following parameters were used: $p_{inertia} = 0,3$, $p_{capacity} = 8$, $p_{deposition} = 0,2$, $p_{erosion} = 0,7$, $p_{evaporation} = 0,02$, $p_{minSlope} = 0,01$, $p_{gravity} = 10$, $p_{maxPath} = 64$;

6.3 Results

As seen in figure 6.1 to 6.4 the eroded terrains look visually appealing and realistic. Through all the parameters described in 5.4 it is not only possible to adjust the algorithm to heightmaps with different characteristics and properties, but it is also possible to erode one terrain to different results as shown in figure 6.4.

In figure 6.1 a completely abstract looking terrain is eroded to a realistic looking, desert like landscape.

In figure 6.2 a detail view of natural features is given. Good recognizable in this figure is the combination of seemingly hard and rough rock and smooth sand. Natural features like water traces, a riverbed like structure and rocky cliffs can be seen.

While in figure 6.2 the scale is smaller - the terrain probably spans over $1km^2$ - the terrain in figure 6.3 is on a large scale basis. It shows a generated mountain ridge which spans over several kilometers in diameter.

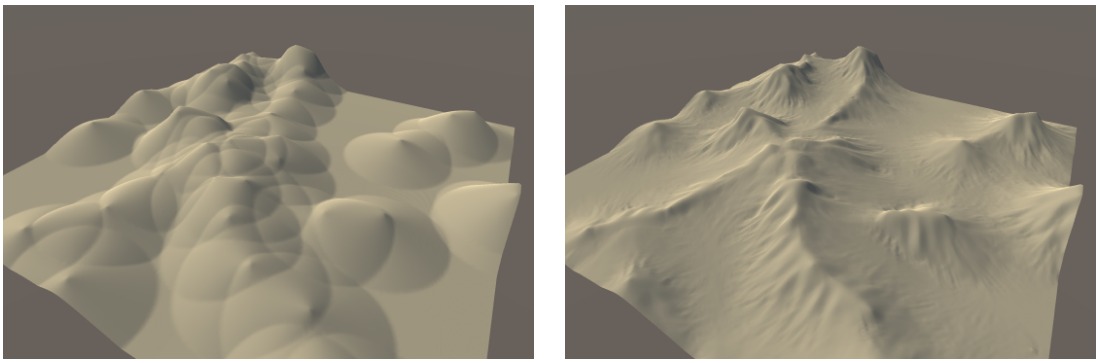


Figure 6.1: Completely abstract looking terrain (left) eroded to look natural and visually appealing. Resolution: 512×512;

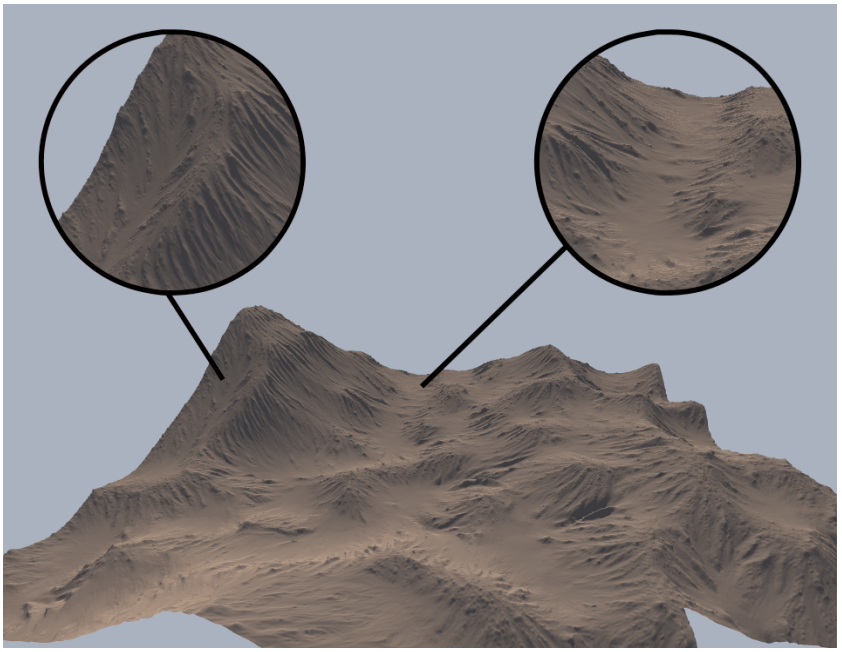


Figure 6.2: Detail view of natural features. Left circle: distinct traces of water; Right circle: a riverbed like structure; Resolution: 1024×1024;

