Fast Hydraulic and Thermal Erosion on the GPU

Balázs Jákó*
Supervised by: Balázs Tóth†

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest/Hungary

Abstract

Computer games, TV series, movies, simulators, and many other computer graphics applications use external scenes where a realistic looking terrain is a vital part of the viewing experience. Creating such terrains is a challenging task. In this paper we propose a method that generates realistic virtual terrains by simulation of hydraulic and thermal erosion on a predefined height field terrain. The model is designed to be executed interactively on parallel architectures like graphics processors.

Keywords: Erosion, simulation, GPU, GPGPU, hydraulic, thermal.

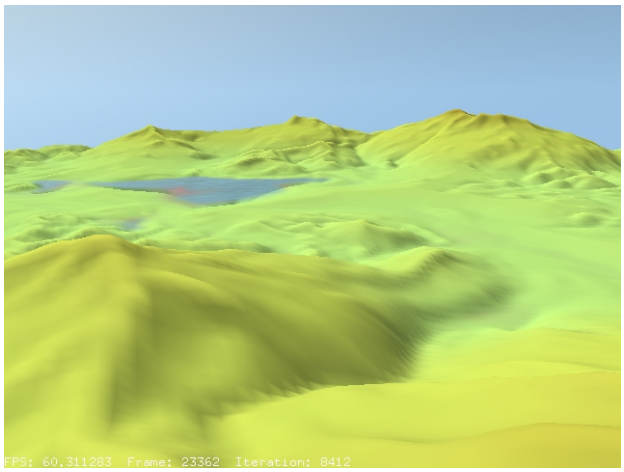


Figure 1: Landscape with mountains, valleys, ridges, riverbeds and lakes, generated by our method.

1 Introduction

Natural eroded terrains have some typical features like valleys, riverbeds, ridges, etc. These are results of different kinds of erosion caused by water, thermal shocks, and wind. Hydraulic erosion is caused by running water on terrain surface generated by from falling rain and springs. The flowing water dissolves soil and transports it to lower locations where the dissolved sediment is deposited. Thermal erosion is caused by temperature changes caused by the alternation of the hot Sun and cold night air. Hard surfaces are cracking up into smaller parts, the decomposed material is moving down to lower areas due to gravity.

There are different approaches to generate virtual terrains with features caused by these phenomena. Methods based on fractal techniques build a terrain that is similar to real-world mountain scenery, but these are isotropic, making ridge and valley generation difficult. Topographical methods use structural models to simulate water systems. These emphasize water flow, but mountain slopes are less natural in results. Physics based models simulate erosion factors and their effects on terrain surface like water flow, thermal shocks, and wind. In our method, we focus on hydraulic and thermal erosion, because these have the most impact on the terrain surface.

The programmability of modern GPUs makes it possible to execute not only graphics algorithms but a wide range of other, more generic tasks [11]. Using general purpose graphics processing units (GPGPUs) for simulation is obvious when the simulation algorithm can be excuted in parallel. The architecture of the GPUs execute appropriate algorithms much faster than traditional CPUs which makes such simulations able to run interactively, allowing direct observation and manual intervention during execution.

Physics based erosion methods apply some kind of fluid simulation. Chiba et al. [4] were the first to propose simulating valleys and ridges using particle systems. Beneš et al. [3] presented a model that uses the Navier-Stokes equations on a 3D regular grid simulating the erosion process. Neidhold et al. [5] used simplified Newtonian physics model for velocity computation on a 2D grid. Up to this, all of these models are computationally expensive, and due to data dependencies they are hard to execute on a parallel hardware. 2D Navier-Stokes equations were solved efficiently by Harris [6] and Wu [14]. Kass et al. [7] solve shallow water equations in their model, which was also used by Beneš et al. [1], to simulate erosion in real-time. Mei et al. [8] used this model in their simulation with employing virtual pipes introduced by O'brien et al. [10] that is the key to parallel execution. Our approach is partly

*balazs.jako@hun-digital.hu
†tbalazs@iit.bme.hu

based on [8]. To improve the realism of the simulation, we adapted the thermal erosion model of Beneš et al. [2] by applying the idea of virtual pipes and merging it into the improved hydraulic erosion algorithm.

2 Erosion Model

Our hydraulic erosion model is an improved version of the method introduced by Mei, Decaudin and Hu [8]. This model works with a 2D uniform grid and uses the following quantities in each (x,y) cell (see Figure 2):

- terrain height b,
- water height d,
- suspended sediment amount s,
- water outflow flux $\mathbf{f} = (f^L, f^R, f^T, f^B)$,
- velocity vector $\overrightarrow{\mathbf{v}}$.

These values are updated in each iteration. The simulation iteration consists of the following five steps:

1. Water incrementation due to rain or water sources.
2. Flow simulation using shallow-water model. Computation of the velocity field and water height changes.
3. Simulation of the erosion-deposition process.
4. Transportation of suspended sediment by the velocity field.
5. Water evaporation.

