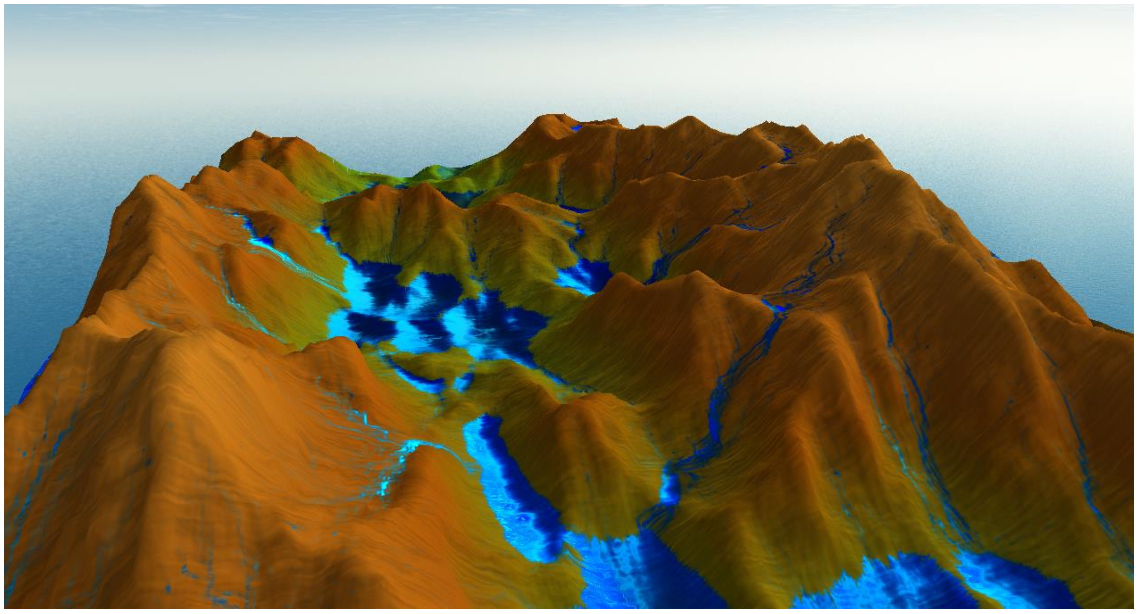
Gaia: Terrain Modeling and Rendering with Hydraulic Erosion Simulation

Mikey Chen, Vimanyu Jain
University of Pennsylvania



1. Introduction:

Terrain generation and rendering is a well research area with many useful applications found in visual effects, films, and games. Techniques such as fetching real world terrain height maps and fractal based procedural algorithms have proven to be powerful tools to generate visually stunning results. However, the aforementioned techniques lack the ability to generate effects over time found in nature. In this paper, we present our implementation on the GPU of erosion simulation on a fixed sized terrain generated by reading a height map.

2. Related works:

There have been numerous works that have been done previously in this area. For example, [St'ava et al.] presents an interactive physics-based modeling of terrain with support of different terrain layers simulated with various erosion and deposition algorithms. [Jako] presents a technique that models a procedural height field terrain through hydraulic and thermal erosion. Finally, [Mei et. al.] proposed a method for capturing the effects of hydraulic erosion through the use of shallow-water fluids model that can be implemented on the GPU. They have shown that realistic results of erosion effects by rainfall and river flows can be achieved efficiently. The work we present in this paper is largely inspired by this technique.

3. Implementation Details

We implemented the entire project in OpenGL, leveraging on framebuffer objects (FBOs) and rendering multiple off-screen textures storing information from our simulation. A big advantage of our implementation is that the data for the simulation stays on the device (GPU) and we aimed to minimize the bus transfers between the CPU and the GPU. A summary of our implementation is mentioned below,

