Interactive hydraulic erosion using CUDA

Richard Bézin, Alexandre Peyrat, Benoît Crespin, Olivier Terraz, Xavier Skapin, Philippe Meseure

Interactive hydraulic erosion using CUDA

Richard Bezin[1], Alexandre Peyrat[1], Benoit Crespin[1], Olivier Terraz[1], Xavier Skapin[2], and Philippe Meseure[2]

[1] XLIM - UMR 6172 - CNRS, France
University of Limoges
[2] University of Poitiers

Abstract. This paper presents a method to simulate hydraulic erosion and sedimentation on a terrain represented by a triangular mesh in real-time. Our method achieves interactive performances by dynamically displacing vertices using CUDA following physically-inspired principles; the mesh is generated in a preprocessing step to avoid degenerated cases in highly deformed areas.

1 Introduction

Fluvial processes study how landforms are created by rivers and streams through erosion, sediment transport and deposit over time. In this paper we're interested in reproducing the hydraulic erosion process due to water, neglecting chemical dissolution and other marginal processes. Our goal is to automatically obtain visually realistic terrains eroded by water flows, for example by deepening a valley due to *stream erosion*. Since hydraulic erosion is mainly due to maximal flood levels rather than normal activity [1], dynamic control of the rate of flow is desirable. The main contribution of our approach is its ability to produce plausible results at an interactive framerate even for very large scenes, by implementing our method within the CUDA framework [2]. As a consequence, some choices are inherent to this implementation, such as using a particle system to represent the water flow, which performs well with highly parallelized architectures, although real-time fluid simulations may rely on other discretization models. A snapshot of our application is shown on Fig. 3.

Another consequence is that some processes may be neglected or simplified in order to maintain interactive rates, if their visual contribution does not appear significant in the final result. As stated earlier, chemical dissolution for instance does not have a significant impact on the visually perceptible details due to hydraulic erosion [1]. Other processes such as sediments acting erosively on the surface would be too computationally expensive for any existing method if we were to represent each rolling and sliding grain in the flow.

This paper is organized as follows. We first describe recent works in the literature addressing the problem of hydraulic erosion in section 2. Section 3 describes our hydraulic erosion model and its implementation with CUDA; an adaptive heightfield generation method is also presented to reduce the amount of triangles in a preprocessing step. Finally, results obtained with our application are shown in section 4.

2 Previous Work

Recent works focus on terrain *modifications* by hydraulic processes, by explicitly representing fluid motion and interactions between the fluid and the initial terrain model. Representations based on voxels [3, 4] divide the terrain into a set of small 3D cubes. Each voxel stores some information about the amount of material it contains, its geological resistance to hydraulic erosion, etc. In [5], visually plausible concave surfaces are created through erosion using discrete surface curvature obtained from the voxel grid. Navier-Stokes equations can be solved by different numerical methods. A semi-Lagrangian method is used in [6] to run on simple scenes in real-time. The capacity C for a particle to transport sediment depends on its velocity: if C is above a predefined threshold, it can erode the terrain by sweeping sediment away, otherwise it may deposit sediment. Other recent approaches take into account the local slope of the terrain and evaporation [7], or different types of material [8]. Smoothed Particle Hydrodynamics (SPH) are used in [9] to represent fluid motion, with the terrain modelled as a triangulated heightfield. Interactions between the fluid and the terrain are computed through the use of static particles sampling the triangulated mesh, which exchange data with SPH particles: erosion is achieved by transferring material from static particles to SPH particles, whereas sediment deposition is the inverse process, and terrain modifications are then obtained by vertical extrusion of the corresponding triangles. Simulations conducted with this approach run in interactive time with up to 25K particles thanks to a GPU implementation.

However, even if results obtained with these methods are visually spectacular, their high computation and memory costs make them either prohibitive for real-time applications or limited to simple scenes. Therefore our goal is to get the same quality with more approximations. A good example is the fluid discretization model: SPH is a well-known method for accurately simulating small-scale details, but other particle-based fluid models may provide better results to enforce incompressibility [10]. By relying on a similar philosophy, we aim at approximating erosion and fluid-terrain exchanges in order to generate a visually realistic but not necessarily precise simulation.