These steps are executed in each iteration step, gradually changing the state variables in each cell. Let $b_t, d_t, s_t, f_t, \overrightarrow{v_t}$ denote the data elements at a given time t and Δt the time step. In the following, we summarize the calculations producing the values at the next $t+\Delta t$ time. Since the model calculates some variable values in two or more steps, we will use subscripts $1,2,...$ to distinguish the temporal values from the final output used in following iterations.

First, we simulate the effects of water arriving at the terrain surface. Unlike the original model, we use constant $r(x,y)$ rain rate for each cell instead of large randomly distributed raindrops falling down to surface. The rain rate specifies the water amount arriving at a given (x,y) cell during Δt time. This gives us more balanced and finer grained results in the long run. The water height is updated by the following formula:

$$d_1(x,y) = d_t(x,y) + \Delta t \cdot r_t(x,y) \cdot K_r \tag{1}$$

where K_r is a global simulation parameter that scales the overall rate of water increment, and d_1 is the intermediate value of the water height.

Then, we calculate the water flow between cells. Each (x,y) cell has four virtual pipes to the four neighbors which

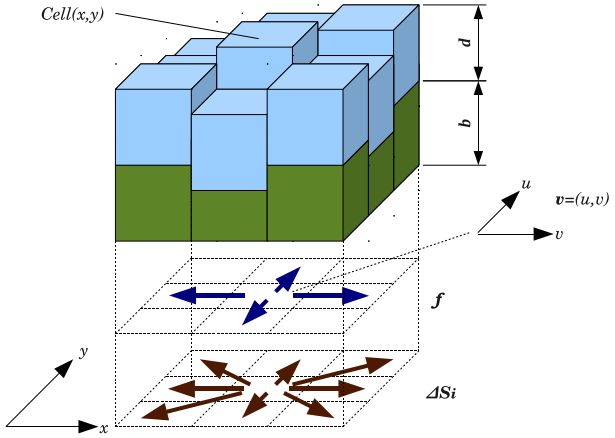


Figure 2: Water and thermal sediment flow model.

transport water outward from the given cell. Neighboring cells also have four virtual pipes, transporting water to opposite directions. The water outflow flux is updated with the pressure difference between interconnected cells. Let's denote $\mathbf{f} = (f^L, f^R, f^T, f^B)$ the outflow flux in a given (x,y) cell, where f^L is the outflow flux to the left neighbor at $(x-1,y)$, and similarly f^R, f^T, f^B are the outflow fluxes to right, top, bottom directions, respectively. We calculate the change of f^L as:

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

where A is the cross section area of the virtual pipe, g is the gravity, l is the length of the virtual pipe, $\Delta h^L(x,y)$ is the height difference between the left and the current cell:

$$\Delta h^L(x,y) = b_t(x,y) + d_1(x,y) - b_t(x-1,y) - d_1(x-1,y) \tag{3}$$

The calculation of f^R, f^T, f^B is performed in a similar way. The total outflow should not exceed the total amount of the water in the given cell. If the calculated value is larger than the current amount in the given cell, then $\mathbf{f}$ will be scaled down with an appropriate K factor:

$$K = \max(1, \frac{d_1 \cdot l_x \cdot l_y}{(f^L + f^R + f^T + f^B) \cdot \Delta t}) \tag{4}$$

where l_x, l_y are the distances between the grid cells in the x,y directions. The outflow flux is multiplied by K:

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

We calculate ΔV water height change with adding f_{out} output and f_{in} input flow values in each (x,y) cell:

$$\begin{aligned} \Delta V(x,y) &= \Delta t \cdot (\sum f_{in} - \sum f_{out}) = \\ &= \Delta t \cdot (f^R_{t+\Delta t}(x-1,y) + f^T_{t+\Delta t}(x,y-1) + \\ &\quad f^L_{t+\Delta t}(x+1,y) + f^B_{t+\Delta t}(x,y+1) - \\ &\quad \sum_{i=L,T,R,B} f^i_{t+\Delta t}(x,y). \end{aligned} \tag{6}$$

Then, we update the water height in the current (x,y) cell:

$$d_2 = d_2(x,y) + \frac{\Delta V(x,y)}{l_x l_y}. \tag{7}$$

Using outflow flux values, we can calculate the $\overrightarrow{\mathbf{v}}$ velocity field that needed to calculate hydraulic erosion and deposition. The calculation of the x component is:

$$\begin{aligned}\Delta W_x = \frac{1}{2}(f^R(x-1,y) - f^L(x,y) + \\ f^R(x,y) - f^L(x+1,y))\end{aligned} \tag{8}$$

The y component is calculated in a similar way. Since we know the velocity vector, we can calculate C water sediment transport capacity that represents how much sediment can be transported in a cell. In the original model C is calculated as:

$$C(x,y) = K_c \cdot \sin(\alpha(x,y)|\overrightarrow{\mathbf{v}}(x,y)|) \tag{9}$$

where K_c is a global simulation parameter controlling sediment capacity, $\sin(\alpha)$ is the local tilt angle, and $\overrightarrow{\mathbf{v}}(x,y)$ is the water flow vector in the cell. This empirical formula erodes terrain proportional to the surface slope. To allow some erosion at nearly flat areas, there is a lower limit for $\sin(\alpha(x,y))$. Fluid erosion in real-world is highly dependent on the water depth. Deep sea floors are practically never eroded, even if there is a stream in the water, because water flow is slower in deeper water levels, although sediment capacity of deeper water is larger due to the larger water volume. On the contrary, a relatively shallow river always dissolves the terrain at the bottom, because water is flowing faster and has direct effect on the bottom. To simulate this, we modified equation (9) by introducing l_{max} limiting function:

$$C(x,y) = K_c \cdot \sin(\alpha(x,y))|\overrightarrow{\mathbf{v}}(x,y)| \cdot l_{max}(d_1(x,y)) \tag{10}$$