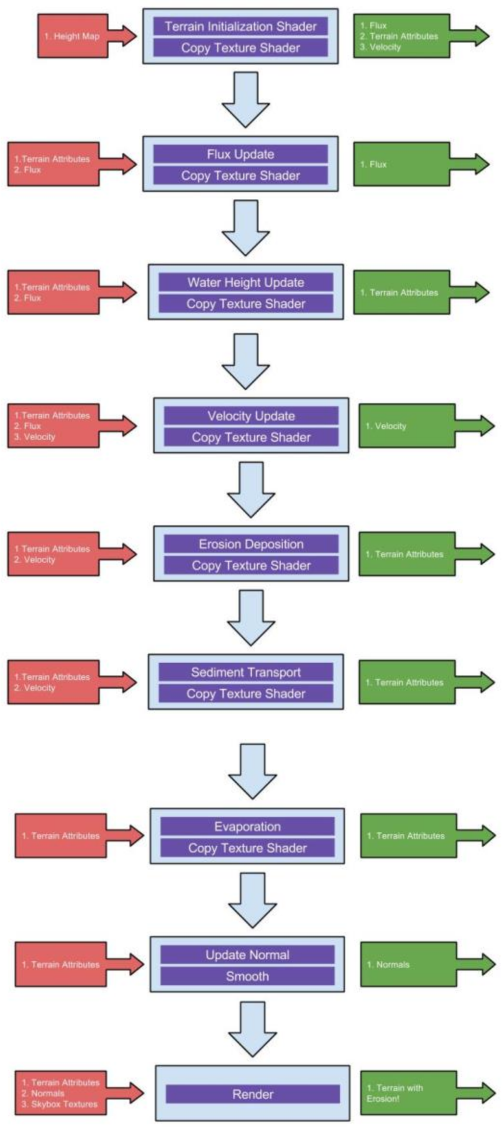


Figure 1. Pipeline Overview

3.1 Terrain Generation

In order to generate a terrain, we first set up a static grid with coordinates ranging from 0 to 1. This grid is then subdivided and passed to the remaining stages of our pipeline. The actual size of the terrain is computed in the tessellation evaluation shader. We provide support for two different techniques of modeling the initial terrain that will be used for simulation. The first and the more straight forward way is to upload a heightmap to the GPU as a texture and simply set the height of each vertex in the static grid accordingly.We also provide support for ridged multifractal terrain, where instead of reading from a texture, each of the height is generated procedurally based on a 2D simplex noise.

3.2 Normals Calculation

We tried a variety of ways to calculate normals, but the resulting images were noisy. We found that the Sobel filter with a 3X3 kernel produced the most accurate results. We had to further smooth them to remove the noisy artifacts. For smoothing, we used the 2D separable convolution technique as discussed in [1].

3.3 Hydraulic Erosion Model

Mei et al [2] adapted the 2.5D shallow water virtual pipe model of O'Brien et al [3] to the GPU. Water is transported through the pipe by the hydrostatic pressure difference between the neighbouring grid cells. This can be done independently for every cell and hence is embarrassingly parallel, and forms the basis for the erosion algorithm.

3.3 Erosion

The algorithms that we have implemented for simulating erosion is largely the same as [2]. In this section, we will provide a brief overview of each stage of the pipeline. For more information and the mathematical formulations, please refer to [2].

3.3.1 Water Increment

The main goal of this stage is to introduce water to the terrain. There are two main effects that we would like to capture:rainfall and river sources. For rainfall, our implementation will randomly select a position for each cell in the terrain and introduce a set amount of water. For river sources, we provide an interactive circular selection tool that allows the user to manually select areas that add water to. This area that is covered by the selection tool can be adjusted via keyboard inputs.

3.3.2 Flow Simulation

Flow simulation is split into three main parts: flux computation, water height update, and velocity update. To facilitate the understanding of this stage, assume that that all cells are interconnect with virtual pipes where water can freely flow.

In flux computation, the pressure difference between each cell is computed and used to accelerate the flow velocity. At the boundaries, the flux is set to zero.
Once the flux has been computed, the new water heights can be updated with the new outflow flux.

Finally, the 2D velocity is computed by sampling the neighboring flux values.

3.3.3 Erosion Deposition

During this stage, we calculate the amount of erosion and deposition that takes place as a result of running water. If the sediment carrying capacity of a cell exceeds its current suspended sediment amount, then some soil is dissolved into the water and added to the suspended sediment. Otherwise, some suspended sediment is deposited on the terrain.

3.3.4 Sediment Transport

We calculate the transportation of suspended sediment by solving the advection using Semi Lagrangian method.

3.3.5 Evaporation

Finally, some water is evaporated into the air due to the environmental temperature.

3.4 Rendering

The output of our simulation sub-system is a single texture containing the terrain height, water height and suspended sediment. We use this to displace the vertices on a pre-defined mesh.

