# Fast Hydraulic Erosion Simulation and Visualization on GPU

Xing Mei
CASIA-LIAMA/NLPR, Beijing, China
xmei@nlpr.ia.ac.cn

Philippe Decaudin
INRIA-Evasion, Grenoble, France
& CASIA-LIAMA, Beijing, China
philippe.decaudin@imag.fr

Bao-Gang Hu
CASIA-LIAMA/NLPR, Beijing, China
hubg@nlpr.ia.ac.cn


**Figure 1. Illustration of our erosion simulation on a "PG"-shaped mountain. (a) The initial terrain. (b) Terrain is being eroded by rainfall. The dissolved soil (denoted by green color) is transported by the water flow (blue). (c) The eroded terrain and the deposited sediment (red) after the rainfall and during the evaporation.**

## Abstract

*Natural mountains and valleys are gradually eroded by rainfall and river flows. Physically-based modeling of this complex phenomenon is a major concern in producing realistic synthesized terrains. However, despite some recent improvements, existing algorithms are still computationally expensive, leading to a time-consuming process fairly impractical for terrain designers and 3D artists.*

*In this paper, we present a new method to model the hydraulic erosion phenomenon which runs at interactive rates on today's computers. The method is based on the velocity field of the running water, which is created with an efficient shallow-water fluid model. The velocity field is used to calculate the erosion and deposition process, and the sediment transportation process. The method has been carefully designed to be implemented totally on GPU, and thus takes full advantage of the parallelism of current graphics hardware. Results from experiments demonstrate that the proposed method is effective and efficient. It can create realistic erosion effects by rainfall and river flows, and produce fast simulation results for terrains with large sizes.*

## 1. Introduction

Natural terrains exhibit a lot of complex shapes such as valleys, ridges, sand ripples and crests. These geomorphological structures are mostly determined by the erosion phenomenon: soil is dissolved and transported by the water flow, sand is carried away by the wind, and stones are decomposed into small blocks by the temperature. Erosion simulation has always been an important research topic in landscape design [5, 8], since it greatly enhances the realism of the synthesized terrains.

We focus on hydraulic erosion, which is the predominant cause of erosion for most outdoor landscapes [9]. It describes how the soil on the terrain is dissolved, transported and deposited by the running water, as described in Section 3. Many algorithms have been proposed to simulate this complex process [2, 19, 20, 24]. Although they produce realistic eroded terrains, they are usually computationally expensive, leading to computation time varying from minutes to hours on common computers. This is inconvenient for terrain designers, who wish to view the dynamic simulation process and edit the parameters interactively. Furthermore, we notice that the topographical changes of the terrain also affect the distribution of the water flows. In-

cluding the erosion effect in fluid simulation will greatly enhance the realism of the running water on terrains, such as rivers, brooks and even lavas. Less work has been done on this problem [6, 7, 17].

In this paper, we propose a fast method for hydraulic erosion simulation, in order to get immediate simulation results for interactive applications. Our first concern is to design an algorithm that can take full advantage of the high parallel computing power offered by today's graphics processing units (GPU). We represent terrain and water surfaces as height fields on 2D regular grids. Then we adapt a shallow water model from [22] to update the water surface and the fluid velocity field. To the best of our knowledge, this model has not been used for erosion simulation before. Based on the velocity field, we calculate the amount of the eroded soil and the deposited sediment in each cell of the grids. Following the erosion-deposition process, the suspended sediment is transported by the velocity field. With careful design, the values in the grid cells can be updated in parallel at each step, which fits well into the general computing framework of the current graphics hardware [11, 23]. Therefore the simulation model is implemented on GPU as a multi-pass algorithm, and the simulated results can be directly used for visualization. Relying on GPU's parallelism and flexible programmability, our method provides significantly faster results than previous methods while preserving the realism. This allows us to create fluvial eroded terrain and view all the important features for erosion in real-time, such as water movement and catchment, evaporation, soil dissolving and sedimentation.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we present our model in detail. The multi-pass implementation on GPU is given in Section 4. In Section 5, we present the experiment results and performances for several examples. Finally we conclude our work in Section 6.

## 2. Related Work

Many erosion algorithms have been proposed in the field of soil science and computer graphics [9, 28]. Some early methods adapt the fractal rules with *ad hoc* changes to produce eroded terrains from scratch. Kelly *et al*. [14] created a drainage network of the rivers with geological data, and then generated fractal terrain around this network. Prusinkiewicz and Hammel [24] integrated rivers into their mountain models by applying midpoint displacement with predefined water basins.

More algorithms try to simulate the erosion process with physically based models and modify the relief of the already-existing terrains. Musgrave *et al*. [19] described a general framework for physically based erosion modeling. Some material is dissolved and transported by the water flow, and finally deposited at another location. The water movement is determined by local gradient of the terrain in a simple diffusion algorithm. Roudier *et al*. [25] proposed applying local geological parameters to the erosion process. Fluvial erosion, gravity creep and chemical dissolution were included in their models. Nagashima [20] presented a similar model to generate valleys and canyons on fractal terrains with a predefined river network. Beneš *et al*. [2] first included evaporation in the simulation, and divided the process into several independent steps.

