# Simulating Hidraulic Erosion Utilizing Graphics Processors

*Bence Salamin*
*John von Neumann Faculty of Informatics*
*Obuda University*
Budapest, Hungary
*sbence@stud.uni-obuda.hu*

*Sándor Szénási*
*John von Neumann Faculty of Informatics*
*Obuda University*
Budapest, Hungary
*szenasi.sandor@nik.uni-obuda.hu*

2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY) | 979-8-3503-8560-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SISY62279.2024.10737605

***Abstract*— The forces of nature have been transforming our world for millions of years. One of these forces is hydraulic erosion, which uses the movement and displacement of water to erode and relocate soil. Most of the terrain producing algorithms are purely mathematical, and the results often seem artificial at best. The solution to this comes in form of simulation software, that emulate nature's forces to produce said terrains. Today's technology allows us to harness the computational powers of GPUs in order to speed up work. The following work shall map and overcome the obstacle that is the simulation of hydraulic erosion and apply GPGPU techniques to accelerate the process.**

***Keywords*— Hydraulics, Erosion, Simulation, Computer Graphics, GPGPU***

## I. Introduction

The world around us is constantly changing. These transformations are an integral part of the forces of nature that affect everything on the planet, whether it is the Central Highlands or a backyard garden. What connects these two distant geographical locations are the soil-transforming processes that shape them. Of course, we are not talking about lithospheric movements gradually altering residential gardens over centuries, as this is not a relevant threat in our daily lives. The more significant soil-transforming process lives in the form of erosion.

Erosion is a geological phenomenon in which materials on the Earth's surface are carried away by various mediums and deposited elsewhere. Specifically, hydraulic erosion refers to the movement of materials by the flow of water, which can be triggered by rivers/streams, floods, coastal waves, or rainfall depending on climate. This phenomenon is a significant part of many engineering problems, such as construction and agriculture. As simple as this idea sounds, creating an accurate simulation suitable for it is quite complex.

In the world of computer science, erosion aims to overcome the problem of creating realistically appearing terrains. Existing fractal implementations do not always produce visually satisfying, natural-looking maps, and noise-based approaches do not always yield results. The path to creating the resulting maps is entirely mathematical and does not apply methods for terrain transformation found in nature. Since the most significant soil degradation process can be simulated by the movement of water, the world of computer graphics has been dealing with producing appropriate solutions for nearly 30 years. The task poses challenges in various fields, from game development to filmmaking. When simulating water movement, as it is extremely resource-intensive, the idea of implementing it with graphical processors is very appealing.

## II. Related Work

The process of hydraulic erosion is difficult to describe mathematically. It depends on numerous dynamically changing parameters, making it challenging to achieve perfect results even with large-scale systems used at the state level for soil loss prediction (such as WEPP, RUSLE2, etc.). Furthermore, most equations describing the processes are mostly empirical. Comprehensive simulation of the event has been studied for over 20 years, and as a result, several successful solutions have been provided by various researchers.

In the following paragraphs we will take a look at previously applied methods for this type of simulation.

### A. Data Structure

The selection of the data structure requires a compromise between accuracy and speed. The more data we store, the more accurate and computationally demanding the simulation becomes. Most programs choose between two structures: the heightmap (HM) and voxels. Voxels define triangles, typically used for fast rendering. Cubes can also be considered, here the advantages of using heightmaps become clearer. These two data structures are evaluated in [7].

The voxel approach results in a more accurate simulation because they can provide much more data/information about themselves and their environment. However, this comes at a cost. In exchange for a more accurate simulation, we get a significantly slower one, and the required storage space also increases substantially. Heightmaps are usually two-dimensional matrices where each "cell" provides feedback on the data stored within it at the given coordinates. Assuming $\underline{n}$ bytes of data are stored in each cell, an HM of size 1024x1024 requires $n$ MB of storage, whereas using voxels, $n$ 10243, meaning $n$ GB will be needed.

Furthermore, several methods effectively utilize the opportunities provided by HM [1, 5], such as the simplicity of parallelization with minimal overhead, low simulation speed without sacrificing realistic implementation. Initially, the simulation was divided into four elementary steps, later refined for accuracy [2] and further expanded [3]. The fundamental

principle of maintaining system stability while being able to execute these steps in any order remained unchanged. The sequence of steps begins with the appearance of water and ends with its evaporation.