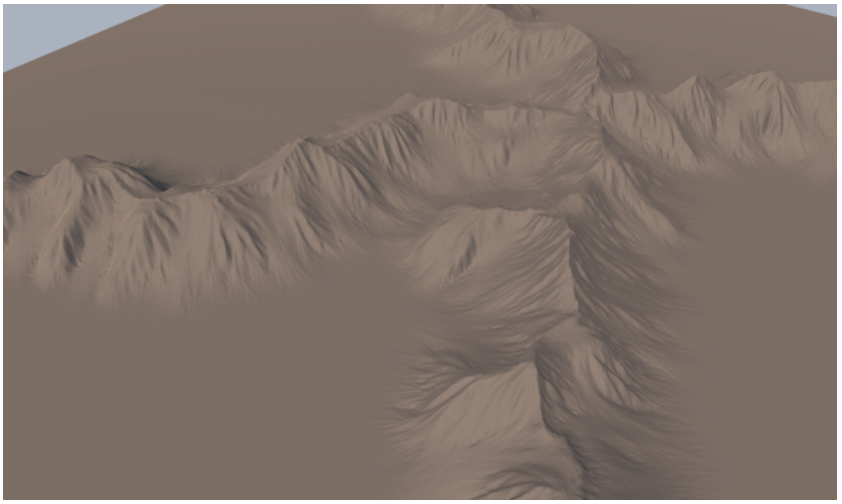


Figure 6.3: Generated Mountain Ridge. Resolution: 1024×1024;

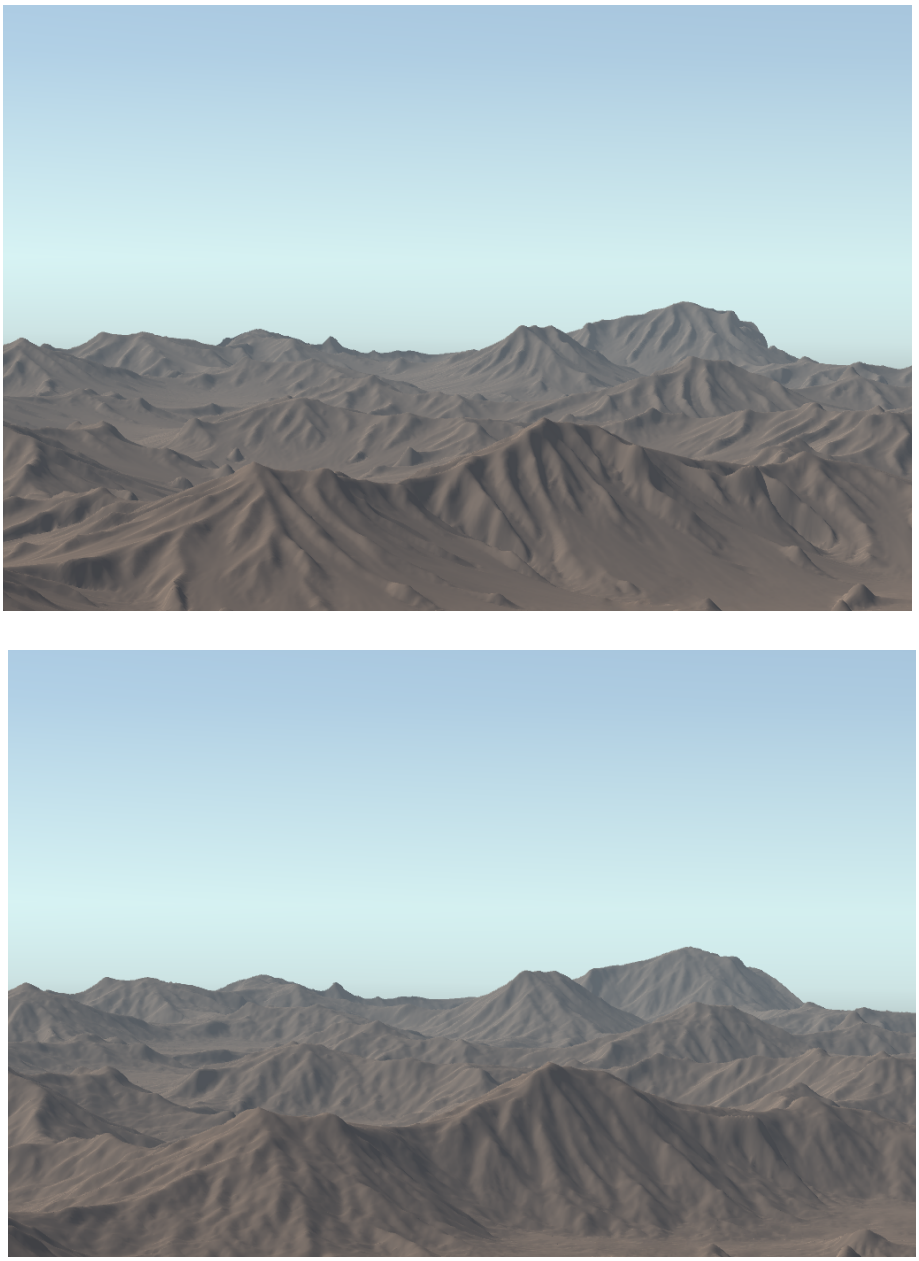


Figure 6.4: The same terrain eroded with different parameters. Resolution: 1024×1024;

7 Conclusion and Future Work

I have presented a simple particle based algorithm to simulate hydraulic erosion on a heightmap. The particles mimic water running on the terrain and distribut sediment. Through a wide range of adjustable parameters the algorithm is highly adaptable to different terrains.