$l_{max}(x)$ is a ramp function that is defined by the following:

$$l_{max}(x) = \begin{cases} 0, x \leq 0 \\ 1, x \geq K_{dmax} \\ 1 - (K_{dmax} - x)/K_{dmax}, 0 < x < K_{dmax} \end{cases}$$

where K_{dmax} is a global simulation parameter controlling the maximum erosion depth. This function scales down the fluid erosion effects by the water depth, so the erosion will occur only in shallower areas, forcing the simulation to dispose sediment at deeper water areas, just like in real world. This produces much more natural looking deep water sea and lake floors than the original model. Moreover, we included true 3D collision between water and terrain surface:

$$C(x,y) = K_c \cdot (-\overrightarrow{\mathbf{N}}(x,y) \cdot \overrightarrow{\mathbf{V}}) \cdot |\overrightarrow{\mathbf{v}}(x,y)| \cdot l_{max}(d_1(x,y)) \tag{11}$$

where $N(x,y)$ is the terrain surface normal at point (x,y) and $\overrightarrow{\mathbf{V}}$ is the 3D water flow vector calculated from the surface tangent and 2D velocity vector $\overrightarrow{\mathbf{v}}$. This modification erodes more soil if the water collides with the surface in angles closer to perpendicular. With our model, we observed some ripples on sea floors similar to sand ripples on real-world seashores.

At this point, there is a decision by using the C capacity. If transported sediment s_t in cell (x,y) is smaller than C, then we dissolve some soil in water:

$$b_{t+\Delta t} = b_t - \Delta t \cdot R_t(x,y) \cdot K_s(C - s_t), \tag{12a}$$
$$s_1 = s_t + \Delta t \cdot R_t(x,y) \cdot K_s(C - s_t), \tag{12b}$$
$$d_3 = d_2 + \Delta t \cdot R_t(x,y) \cdot K_s(C - s_t), \tag{12c}$$

where K_s is the global coefficient. Otherwise, if $C < s_t$, then we dispose some of the transported sediment in a similar way:

$$b_{t+\Delta t} = b_t + \Delta t \cdot K_d(s_t - C), \tag{13a}$$
$$s_1 = s_t - \Delta t \cdot K_d(s_t - C), \tag{13b}$$
$$d_3 = d_2 - \Delta t \cdot K_d(s_t - C), \tag{13c}$$

where K_d is a global parameter controlling deposition speed. Equations 12c and 13c were added to improve long-term stability. Originally, the $\Delta t \cdot K_d(s_t - C)$ suspended sediment amount was subtracted from the terrain heigth b_t without adding it to d_t water height. The overall height of the water surface is defined as $b_t + d_t$ at a given t time. Thus, after the subtraction the overall water surface height was decreased by the amount of the sediment suspended by the water itself, caused water to disappear with the sediment, ignoring the fact that suspended amount is still in the water in fluid form. This caused some unwanted feedback to the water flow simulation from the sedimentation process and causes regular ripples on the water surface in the long run. With our modification, this behavior can be eliminated.

To prevent negative water heights in equation 12c, we clamped the dissolved amount to water height in cell (x,y).

There is one more improvement in the model. In nature, moving sediment becomes softer by the time. To imitate this, we slowly lower the $R(x,y)$ local hardness coefficient of the terrain when some soil is disposed:

$$R_{t+\Delta t}(x,y) = max(R_{min}, R_t(x,y) - \Delta t \cdot K_h K_s(s_t - C)) \tag{14}$$

where R_{min} is the lower limit of hardness, K_h is a global coefficient controlling the sediment softening. The next step in the model is to move dissolved sediment along the water using $\overrightarrow{\mathbf{v}} = (u,v)$:

$$s_{t+\Delta t}(x,y) = s_1(x - u \cdot \Delta t, y - v \cdot \Delta t) \tag{15}$$

If point $(x - u \cdot \Delta t, y - v \cdot \Delta t)$ is not on the grid, the model uses linear interpolation between the four closest grid points. In the last step, we simulate water evaporation:

$$d_{t+\Delta t}(x,y) = d_3(x,y) \cdot (1 - K_e \Delta t). \tag{16}$$

In nature, the evaporation has a negligible effect, but in our model it is important because the scene would fill up with

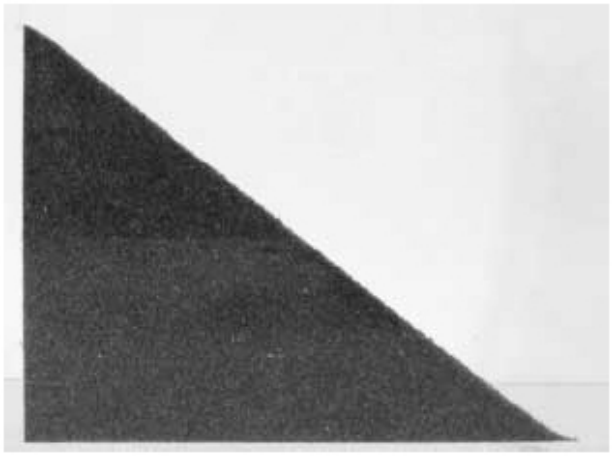


Figure 3: Demonstration of talus angle: photograph of real red sand dumped between flat plexi plates. The material has near linear slope. [13].