The methods mentioned above are based on the simple diffusion model, which is not accurate enough to describe the water movement and sediment transportation. Water movement and sediment transportation are closely related to the velocity field of the running water. Therefore fluid simulation methods were integrated into the erosion process by several researchers. Chiba *et al*. [4] was the first to propose simulating ridges and valleys with particle systems. The velocity of the water particles is dynamically updated with gravity and local inclination angles. The terrain surface is then modified with the accumulated energy when water particles collide with the ground. This work was followed and improved in [27]. Beneš *et al*. [3] employed traditional Navier-Stokes Equations (NSEs) to model the process in a 3D regular grid. The main disadvantage with these algorithms is that they are still computationally expensive, especially the fluid simulation part.

Recently Neidbold *et al*. [21] proposed a simplified Newtonian physics model for velocity computation on 2D Euclidean grids. The velocity value at each cell is accelerated by the gravity and the local tilt angle. When water is transported to a new grid cell, the velocity vector at the destination cell is mixed with the transported water by an empirical formula. Although it is reported that their algorithm runs at 4 fps on a $256 \times 256$ grid (on a 2.4 GHz Pentium IV PC), it is still limited for interactive applications. The velocity computation and the erosion process in their algorithm need dependent data processing, which can not be easily parallelized for further optimization.

With rapid development in hardware, more and more people are turning to the GPU to solve general computing problems (GPGPU), especially in the field of fluid simulation [23]. We aim at finding a flow model which can be mapped efficiently to the GPU and work well with the erosion process. 2D NSEs have been efficiently solved on GPU in [10, 29]. However, extending the framework to solve 3D NSEs is still limited to a small grid size since 3D texture is not well supported in the computation [16] and therefore not suitable for our work. Li *et al*. [15] implemented Lattice Boltzmann Method on graphics hardware to simulate a variety of fluid problems, and the performance is also limited by the usage of 3D textures. In our application, the water flow on the terrain is best described by 2D Shallow Water Equa-

tions (SWEs). Kass *et al*. [13] solved SWEs with an implicit numerical method. The method requires many iterations over the 2D grid at each time step, which is not efficient for our large size terrains. One recent work by Beneš [1] employed this method for real-time erosion simulation, and the performance is reported to be 5 fps for a $300 \times 300$ grid. In 1995, O'Brien *et al*. [22] proposed a virtual pipe model to simulate the shallow water. Water is transported through the pipe by the hydrostatic pressure difference between the neighboring grid cells. In Section 3 and 4, we show that the pipe model can be parallelized on GPU [1] and extended to handle erosion simulation, and therefore forms the basis of our algorithm.

## 3. Hydraulic Erosion Model

The proposed hydraulic erosion model is decomposed into five steps. Before going into the model, we first describe the necessary quantities and notations for the simulation process (cf. Figure 2):
- terrain height $b$
- water height $d$
- suspended sediment amount $s$
- the outflow flux $\mathbf{f} = (f^L, f^R, f^T, f^B)$
- velocity vector $\vec{\mathbf{v}} = (u, v)$

These values are stored for each cell of the 2D grid and kept up-to-date during the simulation. Therefore we need several layers of 2D arrays for data storage. This layered data structure has been extensively used in previous methods [2, 21].

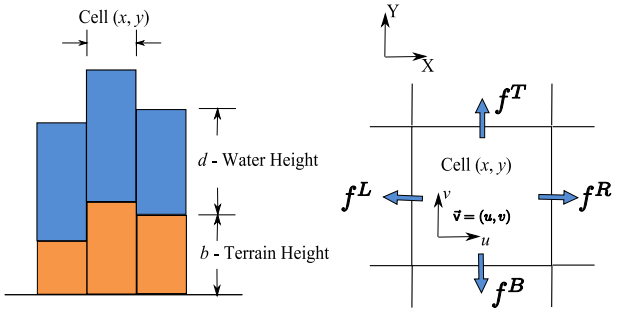


**Figure 2. Our data structure and the neighboring information**

[1] Recently, we found an anterior paper by Maes *et al* [18] that has implemented an improved pipe model on GPU in a similar way. They focus on fluid simulation and incorporate a particle system (CPU-GPU mixed implementation) to handle breaking waves. We focus on extending this model to simulate the erosion process. We also present more details about the GPU implementation of the pipe model, especially about the scaling process and the limitations, which are not covered in their paper.

The five steps of our simulation are:

1. Water increases due to rainfall or river sources.
2. Flow is simulated with the shallow-water model. Then the velocity field and the water surface are updated.
3. Erosion-deposition process is computed with the velocity field.
4. Suspended sediment is transported by the velocity field.
5. Water decreases due to evaporation.

The simulation process iterates over the five steps to get gradually changing results. Each step of the simulation takes in data from previous steps and updates these values using their governing equations. Let $b_t$, $d_t$, $s_t$, $\mathbf{f}_t$ and $\vec{\mathbf{v}}_t$ be the data at given time $t$, $\Delta t$ be the iteration time step, we describe how to update these grid values for the time $t + \Delta t$ within the five steps. Since some of the data are updated in two or three steps, we use subscripts $1, 2, 3...$ to distinguish the intermediate updated value from the final data used for next iteration, such as $d_t$ (water height at time $t$), $d_1$, $d_2$ (intermediate water height) and $d_{t+\Delta t}$ (final water height at $t + \Delta t$).

### 3.1. Water Increment

New water appears on the terrain due to two major effects: rainfall and river sources. For both types, we need to specify the location, the radius and the water intensity (the amount of water arriving during $\Delta t$). For river sources, the location of the sources is fixed. For rainfall, each raindrop falls down on the terrain with a random distribution. Let $r_t(x, y)$ be the water arriving at cell $(x, y)$ per time unit, water height is updated by a simple addition:

$$d_1(x, y) = d_t(x, y) + \Delta t \cdot r_t(x, y) \qquad (1)$$

Note that $d_1$ is an intermediate water height, which will be changed in the following steps.