As future work I propose porting the algorithm to C++, since C# doesn't allow pointers in a safe context and thus the algorithm often has to cache information instead of working on the object itself. Also parallelization of drops would increase the performance drastically. The drops could e.g. be each calculated in an own thread. It might even be possible to use the GPU to calculate the drop paths.

Another aspect in need of improvement is the drain valley effect as described in 5.4. A good approach could be to automatically simulate a few grid points around the map. Then whenever a drop is leaving the actual map it can still finish its iteration and then die.

Also I propose combining the algorithm with a thermal erosion algorithm as in [Ják11]. Thermal erosion describes the effect that small pieces of rock brake loose from the ground and form a layer of sediment on top of the rock. Then sediment slippage is simulated on the sediment layer. This means sediment only settles at a certain angle, called talus angle. If the gradient at any point is higher sediment is distributed from the high to the low grid points until the talus angle is reached.

Another improvement would be to implement more then just one layer of rock as described in [BF01]. Every layer gets different properties like how hard or soft the material is or, if thermal erosion is implemented, how high the talus angle is. This way more interesting natural phenomena could be simulated.

List of Figures

List of Tables

Bibliography

[Ben+06] B. Beneš, V. Těšınský, J. Hornyš, and S. K. Bhatia. *Hydraulic Erosion*. Computer Animation and Virtual Worlds 17(2), 99-108, 2006.

[Ben07] B. Beneš. *Real-Time Erosion Using Shallow Water Simulation*. The 4th Workshop on Virtual Reality Interactions and Physical Simulation - Vriphys'07, 43-50, 2007.

[BF01] B. Beneš and R. Forsbach. *Layered Data Representation for Visual Simulation of Terrain Erosion*. SCCG '01 Proceedings of the 17th Spring Conference on Computer Graphics Page 80, 2001.

[CMF99] N. Chiba, K. Muraoka, and K. Fujita. *An erosion model based on velocity fields for the visual simulation of mountain scenery*. The Journal of Visualization and Computer Animation, vol. 9, 185-194, 1999.

[FFC82] A. Fournier, D. Fussel, and L. Carpenter. *Computer Rendering of Stochastic Models*. Communications of the ACM, vol 25, 371–384, 1982.

[Ják11] B. Jákó. *Fast Hydraulic and Thermal Erosion on the GPU*. Proceedings of CESCG 2011: The 15th Central European Seminar on Computer Graphics, 2011.

[KMN88] A. D. Kelley, M. C. Malin, and G. M. Nielson. *Terrain Simulation Using a Model of Stream Erosion*. Computer Graphics, vol. 22, 263-268, 1988.

[Kri+09] P. Krištof, B. Beneš, J. Křivánek, and O. Šťava. *Hydraulic Erosion Using Smoothed Particle Hydrodynamics*. Computer Graphics Forum (Eurographics 2009) vol. 28 No.2, 219-228, 2009.

[LLC] P. S. LLC. *Terragen 3*. URL: `http://planetside.co.uk/` (visited on 05/11/2015).

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

[MCM89] F. Musgrave, K. C.E., and R. Mace. *The synthesis and rendering of eroded fractal terrain*. SIGGRAPH '89: Proceedings of the annual conference of computer graphics and interactive techniques, 41-55, 1989.

[MDB07] X. Mei, P. Decaudin, and H. B. *Fast Hydraulic Erosion Simulation and Visualization on GPU*. Computer Graphics and Applications, 2007. PG '07, 47-56, 2007.

[SKB15] V. Skorkovská, I. Kolingerová, and B. Beneš. *Hydraulic Erosion Modeling on a Triangular Mesh*. Surface Models for Geosciences Lecture Notes in Geoinformation and Cartography, 237-247, 2015.

[Šta+08] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek. *Interactive Terrain Modeling Using Hydraulic Erosion*. Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2008), 2008.

[Tec] U. Technologies. *Unity3D Engine*. URL: `http://unity3d.com/` (visited on 05/11/2015).

[Vol] A. Volynskov. *Water erosion on heightmap terrain*. URL: `http://ranmantaru.com/blog/2011/10/08/water-erosion-on-heightmap-terrain/` (visited on 05/11/2015).