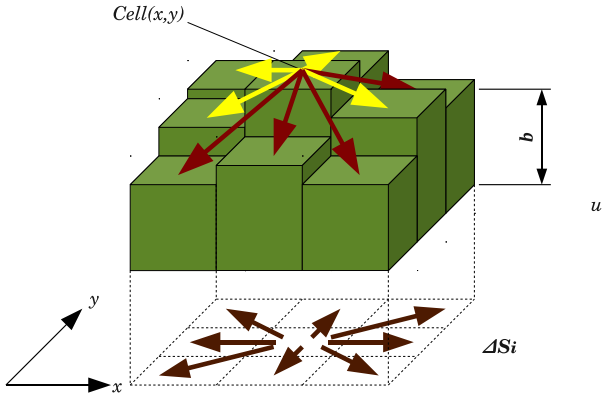


Figure 4: Virtual pipes of thermal erosion. Brown arrows: virtual pipes. Red arrows: soil movement from the $Cell(x,y)$ to the neighbors whose are lower than the talus angle. Yellow arrows: soil movement is inhibited to cells which height is above the talus angle.

water by the time if the water was not removed from the system.

Our thermal erosion model is based on [9]. The original method was designed neither to run on parallel architecture nor in real-time, but with some improvements these problems can be solved. Let's denote the terrain height of the current (x,y) cell by b and its eight neighbors by $b^i, i = 1,2,...,8$. Let's denote the height difference between the current cell and its lowest neighbor by $H = \max\{b - b^i, i = 1,...,8\}$. The area of each of the cells is a and the volume to be moved is $\Delta S = a \cdot H/2$. This is the maximum, otherwise the algorithm will oscillate. To handle local $R(x,y)$ terrain hardness in cell (x,y), we have extended this formula:

$$\Delta S_{t+\Delta t} = a \cdot \Delta t \cdot K_t \cdot R_t(x,y) \cdot H/2 \qquad (17)$$

where K_t is a global coefficient. Then we move this amount to the lower neighbors proportionally if the so called talus angle is larger than that the value determined by material viscosity. The talus angle is an important static parameter of solid granular materials without cohesion between grain particles. To measure this parameter, we should dump the material slowly to a flat surface between two transparent plates. The material will form a slope with an angle, which is a maximum that the given material can reach. Above this angle, the material starts moving to lower levels, making the slope lower. When slope angle reaches this critical angle, then the material does not move anymore [13]. See figure 3.

Let's denote the distance between two cells by d and talus angle by $\alpha = tan((b - b^i)/d)$. Let's denote the set of neighbors that are lying lower than the current element under the talus angle by $A = \{b^i, b - b^i < 0 \wedge tan(\alpha) > (R(x,y) * K_a + K_i), i = 1,...8\}$, where K_a and K_i are global simulation parameters controlling minimum talus angle dependence on $R(x,y)$ local hardness factor. Each element in A will get part of the volume ΔS_i proportional to its height difference:

$$\Delta S_i = \Delta S \frac{b^i}{\sum_{\forall b^k \in A} b^k}. \qquad (18)$$

In contrary to the original model, we do not move ΔS_i volumes directly to cells in set A because this would introduce data write dependency that we wanted to avoid for easy parallel execution. Similarly to fluid flow simulation, we put these quantities into eight virtual pipes carrying material to the neighbors of this cell (see Figure 4), then a separate simulation step updates the terrain height for each cell by summarizing the incoming material flow from their neighbors.

With this modification we can easily execute the algorithm in parallel and integrate it to the fluid-based erosion simulation:

1. Water incrementation due to rain or water sources.
2. Flow simulation using shallow-water model. Computation of velocity field and water height changes.
3. **Soil flow calculation with outflow in virtual pipes of thermal erosion model**.
4. Simulation of erosion-deposition process.
5. Transportation of suspended sediment by the velocity field.
6. **Thermal erosion material amount calculation**.
7. Water evaporation.

With these improvements, we can execute the two erosion models in conjunction with each other. Steep walls of riverbeds carved by hydraulic erosion will start to fall

down due to thermal erosion. And vice versa, the material eroded by thermal erosion will be dissolved and transported by the running water.

