Simulating Hydraulic Erosion Using Shallow Water Equations for Terrain Generation Adapted To Parallelisation On The GPU

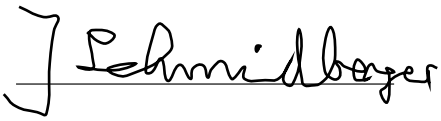
Bachelor Thesis in Computer Science

Jannik Schmidberger, March 2024

Supervisor:	Prof. Dr. Daniel Scherzer
	Ravensburg-Weingarten University of Applied Sciences
Co-supervisor:	Prof. Dr. Sebastian Mauser
	Ravensburg-Weingarten University of Applied Sciences

Declaration of Originality

Hereby I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and all used resources are indicated in the list of references.



Jannik Schmidberger, Bad Waldsee, Mar. 24th 2024

Abstract

Digital landscapes are needed in many software solutions like games, simulations and movies. These landscapes often are procedurally generated by computer algorithms or created by scanning real world terrain data. To improve such terrain data sets, natural phenomena can be simulated. A very common phenomena is hydraulic erosion. This work looks at a hydraulic erosion algorithm based on the virtual pipes model and shallow-water equations. The model is being reevaluated for application on modern computer systems, as it forms the basis for other more advanced models. The resulting terrain achieves visually more realistic terrains but has some artifacts and unwanted features. The model can achieve real-time execution for large terrain data sets and can be interacted with during the simulation through different parameters. This is being achieved by utilising parallelisation on current graphics hardware.

Contents

1 Introduction

Modern 3D software solutions like games, films and simulations can require large and complex terrain data to model believable digital worlds. This terrain data not only can cover multiple kilometers but also often needs to be highly detailed to be believable for the user. To generate these large amounts of data, procedural generation or real world scanned terrain data is often used. These methods can easily provide large data sets but can lack the needed detail [GGP+19]. Therefore it can be required to improve these data sets with manual editing and detailing. While the results of this approach are more likely to be of high quality, but this can require many man hours of work to produce.
A possible solution is to simulate naturally occurring effects of the real world terrain creation to produce authentic 3D terrain data. One highly influential effect in the real world is hydraulic erosion [RSH01]. This approach has been a topic of research for more then 35 years [MKM89] with further improvements and new approaches being published continuously. Other effects entail thermal and wind erosion, often paired with hydraulic erosion. Vegetation effects and tectonic models can also be utilised for terrain generation.

Definition of the task

The goal is to improve the detail of an already existing terrain data set in form of a heightmap. This work is based on the "Fast Hydraulic Erosion Simulation and Visualization on GPU" [MDH07] hydraulic erosion algorithm to evaluate its possible use for generating authentic terrain data. Furthermore the simulation should be controllable and interactive in real time to improve authoring terrain data results to specific needs and requirements. The resulting terrain should be visually appealing and follow natural occurring terrain features to be believable. For this goal the fluid and erosion simulations do not need to be physically correct.

Outline of the thesis

The first chapter introduces the topic and goal of this thesis. The second chapter defines and specifies required terms. The third chapter gives a broad overview of the history in the field of hydraulic erosion in Computer Graphics until recent publications. Chapter four describes the hydraulic erosion algorithm. In chapter five some additional implementation details are shown for the specific implementation. Chapter six includes the results gathered in this thesis. The final chapter, chapter seven, provides the conclusion of this thesis as well as an outlook for future works.

2 Terms and Definitions

All used terms needed for this thesis are defined in this chapter.

Heightmap

A heightmap is a two dimensional grid based data structure. A grid cell can be access via x and y indices. Each grid cell can hold a data value representing a terrain height value for that cell as a float value. All height values are normalized between [0, 1].

Flux

To better differentiate between the water flow $\vec{f}$ and the water flow velocity $\vec{v}$ the water flow $\vec{f}$ is called flux.

Mesh

A mesh is a data structure for mostly 3D objects. A mesh consists of vertices, coordinates in 3D space, which are connected by edges to encompass polygons. The objects surface then can be represented by these polygons.

Voxel

Voxel describe a point inside a three dimensional data grid, also called voxel grid. Similar to meshes, voxel grids can be used to represent 3D objects. A single voxel does not store any coordinates unlike mesh vertices. Voxels instead store data on how to behave when rendered like being invisible, having a color or texture. Voxels therefore are similar to 2D pixels.

GPGPU

GPGPU (General Purpose Computation on Graphics Processing Unit) is a low level programming interface for high level programming on the GPU.

Shader

Shaders similar to GPGPU but optimised for visual rendering of pixels. Shaders provide a high level programming interface for programs on the GPU.

Compute Shader

Compute Shaders are a practical framework for a GPGPU program similar to other frameworks like CUDA. Compute Shaders are hardware independent unlike CUDA.

3 Related Work

The field of hydraulic erosion simulation in Computer Graphics has been a topic of research for over 35 years [MKM89]. In this process water flow is simulated over a terrain and dislodges sediments from the terrain surface. One of the first works in this field being [KMN88], creating river and stream networks and then generating fractal terrain around them.
There are two fundamentally different approaches in hydraulic erosion simulation models, namely in the way the hydraulic model is described. There are eulerian approaches which require a discrete grid to be able to simulate the hydraulic flow between cells. Every cell tracks the contained fluid behaviour as in the incoming and outgoing fluid to and from the neighbouring cells. Alternatively there are lagrangian approaches which describes the hydraulic model with fluid particles. Each particle tracks its own position and velocity and can interact with other surrounding particles. A comparison is shown in Figure 1 by [GGP+19]. In general, lagrangian solutions are more expensive to scale then shallow-water equations solutions due to the water depth [Kel17]. There are lagrangian solutions which ignore water depth to increase scalability shown in Section 3.
This work is based on the eulerian approach using simplified Navier-Stokes equations in form of a virtual pipe model as the underlying hydraulic model.

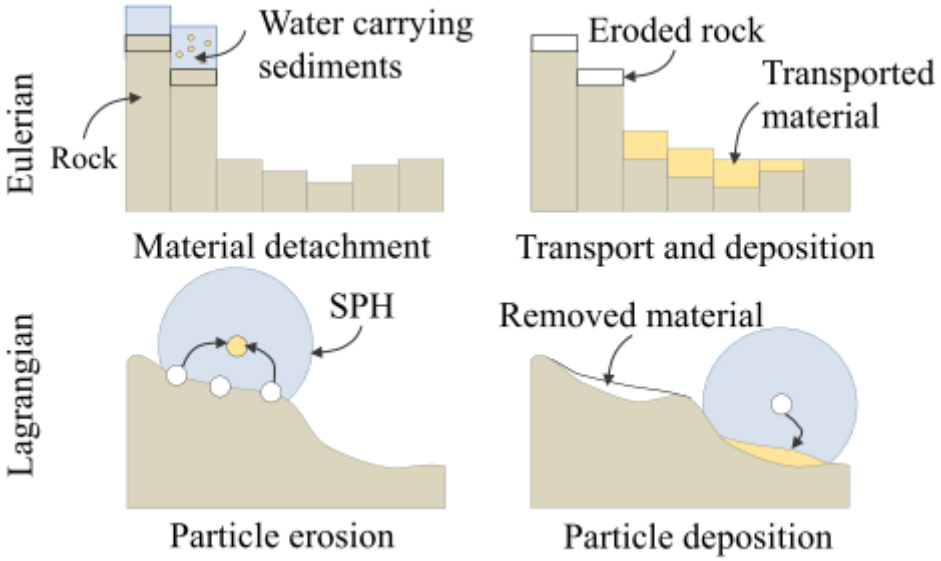


Figure 1: Comparison of Eulerian and Lagrangian approaches to hydraulic erosion [GGP+19]

Navier-Stokes Equations

The Navier-Stokes equations [CF88] form the basis for every physics based fluid simulation [CMM90] but the amount of adherents to these equations can differ between simulations. The most accurate results are achieved by using the complete Navier-Stokes equations at the cost of high computationally complex derivatives. For simulations with a focus on a correct fluid behaviour this might be an acceptable trade-off but for simulations attempting real-time execution and interactivity this is not feasible.
There are many simplifications of the Navier-Stokes equations reducing computational complexity at the cost of reduced simulation accuracy. The most common simplifications are the shallow-water equations introduced to Computer Graphics by [Cas90].