### 3.2. Flow Simulation

We first describe how to compute the outflow flux $\mathbf{f}$ based on the pipe model, and then update the water surface $d$ and the velocity field $\vec{\mathbf{v}}$ with $\mathbf{f}$.

#### 3.2.1 Outflow Flux Computation

As stated in the related work, we adapt the pipe model from [22] to simulate the water movement. In the pipe model, cell $(x, y)$ exchanges water with its neighboring cells through the virtual pipes. A "flow velocity" is maintained for each

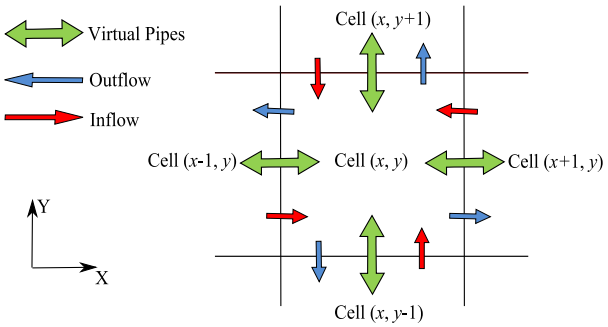


**Figure 3. Pipe model notations. Each cell is connected to its four neighbors through virtual pipes.**

pipe. To avoid confusion with the velocity field $\vec{\mathbf{v}}$, we use the term "flux" instead of flow velocity. At each step, the flux is accelerated by the hydrostatic pressure difference between the two cells connected by the pipe. Then the water height in cell $(x, y)$ is updated by accumulating all the flux values from all the virtual pipes. If the updated water height is negative, all pipes that are removing fluid from the cell collect some water back from neighbors. However, this scaling back process can not be easily mapped to GPU because dependent data processing on the 2D grid is involved in the process, which is not affordable on GPU.

We solve this issue by limiting the outflow flux from one cell with its water amount. We store for each cell only the outflow flux value $\mathbf{f} = (f^L, f^R, f^T, f^B)$, as shown in Figure 3. $f^L$ is the outflow flux from cell $(x, y)$ to its left neighbor $(x-1, y)$, and $f^R, f^T, f^B$ are defined similarly. If the sum of the outflow flux exceeds the water amount of the cell, $\mathbf{f}$ will be scaled down by a factor $K$ to avoid negative updated water height.

We calculate $f^L$ for cell $(x, y)$ as follows:

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

where $A$ is the cross-sectional area of the pipe, $g$ is the acceleration due to gravity, $l$ is the length of the virtual pipe, and $\Delta h^L(x, y)$ is the height difference between cell $(x, y)$ and its left neighbor $(x-1, y)$:

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

For detail derivation of eqn. (2), we refer the reader to O'Brien's original paper [22]. $f^R, f^T, f^B$ are computed in a similar way. The scaling factor $K$ for the outflow flux is then given as:

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

where $l_X$ and $l_Y$ is the distance between grid points in the $X$ and $Y$ directions. So the outflow flux $\mathbf{f} = (f^L, f^R, f^T, f^B)$ is scaled by K:

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

#### 3.2.2 Water Surface and Velocity Field Update

The water height is updated with the new outflow flux field by collecting the inflow flux $f_{in}$ from neighbor cells, and sending the outflow flux $f_{out}$ away from the current cell. For cell $(x, y)$, the net volume change for the water is:

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

The water height in each cell is then updated as:

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

Next, the velocity field $\vec{\mathbf{v}}$ is necessary for the erosion-deposition model (Section 3.3) and the sediment transportation (Section 3.4). Only the horizontal velocity is considered in our study. $\vec{\mathbf{v}}$ can be easily calculated from the outflow flux. The average amount of water that passes through cell $(x, y)$ per unit time in the $X$ direction can be expressed with eqn. (8):

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

And $\Delta W_X$ can also be computed by the velocity component $u$ in the $X$ direction:

$$l_Y \cdot \bar{d} \cdot u = \Delta W_X \tag{9}$$

where $\bar{d} = \frac{d_1 + d_2}{2}$ is the average water height in cell $(x, y)$ during the first two steps. Then $u_{t+\Delta t}$ is computed using eqn. (8) and (9). In a similar way we get the $v$ component of $\vec{\mathbf{v}}$ in the $Y$ direction.

For any fluid simulation method, boundary conditions should be taken into consideration. Since the flow is simulated on a 2D grid, we assume no water can flow out of the grid ("no slip" boundary condition). In the pipe model, we specify the outflow flux on boundary cells to satisfy the conditions. For cells $(x, 0)$ on the left boundary, the outflow flux to the left neighbor $f^L(x, 0)$ should be set to zero. Similar rules apply for the other boundaries.

The pipe model can be seen as as an explicit method for shallow-water equations. The numerical stability of the

solution is determined by the time step $\Delta t$. The maximum time step is restricted by various factors, such as the Reynolds number, the cell size, the boundary conditions. And the scaling step might also introduce instability to the system: if the outflow flux of one cell is scaled by the factor $K$, the outflow flux of its neighbors should be adjusted correspondingly to avoid a numerical error in flow rate, which is not considered in the method. For our model, the new value of one cell is calculated from its four neighbors. Therefore this limitation can be approximately expressed with the CFL condition: $\Delta t \cdot u \leq l_X, \Delta t \cdot v \leq l_Y$. When the size of the grid increases ($l_X$, $l_Y$ decrease), we should decrease the time step proportionally to get stable simulation results.