3 GPU Implementation and Visualization

In our implementation, the cell structure is represented by 2D 4-channel floating-point texture layers stacked upon each other, attached to a single framebuffer object. Texels of the same position in texture layers form one cell, containing all the simulation variables associated with it. With two such framebuffers we calculate one iteration using one of these buffers as an input and the other as an output (ping-pong) [12]. After the iteration we swap these buffers and start over. The iteration process is implemented in a single fragment shader that runs in three passes on a full-size quad rendered over the entire framebuffer, writing the output textures using the multiple render target feature. The implementation exploits linear interpolation and edge wrapping, which are basic features of the graphics hardware.

The texture buffers can be used directly to visualize the terrain height values to offset triangle vertices of the rendered mesh in the y direction. For the sake of simplicity we used a regular grid mesh to render the terrain surface, but heightfield texture can also be used in advanced terrain rendering methods.

Water is rendered in a similar way. The water surface is a simple uniform grid mesh too, rendered with y offset with the water height and a small $-\Delta y$ constant. Due to z-buffering this makes water surface invisible where water height does not exceed Δy. Water surface has a simple fragment shader that calculates simple sky reflections and alpha transparency.

4 Results

We have used regular and randomly generated terrains to test our method. Regular terrains were utilized to test our parallel erosion methods (see Figure 5 and Figure 6) and see they are running without problems. As we can see, thermal erosion makes the terrain material spread around until the critical talus angle is reached, then the material stops moving. Hydraulic erosion carves the surface creating deep valleys.

Random terrains were utilized to test the model in natural-like scenarios. Random terrains were generated with the the well-known Diamond-square algorithm (see Figure 7a). We implemented it non-recursively to make it possible to generate terrains with similar patterns at different resolutions. We used Gaussian distributed random numbers to generate middle point heights in the algorithm instead of regularly distributed ones, which makes the result more realistic.

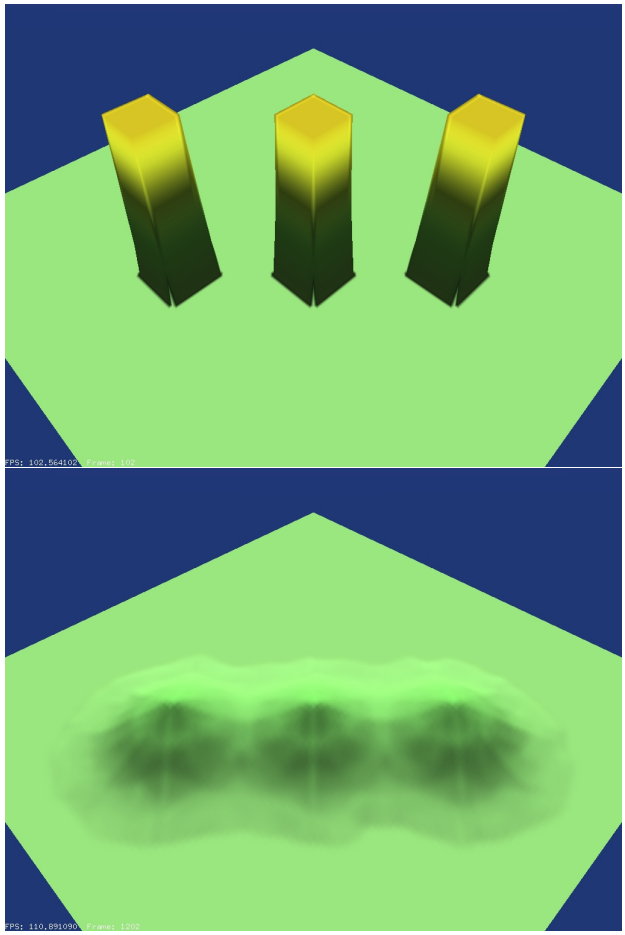


Figure 5: Effects of our parallel thermal erosion method on a regular height field after 1000 iterations.

Our erosion model uses $R(x,y)$ local hardness coefficient in every (x,y) cell. This is a value in range 0..1, that represents the resistance of the soil against in a given cell. Smaller values are representing harder material and vice versa. We generate this coefficient array by copying and scaling the terrain heightfield, adding some random noise to every cell, and applying a global Gaussian filter to the array. The idea behind this was twofold. First, in the nature, the terrain material at higher levels is usually more resistant to erosion than the lower parts. Second, real-world terrain material is never homogeneous, the consistency varies from location to location. With this simple method we can improve the realism of the generated terrain. The surface will be more irregular, similarly to real-world terrains where the material inhomogenity influences the erosion at different locations (see 1).

Figure 7b shows the effects of the improved hydraulic erosion model on the aforementioned random terrain. The hydraulic erosion carves deep grooves into the surface that rarely occurs in nature. This is why we included thermal erosion in the model.