Shallow-Water Equations

For shallow-water equations, the depth dimension of the fluid is ignored, therefore only a fluid surface is considered. This simplification is most applicable when the horizontal scale is much bigger than the vertical scale, making the vertical impact for the simulation at large negligible. Examples for this include rivers, coastlines, oceans and lakes. There are also layered shallow-water equations models to introduce some kind of fluid depth into the simulation while retaining the reduced computation costs or model multiple fluids as seen in [AK09].
A sub type of shallow-water equations are the Saint-Venant equations [Pon90]. Here the flow direction is unidirectional instead of bidirectional.
There are namely two kinds of shallow-water equations, friction shallow-water equations and no-slip shallow-water equations, as described by [Bre09]. These differ mainly in handling the simulation boundary condition. No-slip systems are considered closed systems with no possible exchange with outside effects. Friction systems are considered open systems where fluid can exit the simulation domain, under which methodology is model specific.

Virtual Pipes Model

O'Brien et al. [OH95] proposed a virtual pipes model for simulating shallow water on grid based systems. The virtual pipes model simulates water flow by connecting grid cells with their immediate neighbours through virtual pipes and moving water based on the hydro-static pressure differences between connected cells. This model was developed without parallelisation in mind, which limited performance and scalability. Mei et al. [MDH07] solved this problem by parallelisation and a GPGPU implementation. They also adapted this virtual pipes model to include hydraulic erosion. Figure 3 shows the virtual pipe model by [MDH07].
Besides early performance drawbacks the virtual pipes model can not represent water depth. This problem has been solved by [BMPG11] by layering multiple virtual pipes models. Each layer first is simulated like already established virtual pipes models. Then each layer can interact with the layer immediately above or below. This allows for using virtual pipes models in 3D simulation domains.

Later a multi-layered data structure for the virtual pipes model was introduced by [Kel13] which allowed the virtual pipes model to interact dynamically with objects other than the underlying terrain. This was later improved and expanded upon by [Kel15] to be a generalized model. This multi-layered model was further improved by [DGV+18] to include a viscosity factor to simulate fluids other than water. This model also can handle more complex terrain features like caves and overhangs.

Eulerian Approach

The first erosion simulation on a heightmap grid has been introduced by [MKM89]. This only represents a 2D terrain simulation. The algorithm later was extended to model sand and mud movements by [SOH99] and [ON00].
The first physically-inspired hydraulic erosion model was introduced by [Nag98]. This erosion model is focused on the generation of mountain valleys and ridges and is guided by fluid flow forces and their interaction with the terrain layer. Later [NWD05] adapted this to be run in real time and extended it to use the sediment capacity introduced by [MKM89]. This then was later adapted for parallelisation by [ASA07].
Another development in modeling hydraulic erosion was a new data representation through layering heightmaps presented by [BF01]. Here multiple heightmap layers were stacked on top of one another with each layer representing different geological materials. Each layered material can differ in their attributes like hardness. A geological representation of such material differences was shown in [RPP93]. Therefore the erosion process effected each layer differently resulting in more realistic terrain approximations. Compared to the more accurate 3D voxel grid representations, this approach is less computationally expensive and requires less memory which results in better scalability. This layered approach then was used for hydraulic erosion simulation using shallow-water equations by [Ben07] in 2007. Compared to a volumetric approach, as given below, the layered data structure can not represent material transitions and material mixing. To improve this representation [SK16] introduced a new data representation based on binary space partitioning.
For a more accurate fluid simulation [BTHB06] combined hydraulic erosion with a fully 3D fluid simulation. Here a focus was put on a visually appealing fluid simulation for uses in animation as well as the erosion process. Instead of 2D heightmaps this approach used a voxel grid with each cell storing terrain and material data. Compared to solutions mentioned above this approach has the advantage of true three dimensinality which allows for terrain features like cliff overhangs and cave systems. In return the computational cost is greatly increased, making a real-time simulation unfeasible. Later [TJ10] introduced a new terrain representation based on the Pons and Boissonnat's Delaunay deformable model [PB07] to retain the full 3D terrain representation of voxel grids while also improving the performance and scalability of the simulation by an order of magnitude.

The first physically based hydraulic simulation using GPGPU which facilitates greater performance gains through parallelisation was introduced in 2007 by [MDH07]. This was based on the previous work by [NWD05] and [BF02]. This approach later was extended to include the layered heightmaps representation by [ŠBBK08]. To improve memory restrictions [VBHS11] introduced a virtual layered terrain representation based on the layered terrain representation mentioned before. Here the terrain layers are dynamically subdivided into rectangular tiles. Currently unused tiles are then swapped from the graphics memory to the main computer memory.
The mentioned erosion models can be combined with other natural phenomena for terrain generation. [CCB+18] combined an erosion model with a simulation of the earth crust and tectonic simulation. Thanks to GPGPU implementations and a layered data structure the simulation can run in real time and is interactive. A more general approach was demonstrated by [CGG+17] combining multiple natural phenomena such as rock falls, lightning strikes and vegetation life cycles with hydraulic erosion. To reduce complexity of the simulation and achieve interactive rates, these phenomena are executed in random order and can only interact with the layered terrain representation, not with each other. Vegetation is also represented as a layer in the layered terrain representation. This was further expanded upon by including Vegetation Cover, called Cover Management Factor (C-Factor), in the hydraulic erosion simulation by [HR23]. This was achieved by extending the general erosion model by [MDH07] to include a dead vegetation and alive vegetation layer in a layered terrain representation. This introduced additional simulation steps related to vegetation. Though there are not multiple material layers.

Lagrangian Approach

Contrary to eulerian approaches, lagrangian approaches can be used with arbitrary terrain data structures. This is achieved by simulating fluids with particles utilizing hydrodynamics and gravitational forces. For erosion simulations, particles often have only limited interactivity with neighbouring particles and similar to shallow-water equations, fluid depth is ignored to reduce computational complexity.
This approach can be used for simulating hydraulic erosion by rainfall particularly efficiently. Each raindrop can be represented by a water particle hitting the terrain and moving downhill, simulating erosion processes on the way. This concept was introduced by [CMF98]. The erosion process was based on a velocity field calculated by the particle motion on the terrain. An eulerian erosion model combined with Smoothed Particle Hydrodynamics (SPH) was introduced by [KBKŠ09]. For this work the terrain is represented as an uniform heightmap with the given advantages and disadvantages mentioned earlier.
Later [SKB15] combined a SPH approach with an arbitrary triangle mesh data structure for terrains. This approach allows for complex terrain features like caves and overhangs similar to the volumetric approach by [BTHB06] but requires less memory.

4 Hydraulic Erosion Model

As stated by [MDH07] the simulation model is segmented into five steps. Each step is depending on its predecessor. Some steps are divided into smaller sub steps which in turn also are depending on the correct execution order. Therefore these sub steps are treated as normal steps as well. This segmentation is needed as part of the parallelization adaptation. It is assumed that each calculation for a single cell is done in parallel at the same time for all other cells. If data of neighbouring cells is required for a step, this data has to be computed before being accessed. Therefore this segmentation has been chosen.
The following simulation model is based on the pipe model introduced by [OH95] and [MY97], an explicit realisation of the shallow-water equations. In turn the shallow-water equations are a simplification of the Navier-Stokes equations which provide more accurate results but require far more computationally expensive derivatives. For this real-time simulation the less accurate shallow-water equations are sufficient given the stated goal of this work.
As with many computer models, some input parameters can not be represented correctly and need to be approximated. Approximations can introduce additional inaccuracies and potential instabilities into a model. For this model, the time is represented in discrete time steps Δt instead of being continuous. For very small time steps, the approximation yields satisfactory results. For larger time steps, the model can become numerically unstable. Biswas et al. [BCMP13] have published more details on the topic.
As stated by [MDH07] the numerical stability of the simulation is based on the time step Δt. The maximum for this time step is restricted among others by the Reynolds number, boundary conditions, cell size and the flux scaling factor k. This scaling factor k is computed for each cell but to avoid numerical errors in the flow rate, the scaling factor of neighbouring cells should be accommodated for as well. This will not be considered in the model.
Following the individual steps per simulation cycle are listed. Each step is described in more detail in the following sections. The following step separation has been chosen based on the mathematical formulas.