Although all the equations introduced in the pipe model can be easily extended to handle all the eight neighbors of the current cell, we find in practice that the four von Neumann neighbors can produce satisfactory results.

With the updated velocity field, we proceed in the following steps.

### 3.3. Erosion and Deposition

When water flows over the terrain, some soil will be eroded and transported by the water, and some suspended sediment will be deposited on the ground. This erosion-deposition process is mostly determined by the sediment transport capacity of the flow [9]. This quantity is restricted by many factors, and many prediction models have been proposed in soil science. We employ a simplified empirical equation from [12] to simulate the process. The sediment transport capacity $C$ for the water flow in the cell $(x, y)$ is calculated as:

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

where $K_c$ is a sediment capacity constant, $\alpha(x, y)$ is the local tilt angle and $\vec{\mathbf{v}}$ is the velocity value. From eqn. (10), we can see that $C$ is related to the terrain geometry and the flow velocity. Note that there is one limitation about the computation of $C$: for very flat terrains where the $\alpha$ value approaches zero, $C$ will be very small, which means the water flow will pick up very little soil from the ground and the erosion effect is indistinctive. This problem can be alleviated by limiting the $\alpha$ value with a user-specified minimum threshold. Then we compare $C$ with the suspended sediment amount $s$ to decide the erosion-deposition process. If $C > s_t$, some soil is dissolved into the water and added to the suspended sediment:

$$b_{t+\Delta t} = b_t - K_s(C - s_t) \tag{11a}$$
$$s_1 = s_t + K_s(C - s_t) \tag{11b}$$

where $K_s$ is a dissolving constant. If $C \leq s_t$, some suspended sediment is deposited:

$$b_{t+\Delta t} = b_t + K_d(s_t - C) \tag{12a}$$
$$s_1 = s_t - K_d(s_t - C) \tag{12b}$$

where $K_d$ is a deposition constant. The amount of the suspended sediment $s_1$ will be updated again in the transportation step.

### 3.4. Sediment Transportation

After the erosion-deposition step, the updated suspended sediment is transported with the flow velocity field $\vec{\mathbf{v}}$. This process can be described with the advection equation:

$$\frac{\partial s}{\partial t} + (\vec{\mathbf{v}} \cdot \nabla s) = 0 \tag{13}$$

We use the semi-Lagrangian advection method introduced to graphics by Stam [26] to solve the advection equation. We determine the new value for the current cell $(x, y)$ with the velocity field by taking an Euler step backward in time:

$$s_{t+\Delta t}(x, y) = s_1(x - u \cdot \Delta t, y - v \cdot \Delta t) \tag{14}$$

If the position $(x - u \cdot \Delta t, y - v \cdot \Delta t)$ is not on the grid, we get the $s$ value by using interpolation of the four nearest neighbors. Since the semi-Lagrangian approach is unconditionally stable, there will be no stability problem for this step.

### 3.5. Evaporation

Finally, some water are evaporated into the air due to the environmental temperature. We assume the temperature to be constant during the simulation. And the evaporation model can be represented as:

$$d_{t+\Delta t}(x, y) = d_2(x, y) \cdot (1 - K_e \cdot \Delta t) \tag{15}$$

where $K_e$ is an evaporation constant.

After the evaporation step, we get all the data at time $t + \Delta t$, which is used for the next iteration. We present a compact summary of all the steps as follows:

1. $d_1 \leftarrow WaterIncrement(d_t)$;
2. $(d_2, \mathbf{f}_{t+\Delta_t}, \vec{\mathbf{v}}_{t+\Delta t}) \leftarrow FlowSimulation(d_1, b_t, \mathbf{f}_t)$;
3. $(b_{t+\Delta t}, s_1) \leftarrow ErosionDeposition(\vec{\mathbf{v}}_{t+\Delta t}, b_t, s_t)$;
4. $s_{t+\Delta t} \leftarrow SedimentTransport(s_1, \vec{\mathbf{v}}_{t+\Delta t})$;
5. $d_{t+\Delta t} \leftarrow Evaporation(d_2)$