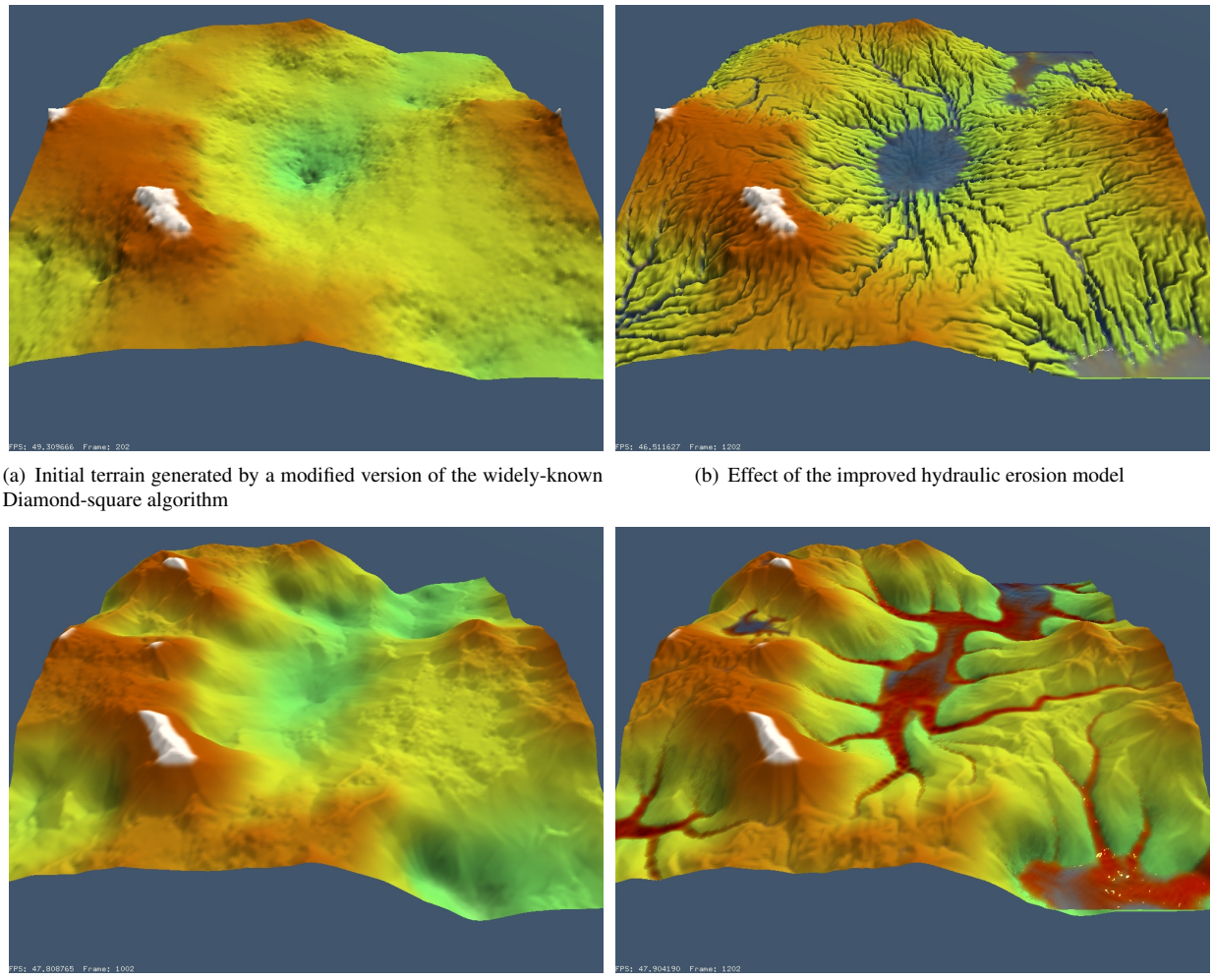
Figure 7c shows the effects of the parallel thermal erosion process on the random terrain. The material at too

Terrain Size	Speed (ms/iteration)				Ratio (GPU/CPU)	
	CPU1	CPU2	GPU1	GPU2	conf. 1	conf. 2
128x128	82.8	25.7	2.57	0.295	32.21	87.1
256x256	325	102.4	9.92	1.01	32.76	101.3
512x512	1289.1	412.6	39.22	3.835	32.86	107.5
1024x1014	5165.6	1647	160.15	15.47	32.54	106.5

Table 1: Performance results

Symbol and Description		Range	Value	Symbol and Description		Range	Value
Δt	Time increment	$[0;0.05]$	0.02	K_s	Soil suspension rate	$[0.1;2]$	0.5
K_r	Rain rate	$[0;0.05]$	0.012	K_d	Sediment deposition rate	$[0.1;3]$	1
K_e	Water evaporation rate	$[0;0.05]$	0.015	K_h	Sediment softening rate	$[0;10]$	5
A	Virtual pipe cross section area	$[0.1;60]$	20	K_{dmax}	Maximal erosion depth	$[0;40]$	10
g	Gravity	$[0.1;20]$	9.81	K_a	Talus angle tangent coeff.	$[0;1]$	0.8
K_c	Sediment capacity	$[0.1;3]$	1	K_i	Talus angle tangent bias	$[0;1]$	0.1
K_t	Thermal erosion rate	$[0;3]$	0.15				

Table 2: Allowed ranges and typical values of global simulation parameters used in our simulator.



(a) Initial terrain generated by a modified version of the widely-known Diamond-square algorithm

(b) Effect of the improved hydraulic erosion model

(c) Effect of our thermal erosion model

(d) Effect of the combined erosion method

Figure 7: Effects of different erosion methods on random terrain after 1000 iterations. The reddish color indicates sediment in the water.