3 Hydraulic Erosion Model

We use a particle-based fluid model, relying on the n-body gravitational simulation provided within CUDA where particle-particle interactions are computed with a DEM method [2]. We also choose to represent the terrain as a non-regular triangular mesh; triangles are obviously the best choice for real-time simulations, but implies some kind of level-of-details process to avoid degenerated triangles in strongly eroded areas, as presented in section 3.3. The main loop of our simulation method can be summarized as:

1. Update fluid particles positions
2. Calculate collisions and accumulate erosion on triangles

3. Calculate sediment deposition from fluid particles and accumulate sedimentation on triangles
4. Update vertices' positions and remove unnecessary particles

We describe in the following how to compute collisions and update vertices based on erosion and sedimentation processes.

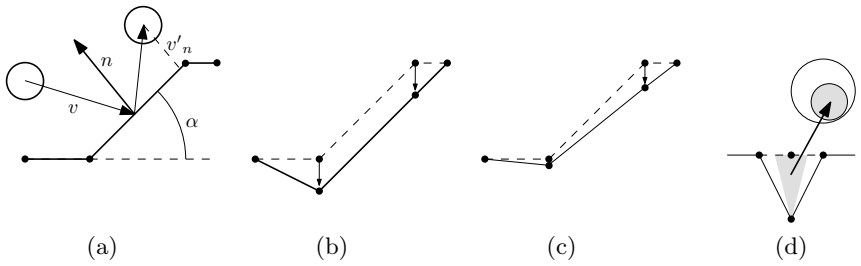

Fig. 1. (a) Collision handling between a particle and a segment in 2D **(b)** Uniform vertical displacement of endpoints **(c)** Weighted displacement depending on endpoints' height values **(d)** 2D representation of the local volume eroded from one vertex (in gray) and transferred to the nearest fluid particle

3.1 Particle/Triangle Collision

In order to obtain a realistic fluid flow, we need to detect and handle collisions between particles and triangles. Our method inspired by [11] computes segment/triangle intersections. In our case this segment is given by the successive positions of a fluid particle p_t and p_{t+1}, and we keep the closest triangle to p_t. Velocity $\boldsymbol{v}$ is decomposed in $\boldsymbol{v_t}$ and $\boldsymbol{v_n}$, its tangential and normal velocity relative to the normal vector $\boldsymbol{n}$ respectively; after collision, normal velocity $\boldsymbol{v'_n}$ is reduced to simulate damping and force particles to slip over the surface as shown on Fig. 1a. When colliding with the triangle, the particle erodes it by a certain amount C (as in [7]):

$$C = K_c \cdot \sin\alpha \cdot |\boldsymbol{v}| \tag{1}$$

where K_c is the erosion rate and α is the triangle's tilt angle, meaning that horizontal areas are less eroded. To apply erosion we then need to displace its vertices downwards: if vertices are uniformly displaced by the same amount $C/3$, the slope of the triangle is preserved but unrealistic results may appear (see Fig. 1b). We choose to apply a larger weight to the highest vertex, which produces more gentle slopes as shown on Fig. 1c. The final displacement applied to a vertex is the sum of displacements computed from each triangle to which it belongs.

3.2 Erosion and sedimentation

Displacements due to hydraulic erosion are applied to update vertices positions as described above, which in turn generate sediments that will be transported by the flow. Actually displacements are not taken into account at each simulation step, instead we choose to accumulate displacements during n frames before applying erosion every n steps only. This implies that triangle-particle collisions are not as accurate as possible, however re-hashing the entire mesh at each step would be too expensive even on GPU. Another problem when applying erosion at each step would be that only small-sized sediment particles are generated, although sediments are usually composed of mineral rocks or sands of various sizes, depending on the composition of the river bed [1]. Parameter n, as well as all the other parameters of our model, can be modified during the simulation.