3.4.1 Tessellation with Level of Detail

In order to be more efficient while rendering large terrains, we tessellated the coarse input mesh in the tessellation shaders (available in OpenGL 4+). Our LOD is based on the screen space coverage of the edges of triangles. We have a minimum threshold and if the edges cover more pixels than this threshold, we tessellate them further.

3.4.2 Ambient occlusion

Ambient occlusion is estimated by sampling the neighbors in a spiral pattern around the point being rendered. We learnt this technique from [5].

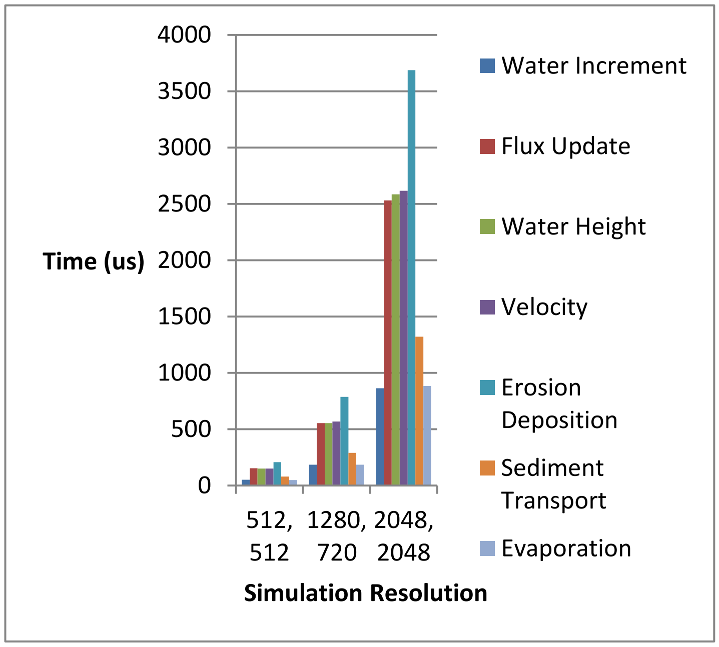
3.4.3 Water shader

Our implementation for the water shader is a simplification of [4]. We have incorporated fresnel refraction approximation calculation, environment map sampling for reflection, diffuse lighting, specular highlight, and water shading based on water height.

3.4.4 Skybox

Our implementation also supports skybox rendering. Our system takes in 6 texture images and uses OpenGL's cubemap texture.

4. Performance Analysis



For performance analysis, we decided to time each stage of our pipeline. Please consult the appendix for the table containing timing information.

5. Conclusions

We have provided a general overview of our implementation for a physically based hydraulic erosion terrain modeling and rendering pipeline. We were able to achieve satisfactory results at interactive framerates. User inputs via keyboard and mouse have also been provided to allow for dynamic manipulation of the simulation. Through OpenGL's framebuffer objects, we were able to follow the parallelization of the algorithm discussed in [2]. Better rendering for water have also been added to make the resulting rendering more visually appealing. Additionally, we also provide support for large terrain rendering with level of detail.

6. Future Work

Currently, our simulation runs on a fixed sized grid. One natural extension to our current work is to incorporate our simulation to infinitely large terrains. We would also be excited to incorporate other procedural terrain generation techniques to compare the results. Currently, the water simulation is done through shallow water equations, which is a 2D algorithm. We would also like to see if incorporating other fluid simulation techniques such as SPH would improve our erosion simulation. Other erosion models such as thermal erosion would also be worthwhile to implement.

7. References

[1]http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html

[2]Fast Hydraulic Erosion Simulation and Visualization on GPU, by Xing Mei et al

[3]Dynamic simulation of splashing fluids, by James O Brien et al

[4]**http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/rendering-water-as-a-post-process-effect-r2642**