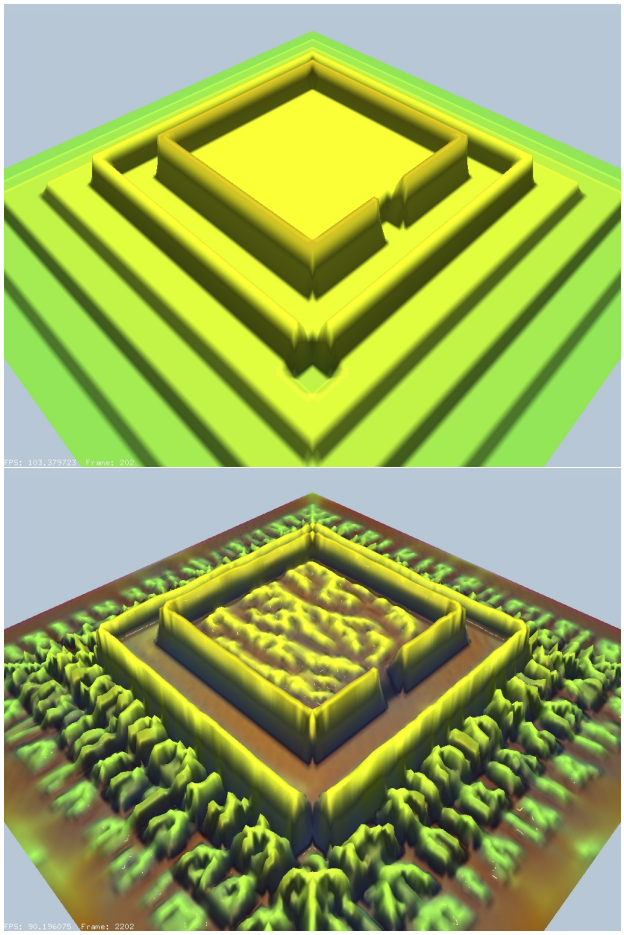


Figure 6: Effects of our hydraulic erosion method on a regular height field after 1000 iterations.

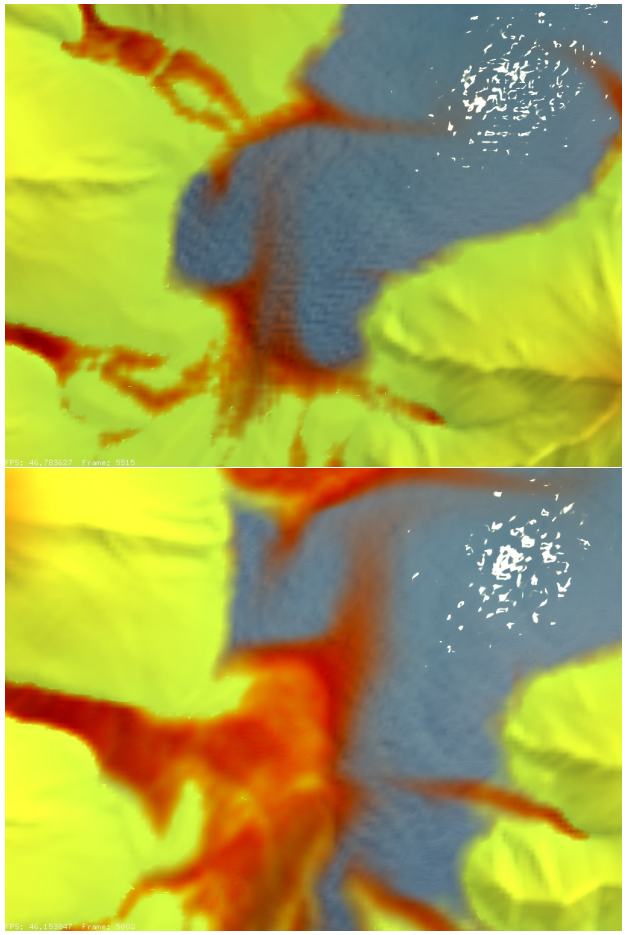


Figure 8: Elimination of the oscillation in the original erosion model.

steep slopes is moving towards the lower regions.

Figure 7d demonstrates the effects of our combined erosion algorithm on a random terrain. We can observe that the riverbeds and valleys are v-shaped which is more realistic than in the original erosion model.

In our framework the following ranges were defined for global simulation parameters:

Figure 8 shows the effect of oscillations on the water surface appearing at long simulations (upper image). These ripples were successfully removed by some modification applied to the model (lower image). See equations 12c and 13c.

Figure 9 illustrates the effect of erosion depth limiting introduced in eqation 10. On the upper image the original model carves riverbeds unrealistically deep. The improved version produces more credible riverbeds (lower picture).

We tested the performance of our model on the two following hardware configurations.