The seven steps are:

1. Increasing water height due to water sources and rain
2. Shallow-Water Model: Updating outflow
3. Shallow-Water Model: Updating water height based on flux
4. Shallow-Water Model: Updating velocity field
5. Erosion and deposition process
6. Transporting sediment based on the velocity
7. Water evaporation

Increasing Water Height

Water can increase among others by either placed water sources w_n, by rain drops or both combined. Each water source w_i has a position (x, y), the strength r_t on how much water to add each iteration and a radius. Rain is handled similar to a water source but instead of a single position, water is dropped with a random distribution on the terrain. Each rain droplet has a strength and radius. For cell (x, y) the water height is increased by adding the sum of all water sources influencing this cell as shown in Figure 2. For this step eqn. (1) is calculating d_1 using simple summation.

For cell (x, y) d_1 is calculated as:

$$d_1(x, y) = d_t(x, y) + \Delta t \cdot \sum_{i=1}^{w_n} r_{it}(x, y) \tag{1}$$

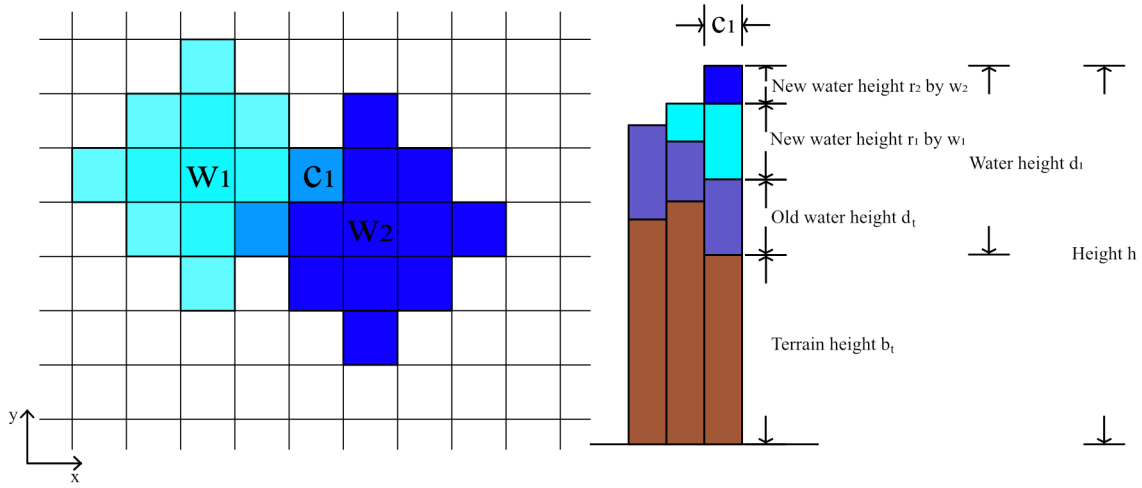


Figure 2: Data grid showing information for cell c_1

Updating Outflow Flux

For the pipe model by [OH95], adapted by [MDH07], each cell is connected via virtual pipes to its neighbouring cells as shown in Figure 3 by [MDH07]. All given equations only consider the four von Neumann neighbours. As stated by [MDH07], these four neighbours produce satisfactory results. Though all equations can be adapted to consider more neighbours like the eight immediate neighbours of a given cell.
Due to hydro static pressure differences water is then exchanged between neighbouring cells. For the simulation only the outflow flux $\vec{f} = (f^L, f^R, f^T, f^B)$ is needed to be stored, inflow flux can be acquired by taking the neighbouring outflow flux. f^L here refers to the outflow flux from cell (x, y) to its left neighbour cell $(x - 1, y)$ and so on.

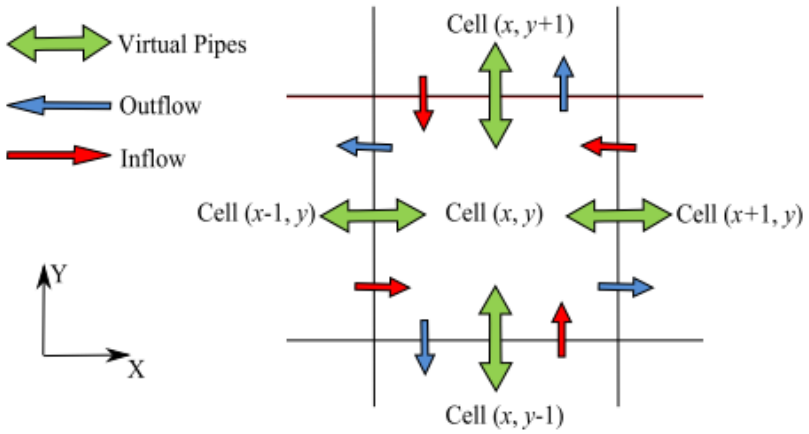


Figure 3: Pipe model notations by [MDH07]. Each cell is connected to its four neighbours through virtual pipes.

This model has been optimized by [MDH07] for parallelisation on the GPU. First the water outflow flux of each cell is calculated. Normally each cell collects water due to inflow flux at the same time but this can lead to race conditions on the GPU. To avoid the possible problem of removing more water from a cell then it has and ending up with a negative water amount, the outflow flux is limited by the water amount of this cell. This is achieved via limiting $\vec{f}$ by the scaling factor k in eqn. (5).

For cell (x, y) f^L is calculated as:

$$f^L_{t+\Delta t}(x,y) = max(0, f^L_t(x,y) + \Delta t \cdot A \cdot \frac{g \cdot \Delta h^L(x,y)}{l}) \tag{2}$$

A more detailed depiction of the derivation for eqn. (2) can be found in O'Brien's original paper [OH95].

For cell (x, y) k is calculated as:

$$k = min(1, \frac{d_1 \cdot l}{(f^L_{t+\Delta t} + f^R_{t+\Delta t} + f^T_{t+\Delta t} + f^B_{t+\Delta t}) \cdot \Delta t}) \tag{3}$$

The scaling factor k in eqn. (3) can guarantee that water is never negative according to Mei et al. [MDH07].

Here $\Delta h^L(x, y)$ refers to the height difference of cell (x, y) and the neighbouring cell $(x-1, y)$. Height h is shown in Figure 2. It is calculated as:

$$\Delta h^L(x, y) = b_t(x, y) + d_1(x, y) - b_t(x - 1, y) - d_1(x - 1, y) \tag{4}$$

Now with eqn. (2) and (3) the final outflow flux $\vec{f}$ for time step Δt can be calculated as:

$$f^i_{t+\Delta t}(x, y) = k \cdot f^i_{t+\Delta t}(x, y), \quad i = L, R, T, B \tag{5}$$

Updating Water Height Based On Flux

To update the water height first the inflow flux f_{in} for cell (x, y) from neighbouring cells is collected in eqn. (6). Then the outflow flux f_{out} from cell (x, y) to the neighbouring cells is summed in eqn. (7). The difference of inflow and outflow is then computed and added to the water delta d_1 in eqn. (8). The result is called d_2.
It is assumed that water can not flow outside the grid. Therefore for boundary cells like $(0, y)$ the inflow and outflow to neighbouring cells not present on the grid is set to 0.

For cell (x, y) f_{in} is calculated as:

$$f_{in} = f^L_{t+\Delta t}(x + 1, y) + f^R_{t+\Delta t}(x - 1, y) + f^T_{t+\Delta t}(x, y - 1) + f^B_{t+\Delta t}(x, y + 1) \tag{6}$$

For cell (x, y) f_{out} is calculated as:

$$f_{out} = \sum_{i=L,R,T,B} f^i_{t+\Delta t}(x, y) \tag{7}$$

For cell (x, y) d_2 is calculated as:

$$d_2(x, y) = d_1(x, y) + \frac{\Delta t \cdot (f_{in} - f_{out})}{l} \tag{8}$$

Updating Velocity Field

The flow velocity $\vec{v}$ is later used to calculate how much sediment is eroded or deposited in the erosion step. The flow velocity can be calculated by the amount of water flowing between cells during a time step Δt. As this model is based on the shallow-water equations only the horizontal velocity $\vec{v} = (u, v)$ is considered. The vertical velocity is comparatively small and can be ignored for this model. The flow velocity approximation in eqn. (9) and (10) yields satisfactory results according to Mei et al. [MDH07].