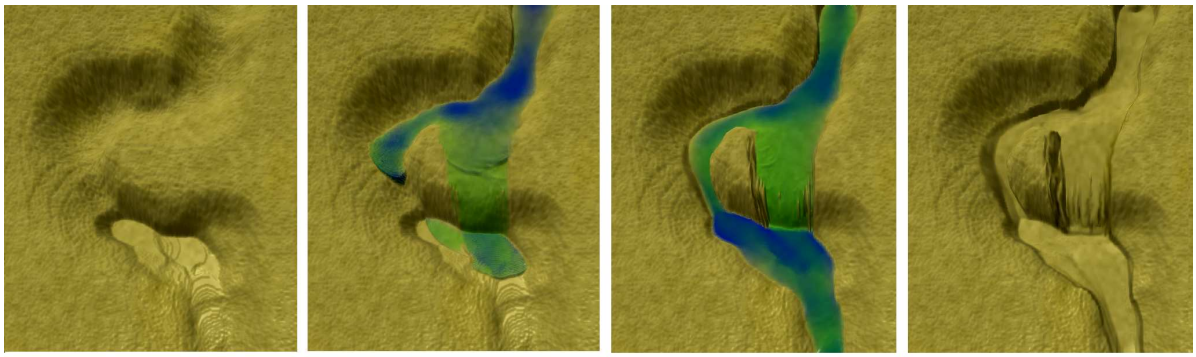


**Figure 4. Water flow erodes the riverbed (b) and creates a new path (c, d).**

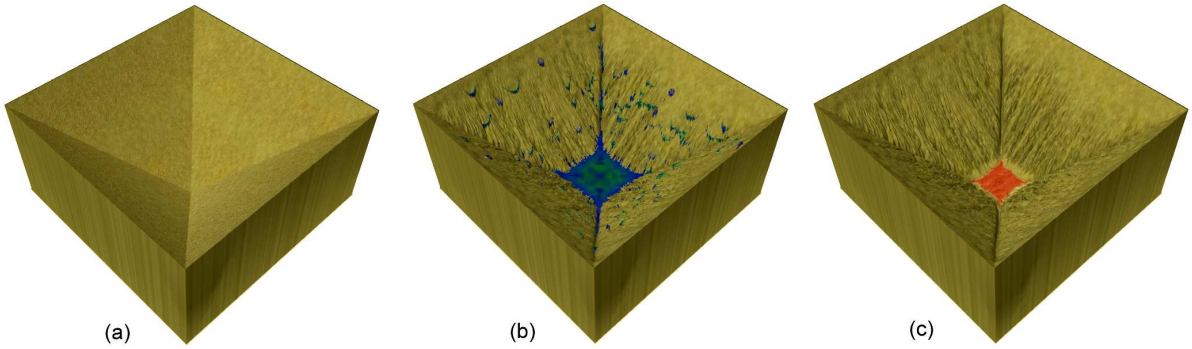


**Figure 5. Erosion and deposition in a basin (b). Center of the basin is flattened by the cumulated sediment (red) after the evaporation (c).**

## 4. Simulation and Visualization on GPU

Harris [11] provided a basic framework for GPGPU programs. The input arrays are packed into 2D textures and sent to the GPU memory. Then a quad is drawn oriented parallel to the image plane. For each pixel in the quad, a fragment is generated by the hardware rasterizer and processed by a customized fragment shader. The fragment shader takes care of the computation for each pixel, which is the kernel of the process. Finally, the output values are written into a texture or a render target, which can be used for the computation in the next pass. Complex GPGPU programs may need many passes through the pipeline to get the final results. Since fragment processors are highly paralleled and optimized, the computation is usually much faster than an equivalent CPU implementation, and scales well with the grid resolution. Furthermore, there is no need to transfer data between the host memory and the graphics memory, which is especially suitable for our simulation and visualization work.

We first pack all the basic quantities into three 2D textures: $b$, $d$, and $s$ are stored in different channels of one texture $T_1$, $\mathbf{f}$ has four components, therefore stored in one independent texture $T_2$, and the velocity field $\vec{\mathbf{v}}$ in texture $T_3$. Following the steps described in Section 3, we implement the model as a multi-pass process on GPU: For step $1, 3, 4$ and $5$, only one pass is needed respectively. For step $2$, we should first update the outflow flux in $T_2$, then update the water height in $T_1$ and the velocity field in $T_3$, so three passes are necessary. In total seven passes are needed for the simulation.

As described in the pipe model, the boundary cells require separate treatment from the interior cells of the grid. Harris [10] suggested using a boundary fragment program and rendering line primitives over the edges of the viewport. We avoid this extra draw call by judging in the fragment shader whether the cell being processed is on the boundary or not. Since the flow control statements have lower penalties on the latest GPUs and only short branches are needed for the judgement, the unnecessary fragment operations on interior cells do not exhibit obvious performance loss in our experiments.

After the data is updated by the simulation process, the data texture $T_1$ (containing terrain and water height) can be