Each element of the HM stores data about its own coordinates and the state of the simulation. Consequently, the data structure allows for GPU parallelization without significant complications.

### *B. Modeling Water and Erosion*

Water mobility is an integral part of the project, addressed by the Navier-Stokes equations in the mathematical world. These equations were processed [5] to be applicable for computer simulation. They were solved in multiple dimensions depending on the simulation's goals. The 3D representation allows for the natural formation of caves and underground structures. However, this is limited to maps with a low number of cells [11] and is highly resource-intensive.

In contrast, 2D implementations already have low-cost computational solutions and implementations [4, 8]. Often, [5] applies the Shallow Water Modell (SWM) for 2-3D interactive processing of Navier Stokes equations. The model had certain constraints, such as assuming a grid-based HM data structure. In contrast, the SWM is perfectly suitable for simulating hydraulics on maps with a larger number of cells and is also applicable for GPU parallelization, as demonstrated by [2].

Another approach is presented in [1], where soil erosion is simulated through an interactive process. This system is called Smoothed Particle Hydrodynamics (SPH). The essence of the processing is to divide water and the surface into molecules but does not differentiate between them, rather moves the molecules using a so-called "donor-acceptor" system along with the water. However, this processing does not allow for the formation of three-dimensional geographical elements and does not make significant distinctions in computational demand compared to projects using the SPH. [1] is also implemented on the CPU, making the molecule-based approach unnecessarily complex for GPU processing.

It is important to highlight the method presented in SWM [8] for moving water between cells. The so-called "pipe model" method connects the eight neighbors of a cell with a virtual pipe, which simplifies the determination of the amount of fluid leaving the cell and entering it, and more importantly, makes it easier to determine velocity vectors. [2] first parallelized the method on the GPU and simplified it by connecting virtual pipes only to the neighboring four cells.

The implementation of erosion in almost all cases (except for example [1]) estimates the force exerted on the soil directly with a velocity vector and determines the amount of sediment that can be carried with the water, similar to events occurring in nature.

### *C. Visualization*

The choice of data structure significantly affects visualization possibilities. Therefore, we discard voxel-based data storage and assume an HM-based structure from now on.

[2] applies the multi-pass algorithm for visualization. The method passes two-dimensional textures to the GPU memory, then draws a quad parallel to the image space. Then, for each pixel of the quad, it calculates the amount of data needed to create the necessary geometric primitives later. This calculation is done by the "fragment shader," which is a phase of the OpenGL pipeline. Finally, the output data is written into a target element (these can be textures or render targets) used for the next such process. One advantage of this solution is that the process is much faster on GPU implementations and scales well with grid resolution. Cells located on the edges of the grid require special treatment. In this solution, the fragment shader determines if there is an extreme value at a given location, so there is no need for a fragmentation process.

[3] works by stacking two-dimensional four-channel floating-point texture layers. These are attached to a single frame buffer object. It stores all associated simulation parameters in texel elements (which, like cells, are positioned at the same coordinates). Then, using the two buffers, it applies the ping-pong method, defining one buffer as input and the other as output, swapping them after the calculation, and restarting. This implementation is done in a single fragment shader, which uses the OpenGL multiple render target function for visualization.

There's also mention of ray tracing. Newer algorithms allow real-time implementation of this method and optimally utilize the capabilities of modern graphics cards. There are solutions excellently supports modern APIs with its processing and functions.

For HM-matrix data structures, to simplify implementation, we can consider simpler alternatives that do not require GPU programming, such as Excel, which would allow for two-dimensional representation with color coding. MATLAB is also noteworthy, as it would be able to visually represent a three-dimensional plot made from our multidimensional array.

## III. Simulation Model

The process to be simulated is broken down into components [3], where various formulas will be applied. The steps for calculating erosion are divided into five main parts, similar to processes occurring in nature:

1) *Water appears in some form.*
2) *Simulation of water movement.*
3) *Sediment uptake due to water movement.*
4) *Movement of sediment in the water.*
5) *Evaporation of water, deposition of sediment.*

Our model will apply discrete time steps to calculate the five main steps, where $t$ indicates the parameter's state at a given time, and $\Delta t$ represents the change in time for the simulation. After calculating the five steps, the next time step occurs ($t$+$\Delta t$), and the process starts again. The simulation will always work with a fixed number of steps, requiring the program to be rerun for further simulation. For some of the equations presented below, we will need the different states of individual fields, so we will distinguish between states using subscripts 1,2, 3, …, $n$.