For cell (x, y) velocity u in X direction is calculated as:

$$u = \frac{f^R(x-1, y) - f^L(x, y) + f^R(x, y) - f^L(x+1, y)}{2} \tag{9}$$

For cell (x, y) velocity v in Y direction is calculated as:

$$v = \frac{f^B(x, y+1) - f^T(x, y) + f^B(x, y) - f^T(x, y-1)}{2} \tag{10}$$

Erosion And Deposition Process

Erosion occurs when water flows over terrain. The eroded sediment is then transported alongside the flow and later deposited back onto the terrain. This process is mainly driven by the sediment transport capacity C of the flowing water [HD01]. The transport capacity is depending on the flow velocity $\vec{v}$ and the terrain tilt angle α. There have been proposed multiple prediction models for the transport capacity, here the model of [JS85] has been used and simplified by [MDH07].

For cell (x, y) the sediment transport capacity C is calculated as:

$$C(x, y) = K_c \cdot \sin(\alpha(x, y)) \cdot |\vec{v}(x, y)| \tag{11}$$

For cell x, y the terrain tilt angle α is calculated as using the normal vector $\vec{n}(x, y)$:

$$\alpha = \max(\alpha_{min}, \arccos(\vec{n}(x, y))) \tag{12}$$

As seen in eqn. (11) all factors are multiplicative. Therefore when the terrain tilt angle α approaches 0, the transport capacity C is approaching 0 as well resulting in a negligible erosion effect. Limiting the terrain tilt angle α to a user specified minimal threshold angle α_{min} solves this problem.
For the erosion-deposition process the transport capacity C is compared to the suspended sediment s for cell (x, y). If the capacity is greater then the suspended sediment $C > s_t$ then additional sediment can be absorbed, therefore erosion is happening shown in eqn. (13). Otherwise sediment is deposited back onto the terrain shown in eqn. (14) and Figure 4.

For the erosion process the new terrain height $b_{t+\Delta t}$ and suspended sediment s_1 for cell (x, y) are calculated as:

$$\begin{aligned} s_1 &= s_t + \min(C - s_t, K_s \cdot \Delta t) \\ b_{t+\Delta t} &= b_t - \min(C - s_t, K_s \cdot \Delta t) \end{aligned} \tag{13}$$

For the deposition process the new terrain height $b_{t+\Delta t}$ and suspended sediment s_1 for cell (x, y) are calculated as:

$$\begin{aligned} s_1 &= s_t - \min(s_t - C, K_d \cdot \Delta t) \\ b_{t+\Delta t} &= b_t + \min(s_t - C, K_d \cdot \Delta t) \end{aligned} \tag{14}$$

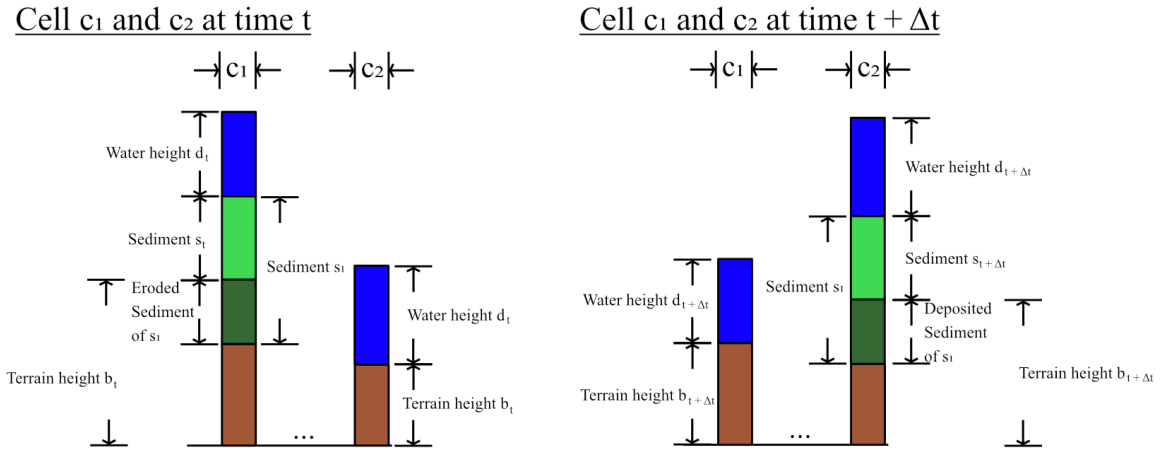


Figure 4: Shown are two cells c_1 and c_2. At time t sediment is added to s_t in c_1 resulting in s_1. Then s_1 is transported to c_2 during time step Δt. In c_2 some sediment is then deposited resulting in the new sediment and terrain values $s_{t+\Delta t}$ and $b_{t+\Delta t}$

Transporting Sediment

After eroding the terrain and adding suspended sediment into the water, the sediment is moved following the water flow as seen in Figure 4. To simulate this, the water flow velocity $\vec{v}$ is used to calculate the new position where the sediment is moved to during a time step Δt. The advection equation can solve this process shown in eqn. (15). The advaction equation can be solved by using the semi-Lagrangian advection method by Stam [Sta99].

$$\frac{\partial s}{\partial t} + (\vec{v} \cdot \nabla) = 0 \tag{15}$$

The new cell position for cell (x, y) is calculated by taking an Euler step backward in time with the flow velocity $\vec{v}$:

$$\begin{aligned} x_1 &= x - u \cdot \Delta t \\ y_1 &= y - v \cdot \Delta t \\ s_{t+\Delta t}(x, y) &= s_1(x_1, y_1) \end{aligned} \tag{16}$$

As this model is working on a discrete grid, the resulting new position should be located at a discrete grid position to prevent inaccuracies. There are multiple ways to achieve this, here a simple linear interpolation with the four nearest grid position has been chosen as shown in Figure 5.

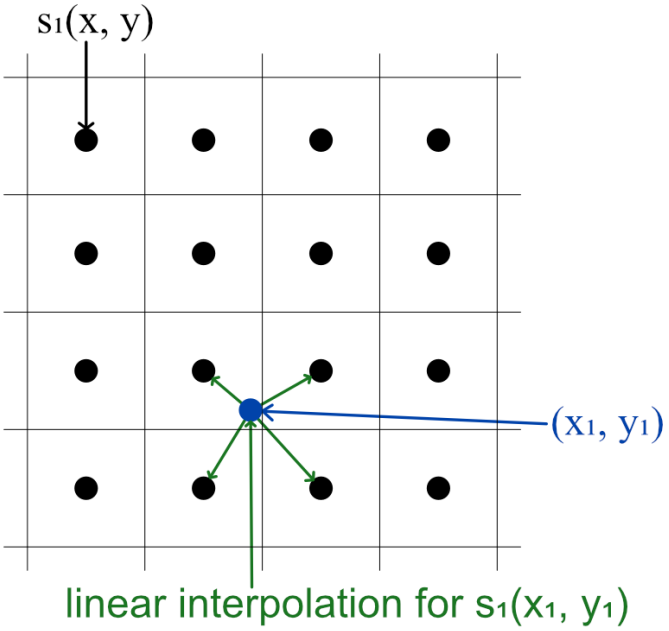


Figure 5: When (x_1, y_1) do not correspond to a discrete grid position then the sediment value s_1 is interpolated with the four neighbouring cells.

As this step is taking data from other cells there can be race condition problems when executed in parallel. The suspended sediment amount after this step can differ from the actual eroded sediment amount, generally being lower then it should be. Therefore this is not a physically correct method but for the stated goals this inaccuracy is acceptable.

Water Evaporation

As the final step in this model, water is evaporated. This step has been simplified from physical evaporation models. This can be done because the models main goal is the erosion and deposition process, not a realistic water simulation.

For cell (x, y) the final water height $d_{t+\Delta t}$ is calculated as:

$$d_{t+\Delta t}(x, y) = d_2(x, y) \cdot (1 - K_e \cdot \Delta t) \tag{17}$$

As the evaporation is calculated multiplicative, the water height $d_{t+\Delta t}$ can never reach 0. Therefore a threshold should be defined to set $d_{t+\Delta t}(x, y) = 0$ if $d_{t+\Delta t}(x, y) < 0.0001$.