We derive the size of a sediment eroded from the bed at a given vertex from the approximated volume of the cone that is "lost" due to erosion as shown on Fig. 1d. This spherical sediment of size s is then transported by the flow, and will eventually be deposited back to the bed. Implementing this transport/deposition process is complicated, since it depends on many parameters, including sediment size and shape, mineral composition and flow velocity. The well-known Hjulström curve for example [1] rules that, for the same flow velocity, small-sized grains are transported for a longer period of time. Because computing exact flow velocity is a time-consuming task, we simplify the problem by considering the *expected transport time* ε of a sediment:

$$\varepsilon = \lceil \varepsilon_M (1 - \frac{s^x}{s_M^x}) \rceil \tag{2}$$

where ε_M and s_M represent the maximal transport time and the maximal size of a sediment particle respectively, and $x = 1/s_M$.

Newly created sediment grains are advected with the flow by attaching them to the nearest fluid particle. At each subsequent step of the simulation their transport time ε is decremented: when it reaches zero the sediment falls from its fluid particle down to the river bed. Eq. 2 ensures that near-zero size sediments will be transported during ε_M steps before being deposited, whereas large sediments of size s_M will only have one step in a flow with the same velocity. If the fluid particle slows down, the sediment will be transported on a shorter distance, which is consistent with Hjulström's rule.

Sedimentation means transferring the volume of the sediment to the nearest triangle below its attached fluid particle. As with erosion, this volume is distributed among the three vertices but those are displaced upwards such that the gained volume is equivalent, using different weights (a similar rule can be found in [9]). Fig. 3 shows the application of our algorithms with eroded or sedimented triangles colored in purple. The iterative nature of our model is visible on Fig. 4.

3.3 Implementation with CUDA and Adaptive Heightfield Generation

Our implementation runs entirely on GPU and can generate particles through emitters or deactivate particles if they evaporate or flow out of the simulation. Particles are stored in a hashing grid to accelerate neighborhood queries. An auxiliary hashing grid storing triangles is used for efficient particle-triangle collision detection [12], and is updated every n steps when erosion or sedimentation is applied (*ie* when vertices are displaced).

The goal with GPU programming is to reduce concurrent access to read or write data. If we consider that a particle may erode or sediment only one triangle at each step, our main problem is to efficiently compute which particles contribute to the displacement applied to each vertex. We give details about this computation below, vertices and particles displacements described in sections 3.1 and 3.2 being easier to implement. To handle particle-triangle interactions, we extend the work presented in [13] based on four successive steps implemented through atomic operations to ensure maximal performances; Table 1 describes an example where particles respectively indexed 0 and 2 collide with triangles defined by vertices $(4, 6, 8)$ and $(2, 6, 8)$. Collisions are stored in two arrays `indexParticle` and `indexVertex` (step **a**), then these arrays are sorted regarding `indexVertex` (step **b**). Two arrays `cellStart` and `cellEnd` are filled using the first and last index of each vertex stored in `indexVertex` (step **c**); for example, vertex 6 starts at index 2 and stops at index 4 in `indexVertex`. Finally particles which contribute to displace each vertex v are obtained by considering the sorted array `indexParticle` from `cellStart[v]` to `cellEnd[v]-1` (step **d**); for vertex 6 these are particles 0 and 2, stored in `indexParticle` at indices 2 and 3.

	0	1	2	3	4	5	6	7	8	...
indexParticle	0	0	0	1	1	1	2	2	2	...
indexVertex	4	6	8				2	6	8	...

(a)

	0	1	2	3	4	5	6	7	8	...
indexVertex	2	4	6	6	8	8				...
indexParticle	2	0	0	2	0	2				...

(b)

	0	1	2	3	4	5	6	7	8	...
cellStart			0		1		2		4	...
cellEnd			1		2		4		6	...

(c)

vertex	0	1	2	3	4	5	6	7	8	...
particles			2		0		0,2		0,2	...