As discussed earlier, our program will be based on SWM; therefore, we will use the HM data structure. Consequently, the simulated terrain will be built in a 2D matrix, where the elements

will be cells. Their positions will be denoted by the cell's (x, y) coordinates.

Each cell contains data necessary for the logic's operation and information about its own state. These data will be regularly updated during the simulation. $di(x,y)$ represents the water, while $bi(x,y)$ represents the height of the soil.

*A. Movement of Water*

The simulation of water movement in the literature presents similar approaches [1, 3]. We determine the water movement by combining the works of [3, 4], and the following equations are based on these works. In our simulation, the water height is increased by natural water sources (streams, rivers, etc.) and precipitation events. The precipitation event is described by the following equation:

$$d1(x,y) = dt(x,y) + \Delta t \cdot rt(x,y) \cdot Kr \qquad (1)$$

In the above equation, $d1$ represents the water height stored in the given cell, $rt$ is a locally determined value indicating the amount of precipitation received by the cell, and $Kr$ is a global parameter used for scaling rainfall.

Next, we will discuss the flow of water between cells. We will implement the flow of water between cells using the PM adaptation used in [8], which has been successfully used for erosion simulation [4].

Each cell (x, y) has four virtual pipes through which it can communicate water with its neighbors. We only store the outflowing water quantity from the cell, denoted by $f = (FT, FB, FR, FL)$. The exchange of fluid between two cells is controlled by the pressure difference between them. The change in state of the upper neighbor is described as follows:

$$F^T_{t+\Delta t} = \max(0, F^T_t(\mathrm{x,y}) + \Delta \mathrm{t} \cdot \mathrm{A}\frac{g \cdot \Delta h^T(x,y)}{l}) \qquad (2)$$

Where $A$ represents the cross-sectional area of the virtual pipe (in m²), $g$ is the gravitational constant, $l$ is the length of the virtual pipe (which is a constant 1 in our case), and $\Delta h^{\mathrm{T}}(x,y)$ is the height difference between the cell and its upper neighbor.

$$\Delta h^T(x,y) = b_t(x,y) + d_1(x,y) - b_t(x-1,y) - d_1(x-1,y) \qquad (3)$$

Communication with the other neighbors occurs similarly. However, there might be a problem with the quantity of water flowing out of the cell, as it might exceed the available amount. To avoid this, a variable $K$ is introduced to determine the amount that can be pumped out of the cell based on the amount of water stored in it. This step is discussed and justified in detail in [2].

$$\mathrm{K} = \min(1, \frac{d_1\, l_X l_Y}{(F^T + F^B + F^L + F^R) \cdot \Delta \mathrm{t}}) \qquad (4)$$

Where $lX$ and $lY$ represent the distance between cells on the map, in the x/y direction, respectively. The upper limit of the formed border is:

$$F^i_{t+\Delta t}(\mathrm{x,y}) = \mathrm{K} \cdot F^i_{t+\Delta t}(\mathrm{x,y}),\ \mathrm{i} = \mathrm{T,B,L,R} \qquad (5)$$

Finally, the change in water is calculated with the inflow from neighboring cells ($fin$) and outflow to neighboring cells ($fout$):

$$\Delta W(x,y) = \Delta t \cdot (\sum f_{in} - \sum f_{out}) = \Delta t \cdot \left(F^R_{t+\Delta t}(x,y+1) + F^L_{t+\Delta t}(x,y-1) + F^T_{t+\Delta t}(x+1,y) + F^B_{t+\Delta t}(x-1,y) - \sum_{i=T,B,L,R} F^i_{t+\Delta t}(x,y)\right) \qquad (6)$$