Simulation Parameter

For this simulation a total of twelve different two dimensional data grids shown in Table 1 have been used. Each data point (x, y) corresponds to a position inside the grid.
There are three different types of data:

- Input data at time t - either the initial data set or the result of the last simulation iteration
- Intermediate data at time Δt - data generated, used or modified during a simulation iteration
- Output data at time $t + \Delta t$ - data output after a simulation iteration

Symbol	Description
b	Terrain height
s	Suspended sediment
d	Water height
f^L	Outflow flux to left neighbour
f^R	Outflow flux to right neighbour
f^T	Outflow flux to top neighbour
f^B	Outflow flux to bottom neighbour
u	Flow velocity in x direction
v	Flow velocity in y direction
d_1	Water height delta 1
d_2	Water height delta 2
s_1	Suspended sediment delta 1

Table 1: Required 2D data grids

Additionally there have been used nine float parameters to control the simulation, shown in Table 2.

Symbol	Description	Value
Δt	Simulation time step	0.002
A	Pipe cross-sectional area	50
l	Length of the virtual pipe	1
g	Gravity	10
α_{min}	Minimal terrain tilt angle	30°
K_c	Sediment transport capacity constant	0.0004
K_s	Sediment dissolving constant	0.00003
K_d	Sediment deposition constant	0.00003
K_e	Water evaporation constant	3

Table 2: Required parameters

5 Implementation

To achieve the set goal of a real time simulation, the simulation runs on the GPU taking advantage of parallelization. In visual computing, generally shaders are used due to their ease of use and design for parallelization. As shaders are primarily designed for visual rendering, there is a lot of optimisation and overhead in form of the rendering pipeline. To avoid the unnecessary overhead, compute shaders can be used. These are special shaders outside the render pipeline, designed for computing large, organised data. Alternatively the CUDA framework by NVIDEA or the open source frameworks OpenCL and OpenACC can be used. All of these general purpose compute frameworks are based on GPGPU (General Purpose Computation on Graphics Processing Unit). As oposed to the very common CUDA framework, compute shaders are hardware independent.
Since this simulation is grid based and has been optimised for parallel execution, shaders in general and compute shaders specifically match the requirements perfectly.

Each step described in Section 4 is being implemented as a new shader kernel. In order to be executed in the correct order. Each kernel is given read and write access to the required data in the GPU memory. Dynamic data that can be modified during the simulation has to be passed to the GPU every time it has to be updated. Therefore this type if data should be kept to a minimum. Here mainly the water sources can be modified, added or removed during the simulation. This way there is some agency in modifying the final terrain.

Handling Data

To simplify and logically group the twelve data grids mentioned in Section 4, four 32-Bit ARGB textures T_1, T_2, T_3, T_4 have been used. If memory is a concern, it is possible to use only three textures at the cost of a less readable implementation. Since the simulation takes place in a compute shader, it is possible to use other data structures like arrays. Here textures have been used because the heightmap input data already is a texture and the resulting output texture T_1 can be further used for rendering the terrain with a surface shader. Textures can be quite useful here for debugging as well since it is very easy to render textures on the screen with shaders. Getting data from the GPU for debugging can be more difficult otherwise.

The following data storage layout has been used:

- The initial input data are the starting terrain height values b in form of a heightmap. These are stored in the texture T_1 in the red channel. Suspended sediment s is stored in the green channel and the water height d is stored in the blue channel.
- Texture T_2 stores the water outflow, here called outflow flux from now on, for each grid position to its four neighbouring cells $\vec{f} = (f^L, f^R, f^T, f^B)$.
- Texture T_3 stores the water velocity field $\vec{v} = (u, v)$.
- Texture T_4 stores intermediate data. There are two water deltas d_1 and d_2 stored in the red and green channels. The sediment delta s_1 is stored in the blue channel.

Visualization

The simulation can be run without any visual representation. But for evaluation and potential authoring the terrain and erosion model have to be represented in a visual method. For that a mesh plane is generated with a resolution corresponding to the heightmap resolution. A surface shader then takes the output texture T_1 as input for rendering. The mesh plane is then displaced by the red and blue channels of T_1. For positions where the blue channel is displacing the mesh, the mesh is rendered with a water texture. Everything else is rendered using terrain textures.

6 Experimental Results

For evaluation and comparability a similar system is used as in [MDH07] and [HR23]. The simulation is being run on muliple grid resolutions multiple times to gain an averaged execution time. The measured variance was negligible. Furthermore visual results are taken and evaluated for expected erosion effects.
The given results in Table 3 were achieved using DirectX 11 on a NVIDEA RTX 2070 Super and AMD Ryzen 7 5800X. On the given hardware the simulation is achieving real-time execution in all resolutions until 4096x4096.

Resolution and Time Step	**Simulation Time**	**Visualization Time**	**Combined Time**	**Cycles per Second**
256x256 ($\Delta t = 0.5$)	0.066	1.705	1.771	564
512x512 ($\Delta t = 1$)	0.299	1.732	2.031	492
1024x1024 ($\Delta t = 2$)	1.113	4.007	5.12	195
2048x2048 ($\Delta t = 4$)	4.411	10.462	14.873	67
4096x4096 ($\Delta t = 8$)	19.512	43.779	63.191	15

Table 3: Experimental performance results. All timed values represent a single simulation cycle and are given in milliseconds

In Figure 6 an initial heightmap based on Voronoi-Noise is shown before the simulation is started as a reference point. After 10,000 iterations the result can be seen in Figure 7. Here the effects of the river source can be seen with clear erosion canals moving down the mountain. Also some cavities can be seen between each mountain. The transition between eroded terrain and uneroded terrain is relatively steep. This could be improved by adding thermal erosion effects. At 10,000 iterations the rain erosion effects are barley visible at the used settings depicted in Table 2.
At 20,000 iterations the resulting erosion of the river source becomes very prevalent while the rain erosion mostly manifests in erosion between mountains and at the bottom of mountains shown in Figure 8. With 30,000 iterations shown in Figure 9 the first rain erosion effects can be made out properly. The more flat ground has been eroded by droplets. Additionally the first rain droplet paths down the mountains can be observed.
In Figure 10 the result of 150,000 iterations can be seen. For this test run, the river source was deactivated and only the rain source was used. Here the erosion paths downhill the mountains can be clearly observed with a expansive river network spanning the lower ground where rain water gathered during the simulation.
Figure 11 depicts the result of 5,000 iterations using only a river source on a more detailed initial heightmap.

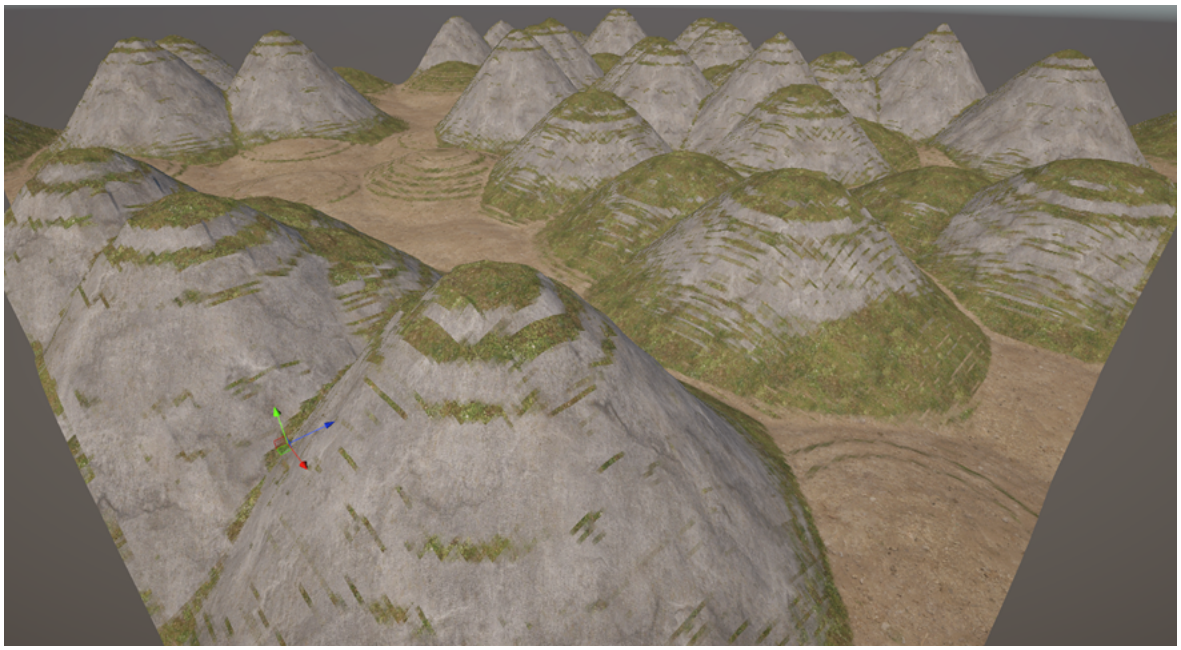


Figure 6: Initial heightmap based on Voronoi-Noise at 0 simulation iterations.

