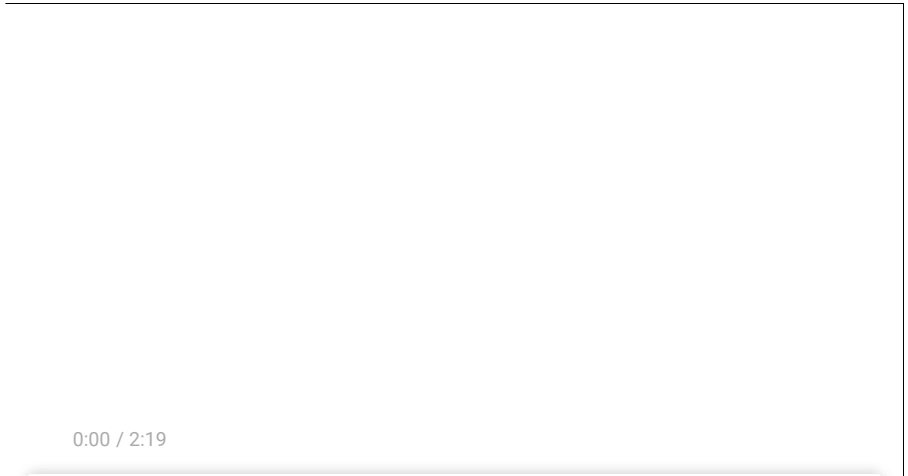
Interactive Hydraulic Erosion Simulator

Dylan Mcleod, Hanlin Chen, Newman Hu

Abstract

We create an interactive hydraulic erosion simulator for the procedural generation of natural terrain. Our simulation uses a shallow water fluid model and velocity field for calculating the erosion and deposition process. The entire simulation is developed to run on the GPU, using C++ and OpenGL, allowing for impressive framerates and full interactivity. Our model illustrates the effects of rain and river sources and allows users to place river sources to sculpt the landscape to their liking. Additionally, our application produces realistic water visualization with fully simulated waves, directional lighting, and textured terrain.



Video Link

Github link

Simulation

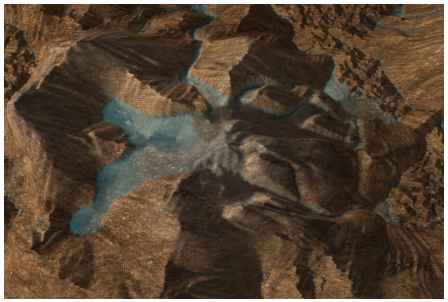
We implemented the hydraulic erosion model from Xing Mei, Philippe Decaudin, and Bao-Gang Hu [1]. Here we only provide a summary of the erosion model which is explained with much greater depth in their paper.

The model is represented by a 2D grid, with each cell containing the following information: terrain height **b**, water height **d**, suspended sediment amount **s**, the outflow flux **f** = (f_L, f_R, f_T , f_B), and velocity vector **v** = (u, v). We then store all of this information in textures on the GPU to be passed into shaders: texture T1 containing **b**, **d**, and **s**; texture T2 containing all components of **f**; and texture T3 containing the components of **v**.

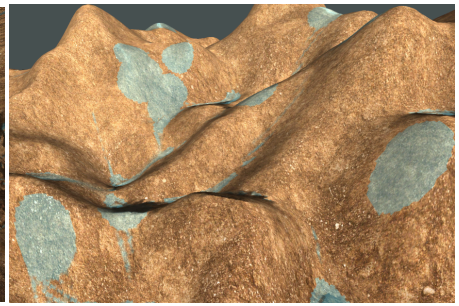
The following five steps provide a compact summary of the simulation process:

1. $d_1 \leftarrow WaterIncrement(d_t)$

At every time step water is added to the terrain through water sources (rivers) or through rain. Rain adds water from a random distribution and water sources are introduced by the user. For a given amount of water $r_t(x, y)$ the water level is updated by: $d_1(x, y) = d_t(x, y) + \Delta t \cdot r_t(x, y)$. This update is computed by a rain fragment shader and packed into T1. We choose to drop the rain in buckets instead of using something like a gaussian distribution, because completely uniform rain is a bit less interesting.