The water height in the cells is updated as follows:

$$d_2(x,y) = d_1(x,y) + \frac{\Delta W(x,y)}{l_X l_Y} \qquad (7)$$

It is also necessary to determine the velocity vector $V$, which is required to simulate erosion due to water pressure (in this study, only horizontal velocity is considered). This can be done by calculating the average flow of water passing through the cell. The water passing through cell (x, y) in the X direction:

$$\Delta V_x = \frac{F^R(x-1,y) - F^L(x,y) + F^R(x,y) - F^L(x+1,y)}{2} \qquad (8)$$

Similarly, the calculation is done in the Y ($\Delta Vy \Delta Vy$) direction. After the calculations, to evaluate the new state, we need to consider water evaporation. We introduce parameter $Ke$ into our simulation, which scales evaporation.

$$d_{t+\Delta t}(x,y) = d_2(x,y) \cdot (1 - K_e \cdot \Delta t) \qquad (13)$$

With this, the calculation of the fifth step is completed, and the next iteration can begin.

Due to the structure of the HM, we need to be careful with the boundaries, namely, what happens if one of the neighbors points outside the map, e.g., cell (0, 0). During the implementation of the program, we avoid water outflow from the map, in such events by treating the existing cells specially, where the appropriate $Fi$ value(s) store a constant value of: -∞ during calculations when encountering such a boundary value.

*B. Erosion Systems*

The effect of water on the ground causes a certain amount of material to detach, which is then carried away by the water. However, this alone is not sufficient. Water also needs an upper limit on the amount of material it can carry. We describe this upper limit with a simplified equation presented in [9]:

$$C(x,y) = K_c \cdot \sin(\alpha(x,y)) \cdot |\bar{v}(x,y)| \qquad (9)$$

Where: $C(x, y)$ describes the sediment carrying capacity of the water flowing in cell (x, y). $Kc$ is a global parameter that scales this value. $\alpha$ is the local slope angle, calculated as:

$$\alpha = \cos^{-1}(\frac{\overline{gv} * \overline{hv}}{|\overline{gv}| * |hv|}) \qquad (9a)$$

Where: $gv$ is the local gradient vector, $hv$ is the projection of $gv$ onto the x, y plane. cos(α) is the scalar product divided by the product of the vectors' lengths

However, a problem arises if the size of $\alpha\alpha$ is quite low, as this makes the erosivity significant in places where the slope of the ground is nearly horizontal, unnecessarily consuming computational resources. To avoid this, we introduce $C\alpha$, a framework determined by the user. This will regulate the acceptability of the $\alpha\alpha$ value. Then, if $C$ is greater than $ss$,

which represents the sediment stored in the water from the ground, we dissolve part of it in the water:

$$b_{t+\Delta t} = b_t - K_s(C - s_t)$$
$$s_1 = s_t + K_s(C - s_t) \quad (10)$$

Where: $Ks$ is the dissolution coefficient, also a parameter set by the user, $s1$ is the quantity of sediment carried. Otherwise, if $C$ is smaller than $s$, their roles in equation (10) are swapped, and $Ks$ is replaced with $Kd$, the deposition coefficient, causing some of the carried sediment to deposit. However, this would not consider the law of water displacement, so to address this, we add the collected sediment to the water height:

$$d_3 = d_2 + \Delta t \cdot K_s(C - s_t) \quad (10a)$$

Conversely:

$$d_3 = d_2 - \Delta t \cdot K_s(s_t - C) \quad (10b)$$

After calculating the erosive processes, we calculate the movement of sediment using the advection equation:

$$\frac{\partial s}{\partial t} + (\vec{v} \cdot \nabla s) = 0 \quad (11)$$

We solve this equation using the method presented in [11]. We determine the new value of the cell $v\,(u, v)$**:**

$$s_{t+\Delta t}(x, y) = s_1(x - u \cdot \Delta t, y - v \cdot \Delta t) \quad (12)$$

If $s1$ were to fall outside the map, its value would be determined by linear interpolation from its four neighbors.

## IV. Experimental results

When examining the measured time results, we take into account two main factors. The first is the configuration of the computer on which we execute the simulation. Consequently, the achieved results vary from one computer to another. The two most influential factors in our case are the CPU and GPU units. The results presented below were achieved on a thirteenth generation Intel(R) Core(TM) i5-13600K 3.50 GHz processor and a NVIDIA GeForce GTX 1050Ti graphics card released in 2016. There is a significant generational and performance difference between the two devices, further outlining the outcome of parallelization.