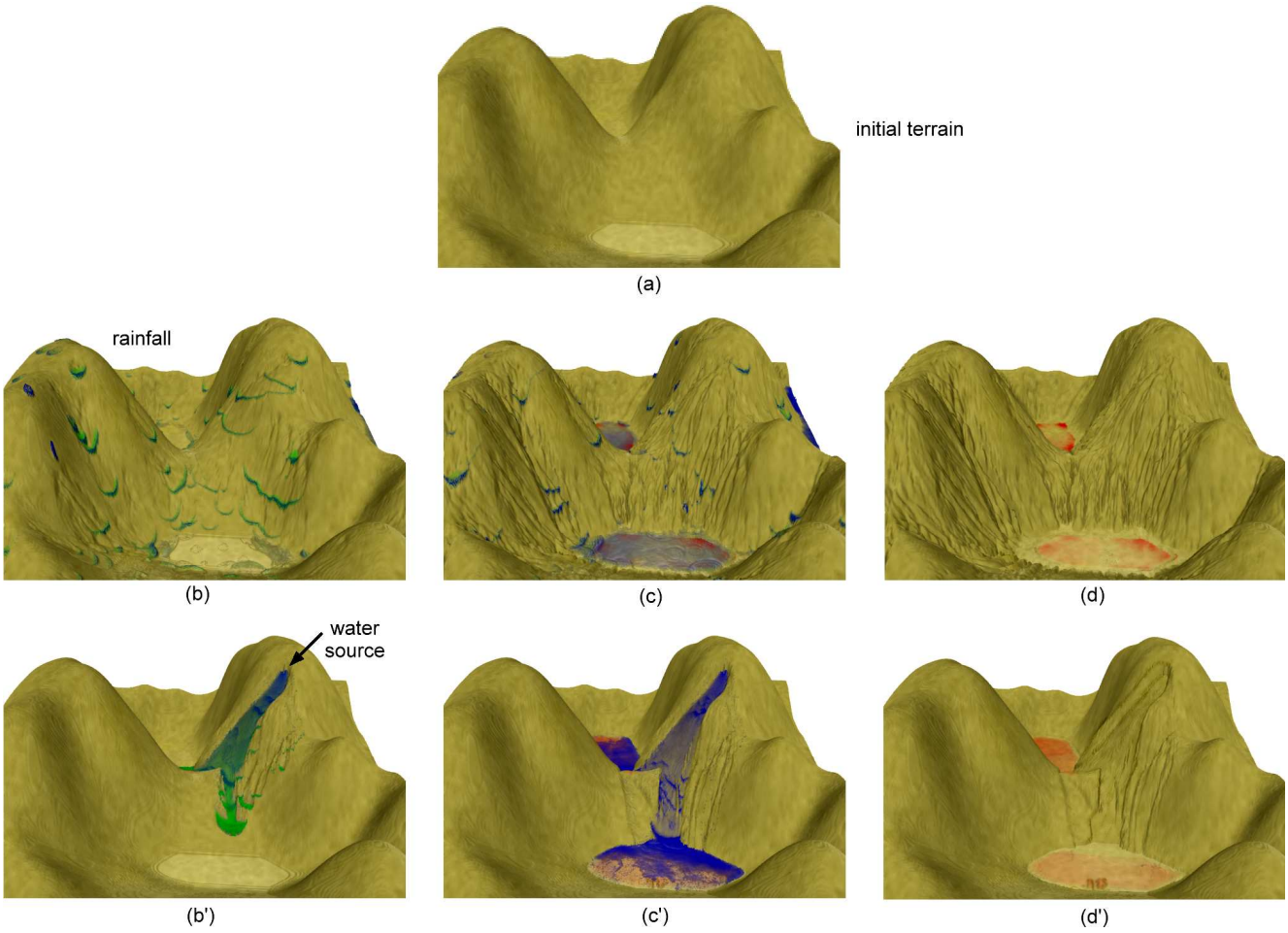


**Figure 6. Mountain eroded by rainfall (b, c, d) and water source (b', c', d') respectively.**

used directly for the visualization. $T_1$ is sent to the vertex shader as a vertex texture to displace the height of a predefined grid mesh. For efficiency reasons we draw only one height field for the whole scene, which combines the terrain and the water surfaces. The height displacement $h$ for grid cell $(x, y)$ is given as $h(x, y) = b(x, y) + d(x, y)$, which is easily obtained with one texture fetch from $T_1$. Then a fragment shader computes the final color for the cell. To get the diffuse color, the water color is blended with the terrain color; its transparency varies with the water height $d$. Phong lighting model is employed for per pixel lighting. Different specular coefficients are used for terrain and water surfaces respectively.

## 5. Experimental Results

We present the experimental results and the performances. The test platform is a Pentium IV 2.40GHz PC equipped with a nVIDIA GeForce 8800 GTX graphics card. Although the experiments were run on a DirectX10 class graphics card, our algorithm can be implemented on any Shader Model 3.0 GPUs, since no DirectX10 features were used in the implementation. For all the examples, we use the color green to indicate the amount of the suspended sediment in the water flow, and color red for the amount of the deposited sediment on the ground, which helps us to get a better view of the important features in the simulation process, such as erosion, deposition and sediment transportation. See video at http://evasion.imag.fr/Publications/2007/MDH07.

The first test scene is a dry river channel on a slope. A water source is put into the channel at an upriver position. A detailed view of the channel during the simulation is presented in Figure 4 and the accompanying video shows the running simulation (sequence 2). Some water follows the channel, eroding the original riverbed. Some water flows out of the channel and erodes part of the river bank, creating a new path for the subsequent downward flow. The eroded soil enters the suspended sediment layer, and gets