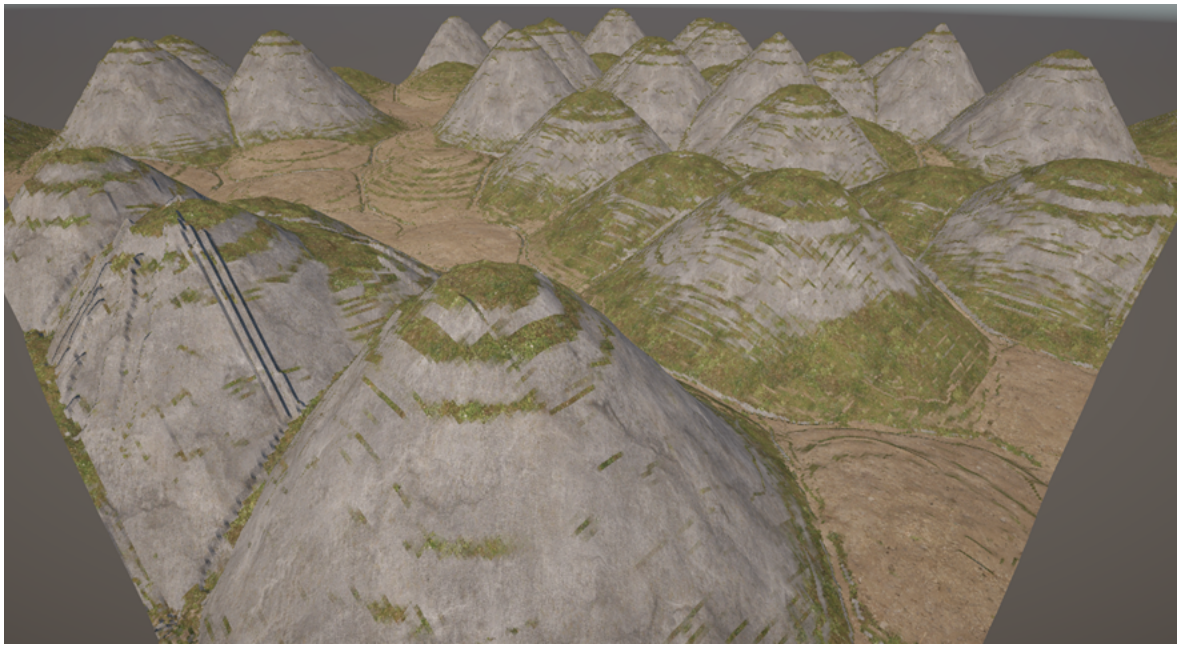


Figure 7: Heightmap based on Voronoi-Noise at 10,000 simulation iterations with rain and river sources.

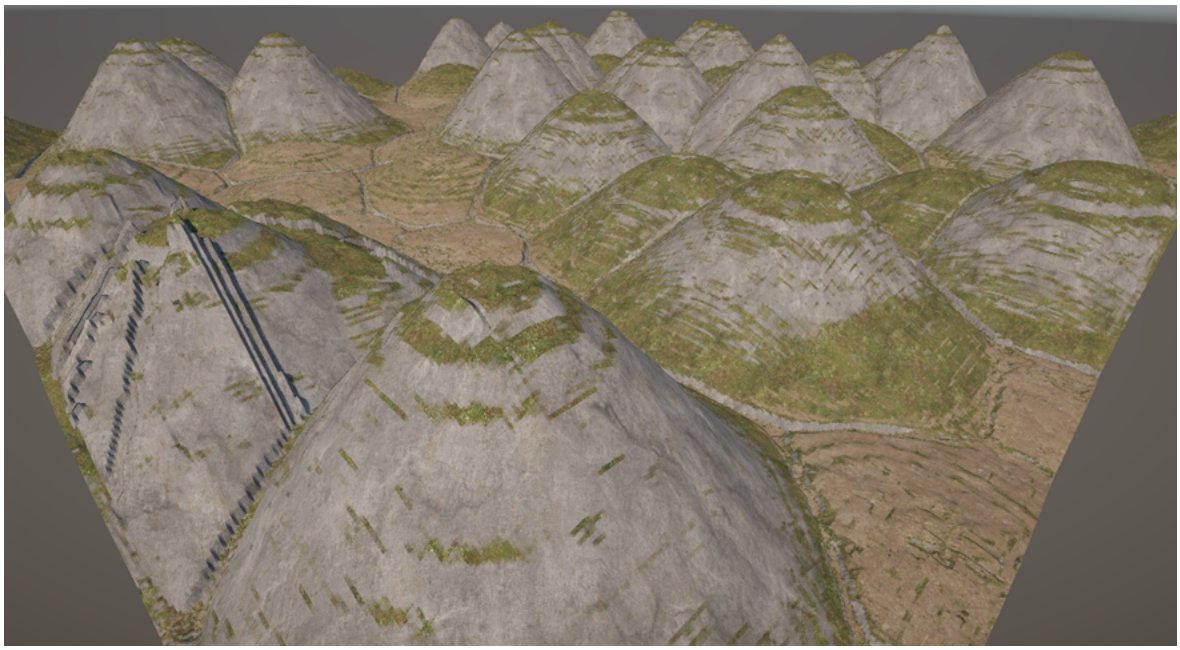


Figure 8: Heightmap based on Voronoi-Noise at 20,000 simulation iterations with rain and river sources.

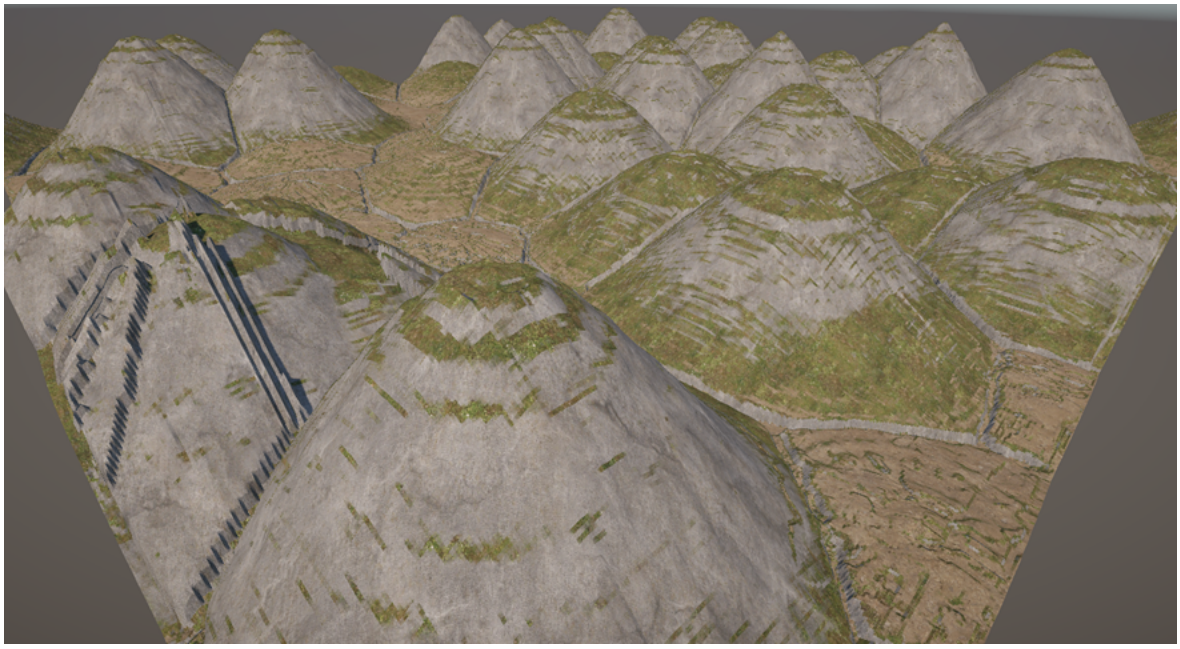


Figure 9: Heightmap based on Voronoi-Noise at 30,000 simulation iterations with rain and river sources.