1. CPU1: AMD Athlon XP 3.2 GHz
   GPU1:ATI Radeon HD3650 AGP

2. CPU2: Intel Core2 Q9550 2.83GHz
   GPU2: ATI Radeon HD4870

We implemented the erosion model both on CPU and on GPU to measure the difference between the two architecture. The results are summarized in Table 1. We can see that the speedup gained by utilizing GPU is very high, we reached 30 to 100 times faster execution. On the second configuration, the ratio is higher, showing that the GPUs are developing faster than the ”traditional” serial CPUs due to their more scalable architecture. Using GPU implementation, our algorithm can run at interactive speed even on larger terrains.

5 Conclusions

We proposed an erosion model that can be executed on massively parallel architectures like graphics processors. The method combines and improves two algorithms to simulate hydraulic and thermal erosion in conjunction with each other. The original hydraulic erosion method is extended to be more versatile and stable. The thermal erosion model is a parallel redesign of an earlier work. It makes it able to run on parallel architectures and be integrated with the fluid erosion model. Our method gener-

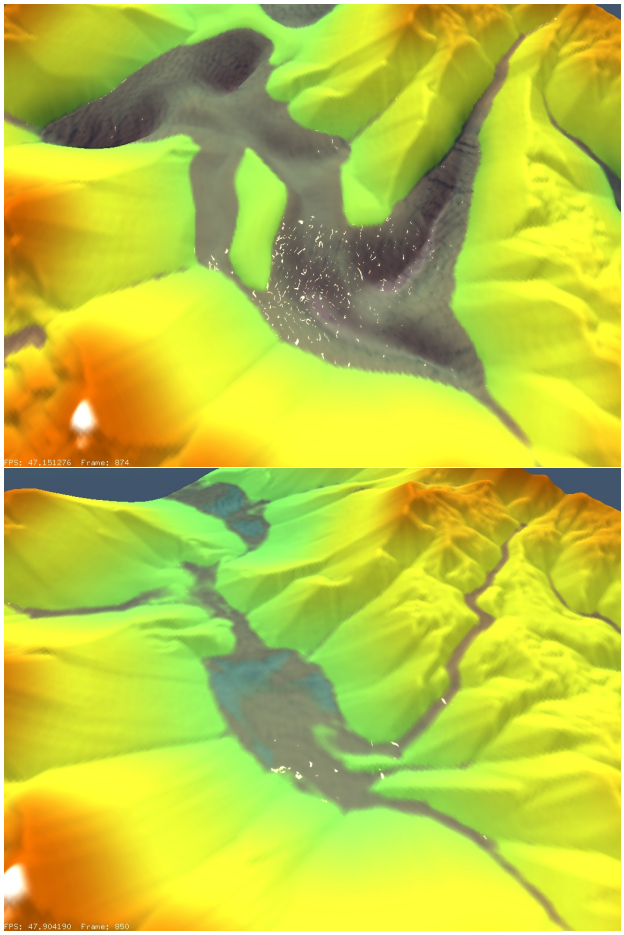


Figure 9: Effects of hydraulic erosion depth limit.

ates more realistic results while still running at interactive speeds.

Acknowledgements

This work has been supported by OTKA K-719922 and by TÁMOP-4.2.1/B-09/1/KMR-2010-0002.

References

[1] Bedřich Beneš. Real-time erosion using shallowwater simulation. *The 4th Workshop on Virtual Reality Interactions and Physical Simulation - Vriphys'07*, pages 43–50, 2007.

[2] Bedřich Beneš and Rafael Forsbach. Layered data representation for visual simulation of terrain erosion. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, page 80, Washington, DC, USA, 2001. IEEE Computer Society.

[3] Bedřich Beneš, Václav Těšínský, Jan Hornyš, and Sanjiv K. Bhatia. Hydraulic erosion: Research articles. *Comput. Animat. Virtual Worlds*, 17(2):99–108, 2006.

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

[5] E. Galin, P. Poulin (editors, B. Neidhold, M. Wacker, and O. Deussen. Interactive physically based fluid and erosion simulation.

[6] Mark Harris. Fast fluid dynamics simulation on the gpu. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[7] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, volume 24 of *SIGGRAPH '90*, pages 49–57, New York, NY, USA, September 1990. ACM.

[8] 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, Pacific Graphics 2007, November, 2007*, pages 47–56, Maui, Hawaii, Etats-Unis, November 2007. IEEE.

[9] Forest Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains, 1989.

[10] James F. O'Brien and Jessica Kate Hodgins. Dynamic simulation of splashing fluids. In *Proceedings of the Computer Animation*, pages 198–, Washington, DC, USA, 1995. IEEE Computer Society.

[11] László Szirmay-Kalos and László Szécsi. *General Purpose Computing on Graphics Processing Units*. MondAT kiadó, 2011.

[12] László Szirmay-Kalos, László Szécsi, and Mateu Sbert. *GPU-Based Techniques for Global Illumination Effects*. Synthesis Lectures on Computer Graphics and Animation. Morgan & Claypool Publishers, 2008.

[13] Péter Vankó. Izgalmas mérések a mérnök-fizikus hallgatói laboratóriumban. *Fizikai szemle, BME TTK*, 9:307, 2006.

[14] Enhua Wu, Youquan Liu, and Xuehui Liu. An improved study of real-time fluid simulation on gpu. *Department of Computer Science, University of Manchester, UK. Since*, 15:139–146, 2004.