[5]http://codeflow.org/entries/2011/nov/10/webgl-gpu-landscaping-and-erosion/

Appendix I

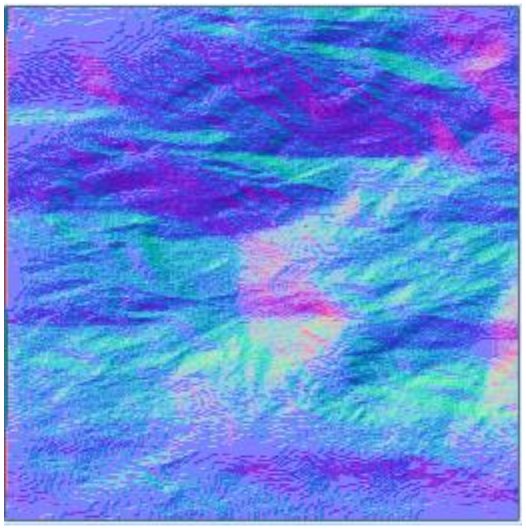


Figure 2. Noisy Normal

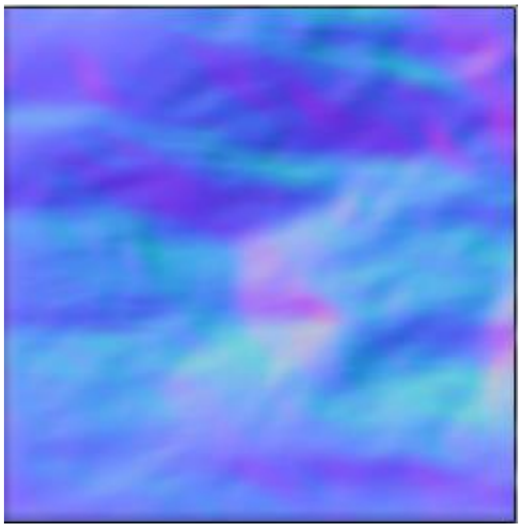


Figure 2. Reference Normals

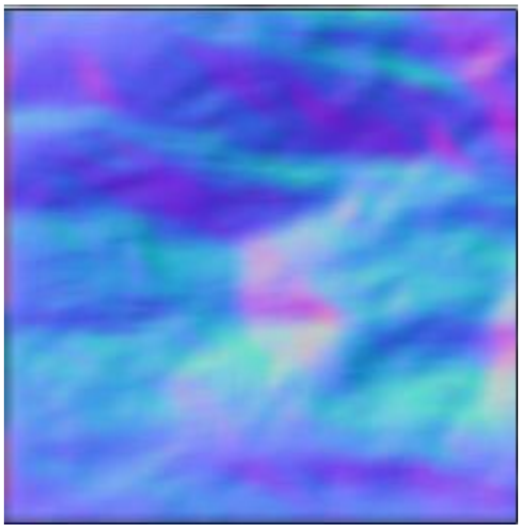


Figure 3. Smoothed Normals

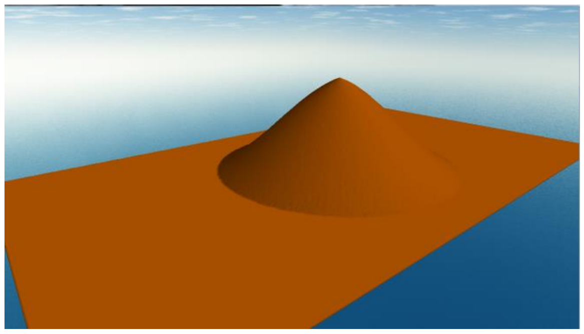


Figure 4. Original Terrain

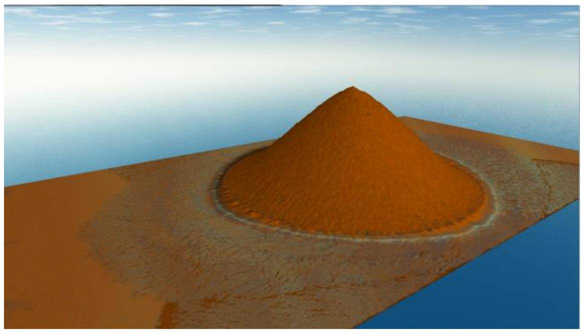


Figure 5. Erosion in progress

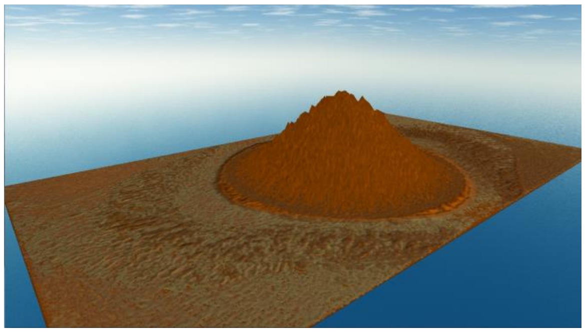


Figure 6. Eroded terrain