Water originating from sources



Some "buckets" of rain

2. $(d_2, \mathbf{f}_t + \Delta t, \vec{\mathbf{v}}_t + \Delta t) \leftarrow FlowSimulation(d_1, b_t, \mathbf{f}_t)$

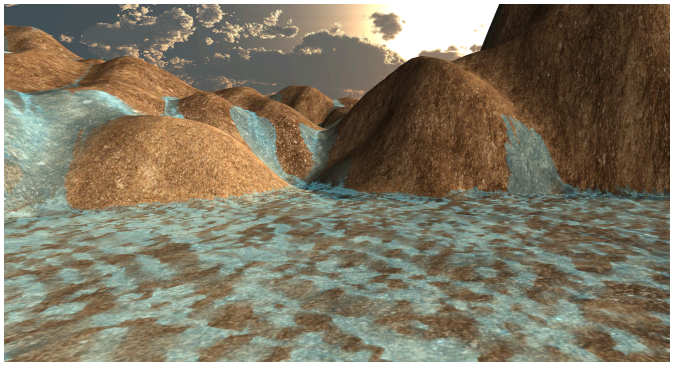
Because this simulation step updates water, flux, and velocity, it requires three shaders to write to all three of the textures. First, we update the outflow flux. As an

$f_{t+\Delta t}(x,y) = max(0, f_t(x,y) + \Delta t \cdot A \cdot g \cdot \frac{\Delta h^L}{l})$

A, g, and l are physical constants and Δh^L is the difference in total height (b + d) between the cell and its left neighbor. Then, to prevent the total flux from exceeding the current water amount, we scale the new fluxes by $K = min(1, \frac{d_1 \cdot l_X \cdot l_Y}{(f^L + f^R + f^T + f^B) \cdot \Delta t})$, where l_X and l_Y are the grid distances in each direction.

Next, we update the water surface by taking in the flux from neighboring cells and sending out the cell's outflow flux. We calculate the net change in water as $\Delta V(x,y) = \Delta t \cdot (\sum f_{in} - \sum f_{out})$ and update the water $d_2(x,y) = d_1(x,y) + \frac{\Delta V(x,y)}{l_X \cdot l_Y}$.

Finally, we update the velocity flow. Again, we will use the x component u as an example, but the process should be repeated for the y component v. First we find the average amount of water passing through in the x direction: $W_X = \frac{f^R(x-1,y) - f^L(x,y) + f^R(x,y) - f^L(x+1,y)}{2}$. Then, we calculate $u = \frac{\Delta W_X}{l_Y \cdot \bar{d}}$, where $\bar{d}$ is the average water level between this and last step, $\bar{d} = \frac{d_1 + d_2}{2}$.



A simulated beach

3. $(b_{t+\Delta t}, s_1) \leftarrow ErosionDeposition(\vec{\mathbf{v}}_{t+\Delta t}, b_t, s_t)$

The whole purpose of modeling water is for some soil to be carried away by the water as erosion. In this model the erosion-deposition process is affected by local tilt angle,

$C(x,y) = K_c \cdot sin(\alpha(x,y)) \cdot |\vec{v}(x,y)|$, where α is the local tilt angle, $\vec{v}$ is the water velocity, and K_c is the sediment capacity constant.

Next we check C against the suspended sediment value s of the cell. If $C > s_t$ some soil is dissolved in the water. K_s is the dissolving constant.

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

$s_1 = s_t + K_s(C - s_t)$

Conversely, if $C \leq s_t$ some soil is deposited onto the terrain and removed from the water. K_d is the deposition constant.