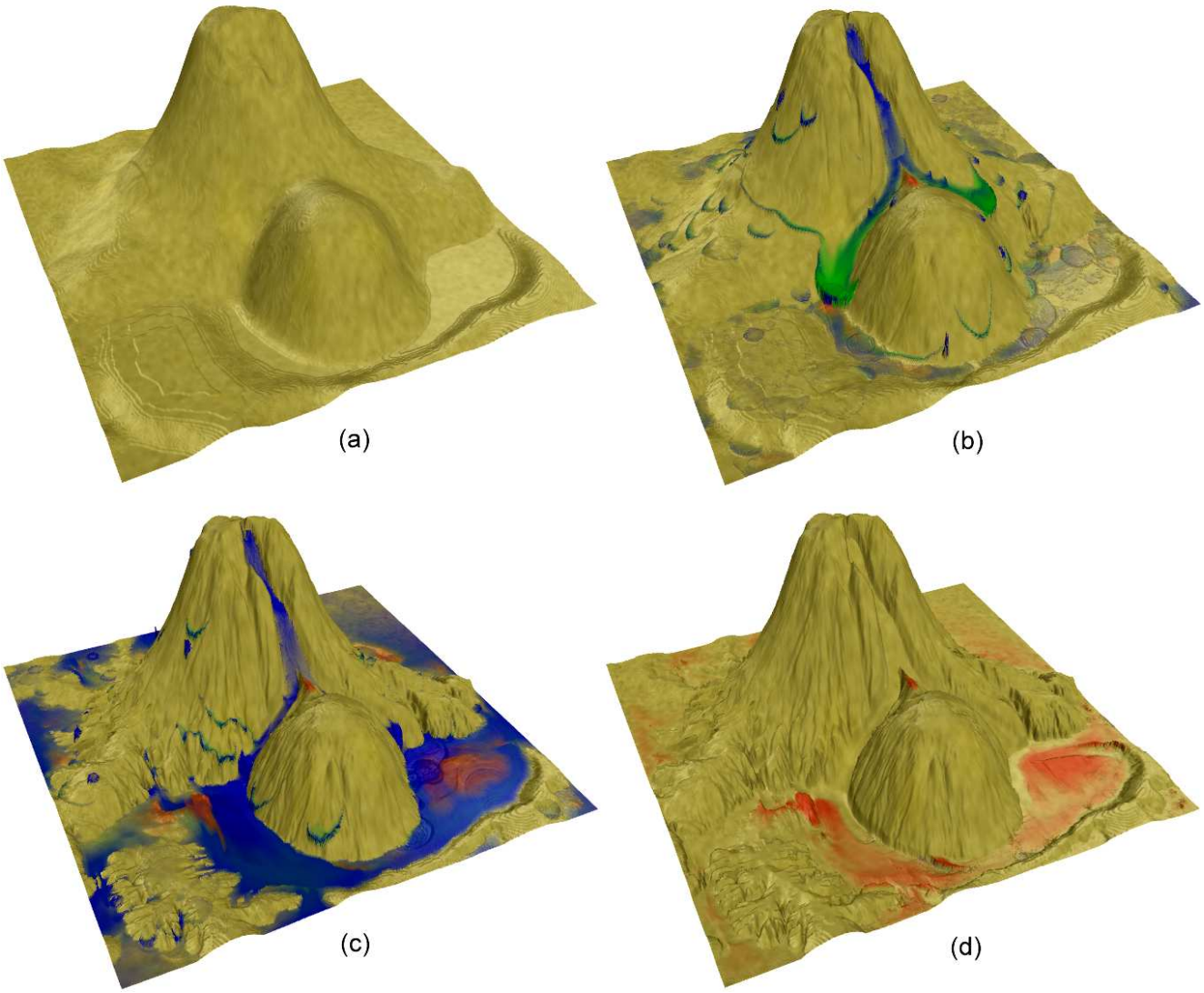


**Figure 7. An erosion process that combines the rainfall and a river source.**

transported by the water flow. This phenomenon is widely observed in nature.

Erosion effect by rainfall on an artificial height field has been illustrated in Figure 1. We provide another example in Figure 5 to show the effect of the deposition process. The test scene is an artificial basin. Four slopes intersect at the bottom of the basin. Rainfall brings soil down from all four slopes and deposits it at the bottom. After plenty of rainfall and the evaporation process, the slopes are highly eroded and the bottom is flattened by the cumulated sediment (denoted in red, cf Figure 5 (c)).

Next we show in Figure 6 a mountain scene which is eroded by rainfall and a fixed river source respectively. The initial terrain is smooth. Rainfall creates interesting gullies on the hillsides, while the fixed source creates a stream bed along the flow directions. In both cases, water and the deposited sediment are accumulated at the bottom, forming two lakes.

A final example which combines the rainfall and a river source in one simulation process is given in Figure 7. Different patterns created by rainfall and the river source, and all the important features for the erosion process can be observed in this example. The accompanying video records the dynamic simulation processes for all the examples presented in this section.

The performances of our algorithm are presented in Table 1. We define one cycle to be one simulation process plus one visualization process. For each cycle, we record the Simulation Time (ST for short, measured in milliseconds), Visualization Time (VT, in milliseconds), and Cycle

| Grid Size | ST (ms) | VT (ms) | CT (ms) | CPS |
|---|---|---|---|---|
| $256 \times 256$ ($\Delta t = 0.002$) | 1.29 | 1.28 | 2.48 | 403 |
| $512 \times 512$ ($\Delta t = 0.001$) | 2.38 | 2.97 | 5.38 | 186 |
| $1024 \times 1024$ ($\Delta t = 0.0005$) | 6.65 | 10.31 | 16.95 | 59 |
| $2048 \times 2048$ ($\Delta t = 0.00025$) | 22.88 | 40.00 | 62.50 | 16 |
| $4096 \times 4096$ ($\Delta t = 0.000125$) | 91.22 | 155.89 | 248.47 | 4 |

**Table 1. Performance results for different grid sizes. ST: average simulation time, VT: avg. visualization time, CT: avg. cycle time (1 cycle = 1 simulation + 1 visualization), CPS: avg. number of cycles per second.**

Time (CT, in milliseconds). CT is also used to provide the CPS (cycles per second). As the computation cost is mostly determined by the grid size, all the examples produce similar performance results. So we only provide the results for the final mountain example in Figure 7. The grid size varies from $256 \times 256$ to $4096 \times 4096$. From Table 1, we can see that our algorithm runs interactively even on a large $2048 \times 2048$ size mountain. Both the simulation process and the visualization process scale well with the changing grid size. The time cost varies linearly with the number of the cells (plus some slight overhead at the low end), which is coherent with the algorithmic complexity of the method. For grid size larger than $512 \times 512$, the visualization process occupies most of the execution time. This is due to the large number of texture fetches in vertex shader (one fetch for each cell). There is still room for further improvement: the simulation stage and the visualization stage are independent of each other, and the grid size for each stage can be different, especially for large size terrains up to $4096 \times 4096$. We can iterate over the simulation process for more than once before each visualization. The grid size for the visualization step can be smaller than the simulation size and the details of the terrain appearance are to be enhanced with normal maps.