The other factor is the size of the map (see: Table I), on which erosion processes take place. This is responsible for the number of cells and consequently the number of threads that need to be launched. In this case, the dimensions of the map are multiples of thirty-two to fully harness GPU acceleration. Other simulation parameters have a strong impact on computational requirements, but they are documented for reproducibility. The program executes the appropriate sequence of steps 50 times, then displays the final result (Figure 1). The measurements in the table refer to the total number of cycles. Each metric was executed on the "cone" maps obtained from the same instance of the program. The simulation is primarily performed on the GPU, but the CPU runtime is also displayed as a reference value.

Table. I: Measured simulation time.

| Size of map | Δt CPU(s) | Δt GPU(s) | Δt MAT(s) | Δt SMAT(s) |
|---|---|---|---|---|
| 256x256 | 3.3678 | 0.7072 | 1.0313 | 4.3939 |
| 512x512 | 12.4146 | 1.6418 | 2.8567 | - |
| 1024x1024 | 51.1994 | 4.5739 | 5.1003 | - |
| 2048x2048 | 198.3065 | 16.3993 | 11.1474 | - |

Based on the results, it is clear that thanks to parallelization, we can achieve nearly fourteen times acceleration. This value is particularly significant for high-resolution maps. The visualization process sometimes incurs almost the same cost as the simulation itself. However, it is important to note that this value represents the entire visualization process. It includes uploading data to the engine's memory twice, as we visualize not only the end but also the initial state.

An advantage can be gained if we perform the generation without closing the program. This is because the initialization time of the Matlab engine can be ignored after the first run.

The measured time and the size of the map are nearly directly proportional. Based on the obtained data, if we increase the number of cells fourfold, then the CPU simulation time stays within εC = 0.4969613 seconds and the GPU simulation time within εD = 0.8941916 seconds the time distanse from four times the previous measurement.

Figure 1. shows the result of a simulation using our program.

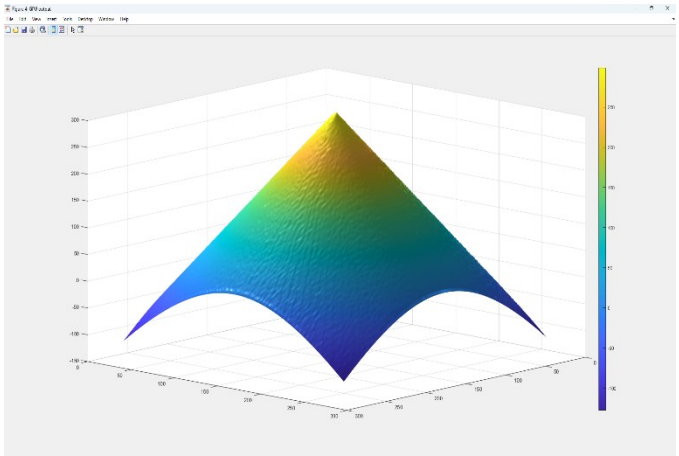

Fig. 1. Simulation result

Comparing the results proves to be a complex task, as relevant literature distinguishes a wide spectrum of software and hardware differences. Time values measured in [5] are most compatible with ours for comparison, due to the similarity in the execution of the simulation and calculation steps. Ultimately, we achieved approximately threefold acceleration in the measured time values, but the significant hardware capacity difference between the two projects must be taken into account.

## V. Conclusion and Future work

In this paper, we have designed a simulation program aimed at simulating real physical phenomena. To achieve this, we investigated and applied numerous methods still used in computer graphics to this day [11]. We compared the SWM method with particle-based simulation tools and defined a Matlab-based visualization model against GPU pipelines and other two-dimensional methods. Furthermore, we examined agricultural applications to bring the simulation closer to nature in this mathematical manifestation.

The implementation of the planned application has been documented component by component, and the parameters used for testing the operation have been provided to the readers.

We tested the components of our program using both black- and white-box methods. During the tests, we ensured execution according to system requirements and validated input and output data.

During the evaluation of the specified program implementation, we confirmed that the planned software is capable of quickly and efficiently mapping the process of hydraulic erosion. The user has access to numerous tools in the development process, allowing them to refine the execution according to their needs and thus providing exceptional versatility.