(d)

Table 1. Finding particles that contribute to the displacement applied to each vertex

To get realistic results, we need to refine triangles in strongly eroded areas to avoid visually perceptible artifacts due to initial mesh resolution. This well-known problem [14] implies recursive subdivisions of neighbouring triangles, but we can't refine the mesh on the fly without degrading performances or increasing memory consumption to store neighbouring relationships. We choose to implement an offline pre-processing pass to generate more details where the erosion is

likely to occur. Generating larger triangles in areas where less erosion is expected is also a way to increase performances since there will be less triangles to handle in the simulation. Fortunately, detecting areas where erosion may be strong is relatively easy because the fluid always flows downwards; selection criteria for a triangle to be subdivided are thus its depth and its gradient vector given by the heightfield using vertices interpolation. Therefore the highest gradient and depth in a region, the more details we add (see Fig. 2).

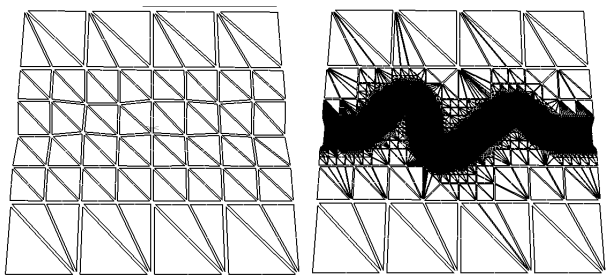

Fig. 2. Different levels of mesh subdivision.

4 Results and Conclusion

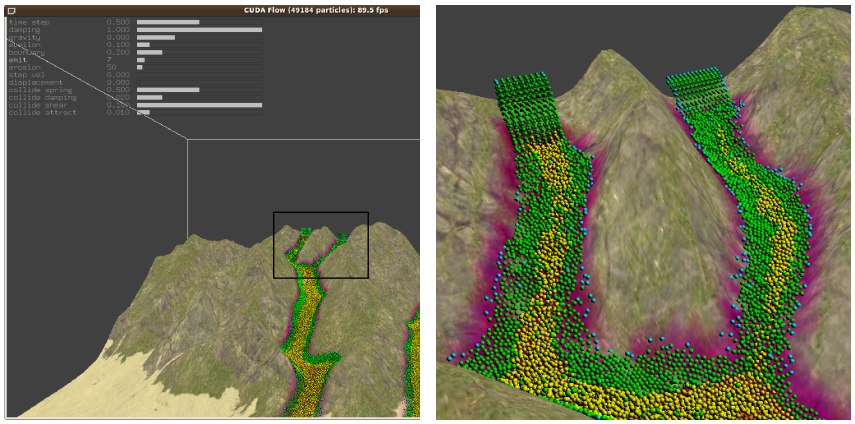

Fig. 3. Left. Streams falling down and eroding a mountain in real-time. **Right.** Zoom on a specific area, showing affected triangles in purple and particles coloration given by their position in the flow

After the terrain is subdivided by the method above and the user has set particles emitters, simulation starts and lets the user modify all parameters in real-time through sliders (time step, erosion rate, sediment capacity constant, etc.), as shown on Fig. 3 (top-left). The snapshot presented on Fig. 3 shows our interactive application with multiple fast-flowing streams with 130k particles running down a mountain terrain with 80k triangles at approx. 20 fps on a NIVIDIA Quadro FX 3700M. As a comparison, 1.38 seconds per frame are reported in [9] for the same number of particles, but these timings include a basic rendering of the fluid's surface. Another example is shown on Fig. 4 where the initial mesh presented on Fig. 2 (right) is deepened by a running river, and rendered offline on a supercomputer with 3x Tesla C1060 processors at approx. 9 seconds per frame for 20k triangles and 130k particles.