## 6. Conclusions and Future Work

We have presented an efficient method for fast hydraulic erosion simulation and visualization. Relying on a well adapted shallow water model, our method produces realistic eroded terrains in a physically based way. The proposed method also has the good property of independent data processing, which allows its efficient implementation as a multi-pass process on GPU. We can handle large size terrains and view the erosion process at interactive rates, which are usually difficult for previous techniques. However there are still several problems to be further studied about our method: as mentioned in Section 3, the erosion model we employed has some difficulties in simulating the erosion process on very flat terrains or terrains with caves. The stability of the systems is restricted by the time step and the scaling process of the fluid solver.

As a future work, based on this method, we would like to develop an interactive system which allows terrain designers to edit the terrain surface, the water sources and the physical parameters interactively and see the feedback in real time. We would also like to include other erosive processes in our algorithm, such as thermal weathering and wind erosion. Simulating the erosion effects on non-height-field objects is also an interesting topic we would like to explore in the future.

## Acknowledgements

The authors would like to thank Jamie Wither for proofreading. We would also like to thank the anonymous reviewers for their valuable comments. Xing Mei is supported by LIAMA NSFC 60073007 and by a grant from the European Community under the Marie-Curie project VISITOR MEST-CT-2004-8270. Philippe Decaudin is supported by a grant from the European Community under the Marie-Curie project REVPE MOIF-CT-2006-22230.

## References

[1] B. Beneš. Real-time erosion using shallow water simulation. In *VRIPHYS'07: 4th Workshop in Virtual Reality Interactions and Physical Simulation*, 2007.

[2] B. Beneš and R. Forsbach. Visual simulation of hydraulic erosion. *Journal of WSCG*, 10:79–86, 2002.

[3] B. Beneš, V. Těšínský, J. Hornyš, and S. K. Bhatia. Hydraulic erosion. *Computer Animation and Virtual Worlds*, 17(2):99–108, 2006.

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

[5] O. Deussen and B. Lintermann. *Digital Design of Nature*. Springer, Berlin, Germany, 2005.

[6] J. Dorsey, A. Edelman, H. W. Jensen, J. Legakis, and H. K. Pedersen. Modeling and rendering of weathered stone. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 225–234, 1999.

[7] J. Dorsey, H. K. Pedersen, and P. Hanrahan. Flow and changes in appearance. In *SIGGRAPH'96: Proceedings of*

*the 23th annual conference on computer graphics and interactive techniques*, pages 411–420, 1996.

[8] D. Ebert, K. Musgrave, P. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers, San Francisco, USA, 2003.

[9] R. S. Harmon and W. W. Doe III. *Landscape Erosion and Evolution Modeling*. Kluwer Academic/Plenum Publishers, New York, USA, 2001.

[10] M. Harris. Fast fluid dynamics simulation on the gpu. In *GPU Gems*, chapter 38, pages 637–666. Addison-Wesley Professional, 2004.

[11] M. Harris. Mapping computational concepts to gpus. In *GPU Gems 2*, chapter 31, pages 493–508. Addison-Wesley Professional, 2005.

[12] P. Y. Julien and D. B. Simons. Sediment transport capacity of overland flow. *Transactions of the ASAE*, 28(3):755–762, 1985.

[13] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 49–57, 1990.

[14] A. D. Kelly, M. C. Malin, and G. M. Nielson. Terrain simulation using a model of stream erosion. In *SIGGRAPH'88: Proceedings of the 15th annual conference on computer graphics and interactive techniques*, pages 263–268, 1988.

[15] W. Li, X. Wei, and A. Kaufman. Implementing lattice boltzmann computation on graphics hardware. *The Visual Computer*, 19(7-8):444–456, 2003.

[16] Y. Liu, X. Liu, and E. Wu. Real-time 3d fluid simulation on gpu with complex obstacles. In *Proceedings of Pacific Graphics'04*, pages 247–256, 2004.

[17] Y. Liu, H. Zhu, X. Liu, and E. Wu. Real-time simulation of physically based on-surface flow. *The Visual Computer*, 21(8-10):727–734, 2005.

[18] M. M. Maes, T. Fujimoto, and N. Chiba. Efficient animation of water flow on irregular terrains. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 107–115, 2006.

[19] F. K. Musgrave, C. E. Klob, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH'89: Proceedings of the 16th annual conference on computer graphics and interactive techniques*, pages 41–50, 1989.

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

[21] B. Neidhold, M. Wacker, and O. Deussen. Interative physically based fluid and erosion simulation. In *Eurographics Workshop on Natural Phenomena'05*, pages 25–32, 2005.

[22] J. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *Proceedings of Computer Animation'95*, pages 198–205, 1995.

[23] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[24] P. Prusinkiewicz and M. Hammel. A fractal model of mountains with rivers. In *Proceedings of Graphics Interface'93*, pages 128–137, 1993.

[25] P. Roudier, B. Peroche, and M. Perrin. Landscapes synthesis achieved through erosion and deposition process simulation. *Computer Graphics Forum*, 12(3):375–383, 1993.

[26] J. Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[27] B. Sutherland and J. Keyser. Particle-based enhancement of terrain data. In *Siggraph'06 Research Poster*, page 96, 2006.

[28] G. Valette, S. Prévost, L. Lucas, and J. Léonard. Soda project: A simulation of soil surface degradation by rainfall. *Computer&Graphics*, 30(4):494–506, 2006.

[29] E. Wu, Y. Liu, and X. Liu. An improved study of real-time fluid simulation on gpu. *Journal of Computer Animation and Virtual World*, 15(3-4):139–146, 2004.