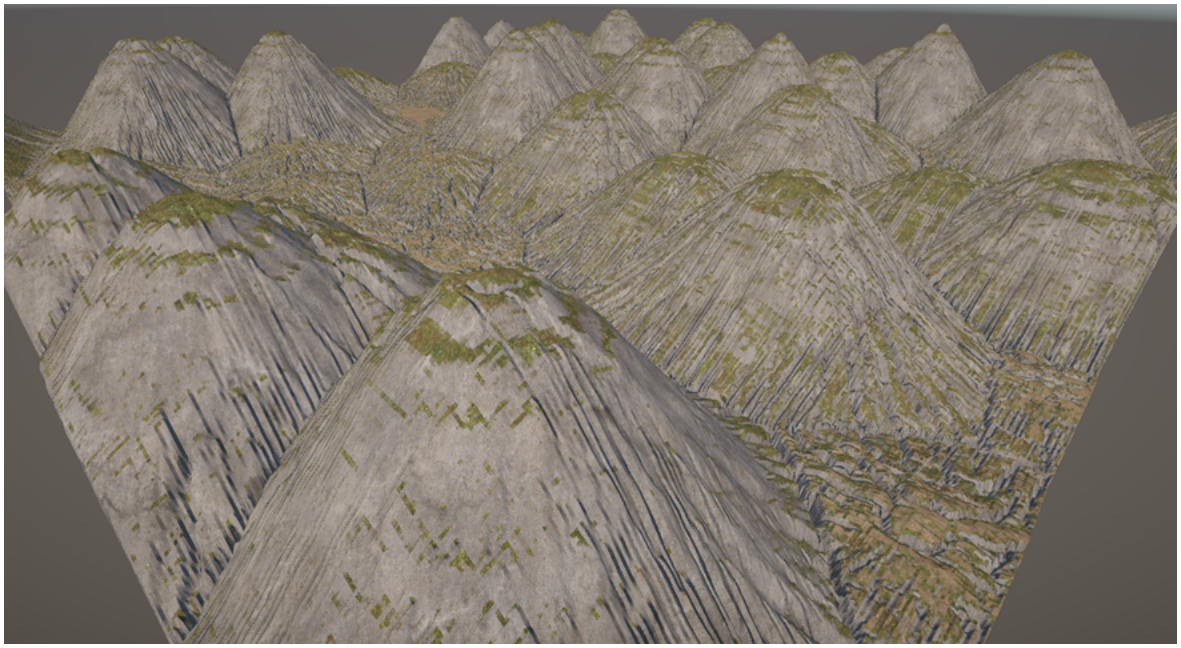


Figure 10: Heightmap based on Voronoi-Noise at 150,000 simulation iterations with only a rain source.

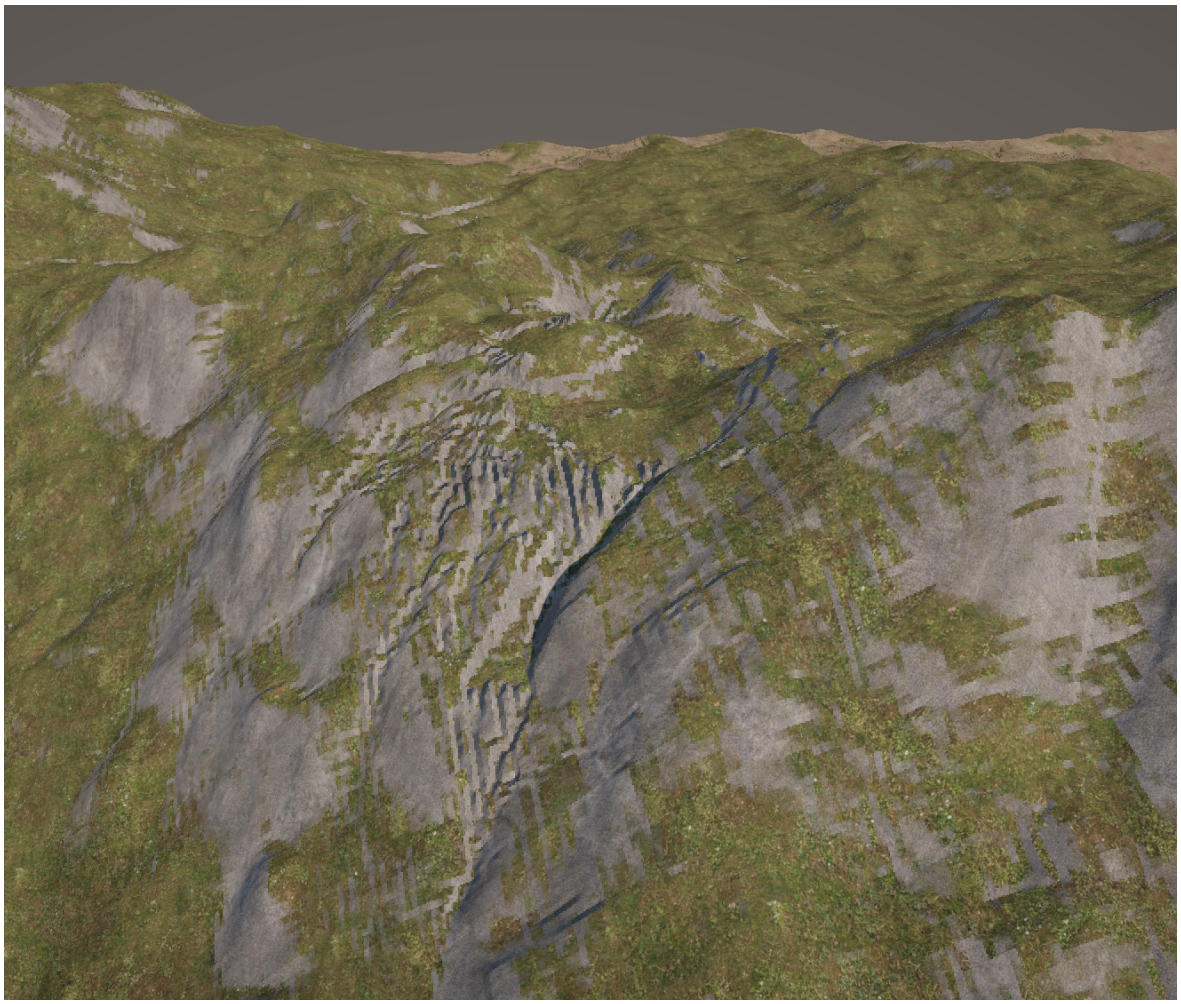


Figure 11: Detailed view of river source erosion results after 5,000 iterations on a detailed heightmap.

7 Conclusion

The erosion model introduced in 2007 by [MDH07] has improved in performance after 14 years since its introduction by an order of magnitude. The simulation has achieved real-time feedback and is interactive through parameters and water sources being editable.
Many works published after [MDH07] are expanding the introduced erosion and shallow-water model by adding new erosion types, layered data structures or other environmental effects. But this hydraulic erosion model by [MDH07] is often used as the basis for expansions.
While visual erosion results are convincing, the model is not modeling physically correct erosion and deposition processes with inaccurate sediment transportation mentioned in Section 4. The erosion model has shown to be sub optimal for more uniform terrains, resulting in uniform erosion effects like the river source erosion in Figure 8.

Future Work

Future works could combine multiple erosion effects with other environmental effects like vegetation seen in [HR23] or include improved and more detailed terrain representations like the layered data structure. This could improve the steep transition between eroded and uneroded terrain seen in Figure 9. An interesting topic to explore in the future might be a combined approach of the presented shallow-water approach for water bodies like coastlines, lakes and rivers combined with a lagrangian particle approach for rain and spray erosion. This would allow both approaches to utilise their advantages more effectively.