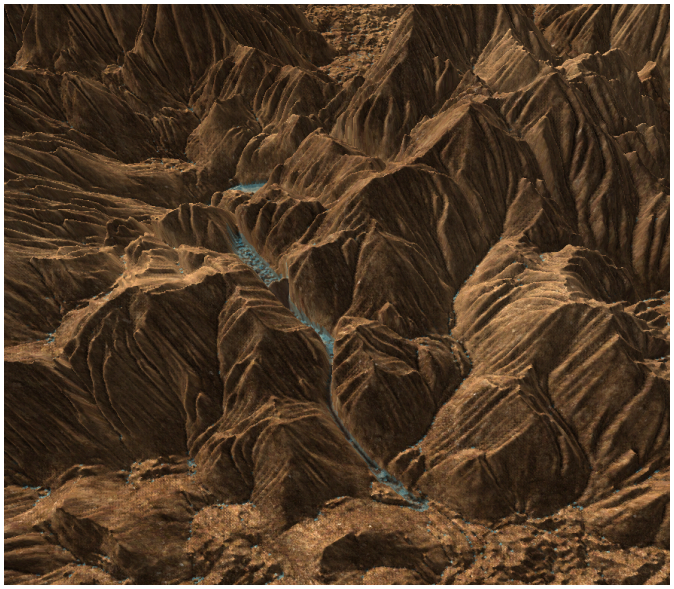
$b_{t+\Delta t} = b_t + K_d(s_t - C)$

$s_1 = s_t - K_d(s_t - C)$

We also deviate from the reference and multiply C(x,y) by the height of the water at x,y, clamped between 0 and 1. Otherwise, an arbitrarily small amount of water could dissolve a lot of sediment which is not realistic.

4. $s_{t+\Delta t} \leftarrow SedimentTransport(s_1, \vec{\mathbf{v}}_t + \Delta t)$

The sediment transport process is described by the equation $\frac{\delta s}{\delta t} + (\vec{\mathbf{v}} \cdot \Delta s) = 0$. We get the update step by taking an Euler step back in time:
$s_{t+\Delta t}(x,y) = s_1(x - u \cdot \Delta t, y - v \cdot \Delta t)$. When the step coordinates (x - u · Δt, y - v · Δt) don't correspond to exact grid points, we use bilinear interpolation to interpolate the value of s_1 based on the four surrounding grid points.



The awesome power of water, splitting a mountain

5. $d_{t+\Delta t} \leftarrow Evaporation(d_2)$

Finally, some water gets evaporated into the air (assumes constant temperature) proportionally to the amount of water, based on the evaporation constant K_e:
$d_{t+\Delta t}(x, y) = d_2(x, y) \cdot (1 - K_e \cdot \Delta t)$

Visualization

To visualize our results, we render in two passes -- one pass for the terrain, and another for the water. For the terrain, we use the b(x,y) value in the vertex shader to determine the height. Then, in the fragment shader we use a procedural texture, combining a dirt, sand, and rock texture according to perlin noise, lit with phong shading. For water, we use b(x,y) + d(x,y) for the height, and discolor it based on the amount of dissolved sediment, s(x,y). The water and terrain renderings are combined in a final shader, allowing us to shade the water based on how deep a projected camera ray travels before hitting something else. Also, the nice waves and ripples in the water are completely generated from the simulation and not a rendering illusion.

Additional Implementation and Challenges

a certain route, eroding it a little, then it would continue to choose to go down that route again and again. While this is realistic to a degree, we were finding that large amounts of water (literal rivers worth) would create only very narrow channels, and no wide ones.

To solve this, we made a few changes/additions to the reference procedure. (1)When performing the erosion/deposition update, we adjust the dissolving rate based on Perlin noise and the height of the terrain, in order to mimic different materials (rocks, below the surface, tend to be more resistant to erosion). The inspiration for this tweak came from Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek [2], which demonstrated different sediment layers with different erosion characteristics. (2) At the end of every simulation cycle, we add a smoothing shader that, at local extrema, enforces the change in the terrain's height to not be too large.