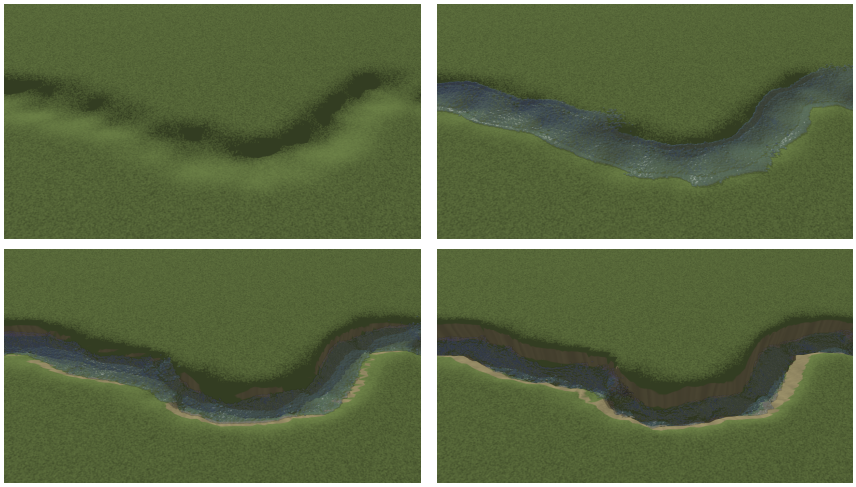

Fig. 4. Canyon progressively deepened by a river

Our method can be considered as physically-inspired, since it relies on physical principles such as gravity and friction but neglects other phenomena which would be too computationally expensive. Results are visually realistic and obtained in real-time; we are able to increase the number of particles in the simulation significantly compared to previous approaches. Integrating other phenomena related to erosion in the future should be relatively straightforward, for instance rain or sediment diffusion into the fluid (as in [9]). Handling dynamic topological changes occurring with hydraulic erosion (some parts of a cliff falling into a river is a good example) will require more complex mesh representations such as a 3D topological model, and an extra amount of work to maintain interactive rates. Another example is *lateral erosion*, which has an impact for example for valleys deepened by rivers. Unlike most existing methods which only consider vertical

erosion, in our approach we can choose to displace vertices along their normal vector instead of the vertical axis. However lateral erosion has to be limited because it can involve complex topological problems, such as merging or splitting material volumes.

References

1. Kevin Hiscock. *Hydrogeology: Principles and Practice*. Wiley-Blackwell, 2005.
2. Hubert Nguyen, editor. *GPU Gems 3*. NVIDIA Corporation, 2008.
3. Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans K. Pedersen. Modeling and rendering of weathered stone. In *SIGGRAPH*, pages 225–234, 1999.
4. N. Ozawa and I. Fujishiro. A morphological approach to volume synthesis of weathered stone. In *Volume Graphics*, pages 367–378, 1999.
5. D. Jones Michael, Farley McKay, Butler Joseph, and Beardall Matthew. Directable weathering of concave rock using curvature estimation. *IEEE Transactions on Visualization and Computer Graphics*, pages 81–94, 2009.
6. B. Neidhold, M. Wacker, and O. Deussen. Interactive physically based fluid and erosion simulation. In *Eurographics Workshop on Natural Phenomena*, 2005.
7. Xing Mei, Philippe Decaudin, and Bao-Gang Hu. Fast hydraulic erosion simulation and visualization on gpu. In *Pacific Graphics*, pages 47–56, 2007.
8. Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek. Interactive terrain modeling using hydraulic erosion. In *Symposium on Computer Animation*, pages 201–210, 2008.
9. Peter Kristof, Bedřich Beneš, Jaroslav Krivanek, and Stava Ondrej. Hydraulic erosion using smoothed particle hydrodynamics. In *Eurographics*, pages 219–228, 2009.
10. Funshing Sin, Adam W. Bargteil, and Jessica K. Hodgins. A point-based method for animating incompressible flow. In *Symposium on Computer Animation*, pages 247–255, 2009.
11. Didier Badouel. *An efficient ray-polygon intersection, Graphics Gems I*, pages 390–393. 1990.
12. Javor Kalojanov and Philipp Slusallek. A parallel algorithm for construction of uniform grids. In *High Performance Graphics*, pages 23–28, 2009.
13. Simon Green. Particle-based fluid simulation for games. In *Game Developers Conference*, 2008.
14. Brian Von Herzen and Alan H. Barr. Accurate triangulations of deformed, intersecting surfaces. In *SIGGRAPH*, pages 103–110, 1987.