When evaluating the final results, we confirmed the correctness of the operation and its suitability for creating maps that appear realistic. The planned application is successfully able to process matrix data structures with the parallelized implementation of the SWM. Our input interface is intuitive and guarantees the correctness of input data, while also allowing for easy expansion of the input data list. During visualization, we have full access to and interaction with the generated terrain.

Furthermore, the program remains open to further development and optimization of computational processes and systems. Thermal erosion could be a potential focus on further development and re.

Heat/thermal erosion provides a perfect example of the opportunities offered by natural processes. The soil transformation process takes into account heat changes, which depend on the movement of the sun and the heat retention of the air. Such an implementation can be observed [12,13]. This natural phenomenon appears in numerous studied implementations, and a noticeable trend is the finer-grained state of the final result compared to tools operating purely on hydraulic principles.

Additionally, there is the possibility of introducing layers mentioned in [6], where the erosion processes depend on the geological quality of a layer. Consequently, this allows for a more detailed exploration of erosion process properties. Here, expansion of functions performing erosion and the cell data structure would be required.

Introducing custom types for individual soil layers is conceivable, storing information on the expected behavior during soil degradation processes or how they assist them. Functions could erode surfaces from top to bottom, and it's possible to replicate the decay and sliding of looser materials.

Our current visualization method comes with strong limitations. Initiating a motor is time-consuming, and each visualization of a map occupies significant memory space. However, beyond these constraints, there are still untapped opportunities in leveraging Matlab tools comprehensively.

## References


[1] Krištof, P., Beneš, B., Křivánek, J., & Šťava, O. Hydraulic Erosion Using Smoothed Particle Hydrodynamics. Computer Graphics Forum, 28 (2), 2009, pp. 219-228

[2] Xing, M., Philippe, D., Bao-Gang, H. Fast Hydraulic Erosion Simulation and Visualization on GPU. In 15th Pacific Conference on Computer Graphics and Applications, Pacific Graphics, 2007, pp. 47-56

[3] Jákó, B., Tóth, B. Fast Hydraulic and Thermal Erosion on GPU. In: *Eurographics*. 2011, pp. 57-60

[4] Bryan, R. B. Soil Erodibility and Processes of Water Erosion on Hillslope. Geomorphology, 32 (3-4), 2000, pp. 385-415

[5] Kass, M., & Miller, G. Rapid, Stable Fluid Dynamics for Computer Graphics. ACM SIGGRAPH Computer Graphics, 24(4), 1990, 49-57 old.

[6] Šťava O. Interactive Terrain Modeling Using Hydraulic Erosion. In: Proceedings of the 2008 Acm Siggraph/Eurographics Symposium on Computer Animation. 2008, pp. 201-210

[7] Benes, B., & Forsbach, R. (n.d.). Layered Data Representation For Visual Simulation Of Terrain Erosion. Proceedings Spring Conference on Computer Graphics, 2001, pp. 80-86

[8] O'brien, J. F., Hodgins, J. K. Dynamic Simulation Of Splashing Fluids. In: Proceedings Computer Animation '95. IEEE, 1995, pp. 198-205

[9] Julien, P. Y., Simons, D. B. Sediment Transport Capacity of Overland Flow. Transactions of the ASAE, 28 (3), 1985, pp. 755-762

[10] Stam, J. Stable Fluids. In SIGGRAPH ’99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, 1999, pp. 121-128

Liu, Y., X. Liu, E. Wu. Real-time 3D Fluid Simulation On GPU with Complex Obstacles. In Proceedings of Pacific Graphics’04, 2004, pp. 247-256

[11] Cordonnier, G., Braun, J., Cani, M.-P., Benes, B., Galin, É., Peytavie, A., & Guérin, É. Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion. Computer Graphics Forum, 35(2), 2016, pp. 165–175

[12] Whitton, Mary C. eminal Graphics Papers: Pushing the Boundaries, Volume 2, Association for Computing Machinery, 2023

[13] Porkolab, L, and Lakatos, I. "A Simulation System for Testing Side Crashes, in Non-Traditional Seating Positions, for Self-Driving Cars." Acta Polytechnica Hungarica 20.7 (2023): 63-82.