References

[AK09] Rémi Abgrall and Smadar Karni. Two-layer shallow water system: A relaxation approach. *SIAM Journal on Scientific Computing*, 31(3):1603–1627, 2009. 8

[ASA07] Nguyen Hoang Anh, Alexei Sourin, and Parimal Aswani. Physically based hydraulic erosion simulation on graphics processing unit. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, GRAPHITE '07, page 257–264, New York, NY, USA, 2007. Association for Computing Machinery. 9

[BCMP13] BN Biswas, Somnath Chatterjee, SP Mukherjee, and Subhradeep Pal. A discussion on euler method: A review. *Electronic Journal of Mathematical Analysis and Applications*, 1(2):2090–2792, 2013. 11

[Ben07] Bedrich Benes. Real-Time Erosion Using Shallow Water Simulation. In John Dingliana and Fabio Ganovelli, editors, *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2007)*. The Eurographics Association, 2007. 9

[BF01] B. Benes and R. Forsbach. Layered data representation for visual simulation of terrain erosion. In *Proceedings Spring Conference on Computer Graphics*, pages 80–86, 2001. 9

[BF02] Bedřich Beneš and Rafael Forsbach. Visual simulation of hydraulic erosion. *Journal of WSCG*, 10:79–86, 2002. 10

[BMPG11] Louis Borgeat, Philippe Massicotte, Guillaume Poirier, and Guy Godin. Layered surface fluid simulation for surgical training. In Gabor Fichtinger, Anne Martel, and Terry Peters, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011*, pages 323–330, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 8

[Bre09] Didier Bresch. Chapter 1 - shallow-water equations and related topics. In C.M. Dafermos and Milan Pokorný, editors, *Handbook of Differential Equations*, volume 5 of *Handbook of Differential Equations: Evolutionary Equations*, pages 1–104. North-Holland, 2009. 8

[BTHB06] Bedřich Beneš, Václav Těšínský, Jan Hornyš, and Sanjiv K. Bhatia. Hydraulic erosion. *Computer Animation and Virtual Worlds*, 17(2):99–108, 2006. 9, 10

[Cas90] Vincenzo Casulli. Semi-implicit finite difference methods for the two-dimensional shallow water equations. *Journal of Computational Physics*, 86(1):56–74, 1990. 8

[CCB+18] Guillaume Cordonnier, Marie-Paule Cani, Bedrich Benes, Jean Braun, and Eric Galin. Sculpting mountains: Interactive terrain modeling based on subsurface geology. *IEEE Transactions on Visualization and Computer Graphics*, 24(5):1756–1769, 2018. 10

[CF88] Peter Constantin and Ciprian Foiaş. *Navier-stokes equations*. University of Chicago press, 1988. 8

[CGG+17] Guillaume Cordonnier, Eric Galin, James Gain, Bedrich Benes, Eric Guérin, Adrien Peytavie, and Marie-Paule Cani. Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Trans. Graph.*, 36(4), jul 2017. 10

[CMF98] Norishige Chiba, Kazunobu Muraoka, and Kunihiko Fujita. An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and Computer Animation*, 9:185–194, 1998. 10

[CMM90] Alexandre Joel Chorin, Jerrold E Marsden, and Jerrold E Marsden. *A mathematical introduction to fluid mechanics*, volume 3. Springer, 1990. 8

[DGV+18] François Dagenais, Julián Guzman, Valentin Vervondel, Alexander Hay, Sébastien Delorme, David Mould, and Eric Paquette. Real-time virtual pipes simulation and modeling for small-scale shallow water. In *VRIPHYS*, pages 45–54, 2018. 9

[GGP+19] Eric Galin, Eric Guérin, Adrien Peytavie, Guillaume Cordonnier, Marie-Paule Cani, Bedrich Benes, and James Gain. A review of digital terrain modeling. *Computer Graphics Forum*, 38(2):553–577, 2019. 5, 7, 29

[HD01] Russell S Harmon and William W Doe. *Landscape erosion and evolution modeling*. Springer Science & Business Media, 2001. 15

[HR23] Brian Hawkins and Brian Ricks. Improving virtual pipes model of hydraulic and thermal erosion with vegetation considerations. *The Visual Computer*, 39(7):2835–2846, 2023. 10, 21, 25

[JS85] PY Julien and DB Simons. Sediment transport capacity of overland flow. *Transactions of the ASAE*, 28(3):755–0762, 1985. 15

[KBKŠ09] Peter Krištof, Bedrich Beneš, Jaroslav Křivánek, and Ondrej Šťava. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum*, 28(2):219–228, 2009. 10

[Kel13] Timo Kellomäki. Interaction with dynamic large bodies in efficient, real-time water simulation. 2013. 9

[Kel15] Timo Kellomäki. Rigid body interaction for large-scale real-time water simulation. *International Journal of Computer Games Technology*, 2014:15–15, 2015. 9

[Kel17] Timo Kellomäki. Fast water simulation methods for games. *Comput. Entertain.*, 16(1), dec 2017. 7

[KMN88] Alex D. Kelley, Michael C. Malin, and Gregory M. Nielson. Terrain simulation using a model of stream erosion. *SIGGRAPH Comput. Graph.*, 22(4):263–268, jun 1988. 7

[MDH07] Xing Mei, Philippe Decaudin, and Bao-Gang Hu. Fast hydraulic erosion simulation and visualization on gpu. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pages 47–56, 2007. 5, 8, 10, 11, 12, 13, 15, 21, 25, 29

[MKM89] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *SIGGRAPH Comput. Graph.*, 23(3):41–50, jul 1989. 5, 7, 9

[MY97] David Mould and Yee-Hong Yang. Modeling water for computer graphics. *Computers & Graphics*, 21(6):801–814, 1997. Graphics in Electronic Printing and Publishing. 11

[Nag98] Kenji Nagashima. Computer generation of eroded valley and mountain terrains. *The Visual Computer*, 13(9-10):456–464, 1998. 9

[NWD05] Benjamin Neidhold, Markus Wacker, and Oliver Deussen. Interactive physically based fluid and erosion simulation. In *Eurographics workshop on natural phenomena*, 2005. 9, 10

[OH95] J.F. O'Brien and J.K. Hodgins. Dynamic simulation of splashing fluids. In *Proceedings Computer Animation'95*, pages 198–205, 1995. 8, 11, 12, 13

[ON00] Koichi Onoue and Tomoyuki Nishita. A method for modeling and rendering dunes with wind-ripples. In *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*, pages 427–428. IEEE, 2000. 9

[PB07] Jean-Philippe Pons and Jean-Daniel Boissonnat. Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. 9

[Pon90] V. M. Ponce. Generalized diffusion wave equation with inertial effects. *Water Resources Research*, 26(5):1099–1101, 1990. 8

[RPP93] Pascale Roudier, Bernard Peroche, and Michel Perrin. Landscapes synthesis achieved through erosion and deposition process simulation. In *Computer Graphics Forum*, volume 12, pages 375–383. Wiley Online Library, 1993. 9

[RSH01] William W. Doe Russell S. Harmon. *Landscape Erosion and Evolution Modeling.* Springer New York, NY, 2001. 5

[ŠBBK08] Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek. Interactive terrain modeling using hydraulic erosion. In *Proceedings of the 2008 acm siggraph/eurographics symposium on computer animation*, pages 201–210, 2008. 10

[SK16] Věra Skorkovská and Ivana Kolingerová. Complex multi-material approach for dynamic simulations. *Computers & Graphics*, 56:11–19, 2016. 9

[SKB15] Věra Skorkovská, Ivana Kolingerová, and Bedrich Benes. Hydraulic erosion modeling on a triangular mesh. *Surface Models for Geosciences*, pages 237–247, 2015. 10

[SOH99] Robert W Sumner, James F O'Brien, and Jessica K Hodgins. Animating sand, mud, and snow. In *Computer Graphics Forum*, volume 18, pages 17–26. Wiley Online Library, 1999. 9

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 121–128, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 16

[TJ10] Luther A Tychonievich and MD Jones. Delaunay deformable mesh for the weathering and erosion of 3d terrain. *The Visual Computer*, 26:1485–1495, 2010. 9

[VBHS11] Juraj Vanek, Bedrich Benes, Adam Herout, and Ondrej Stava. Large-scale physics-based terrain editing using adaptive tiles on the gpu. *IEEE Computer Graphics and Applications*, 31(6):35–44, 2011. 10

List of Figures

List of Tables