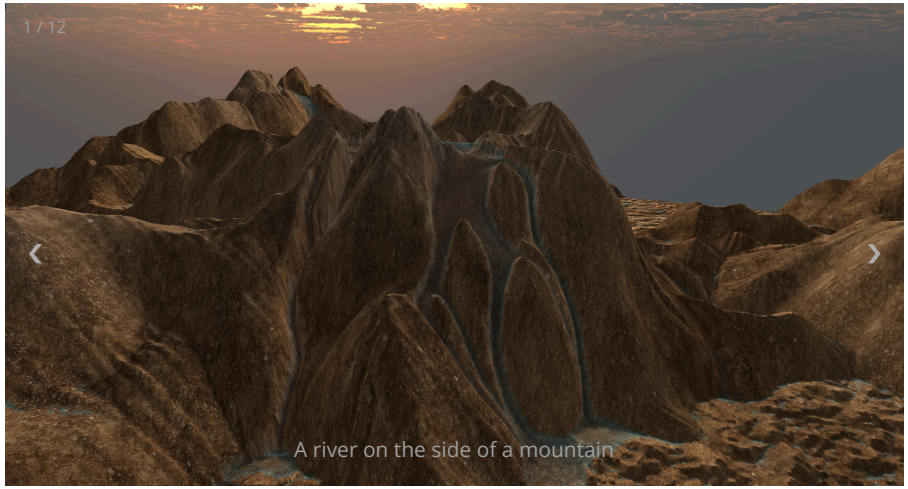
We found that to get the best results, we'll make the smoothing initially very high and place sources, creating wide channels, then we introduce rain and lower the smoothing, creating thin cracks.

Lessons

Overall, this was a very interesting and educational project. We expanded our proficiency in writing shaders and we were able to apply many of the rendering techniques taught in CS184. One of the main problems we encountered was not planning sound infrastructure for the codebase before implementing the simulation. In future work, we should construct the project to be more extensible for additional cool features.

Results

You can use the left/right arrows to navigate the slideshow.



References

[1] Xing Mei, Philippe Decaudin, Bao-Gang Hu. Fast Hydraulic Erosion Simulation and Visualization on GPU. PG '07 - 15th Pacific Conference on Computer Graphics and Applications, Oct 2007, Maui, United States. pp.47-56, ff10.1109/PG.2007.15ff. ffinria-00402079f

[2] Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek. 2008. Interactive terrain modeling using hydraulic erosion. In Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08). Eurographics Association, Goslar, DEU, 201–210.

Contributions

Thank you Dylan for putting together the starter code, most of the C++/OpenGL simulation and rendering code, and the visualization shaders. Also, thank you for experimenting and doing a bunch of little things to improve the results slightly.

Thank you Newman for creating most of the user interface associated with changing simulation parameters, placing water sources, and controlling the camera. Also, for implementing the openGL terrain loader and writing the preliminary terrain shaders.

Thank you Hanlin for researching possible implementations and writing the shaders related to erosion simulation.

Milestone updates

Status report

We found two very helpful papers on the subject of simulating erosion [1] and [2]. Both of these papers can be implemented using fragment shaders on textures which represent heightmaps/terrain characteristics/etc. So, to start off, we started by setting up a simple simulation using fragment shaders, namely conway's game of life. Since Conway's game of life uses the information of textures in a small region to determine the next pixel of that texture, we figured it would be a good test of our fragment shader simulation code.

To simulate things using OpenGL fragment shaders, we use two OpenGL framebuffers (textures which can be rendered to), one representing the current frame and one representing the next. We run the simulation shader with the input texture being the current frame to determine the next frame, then we swap the framebuffers so that the next frame becomes the current. If we end up needing numerical derivatives (storing more than one timestep), we can just modify this to use more framebuffers, or more buffering.

Our erosion model will be based on a 2-D grid with virtual pipes dictating the flow between cells. Each cell contains information for the following: terrain height, water height, suspended sediment amount, the outflow flux components (left, right, top, bottom), and velocity vector. Then, the simulation will run the following updates: 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

various pieces of information into appropriate textures that can then be passed into the shader.

We haven't really followed our initial plan, as the way we wanted to implement the simulation wouldn't have worked well. Simulating all the water and terrain/sediment as particles would have been too complicated, so instead, we are now following the model that we found in the two papers referenced above. They simplify the simulation to height maps for water/terrain heights, and use shallow water equations to model the behavior of the water, rather than a fully 3-D implementation. So, because passing textures to shaders was an essential step and the math modeling the water/terrain updates is quite complex, we decided to first approach the pipeline of passing textures to the shader and having the shader make updates (we used Conway's game of life as a simple example).

Now that we have our program set up to pass in textures to the shader, we need to create input textures corresponding to starting terrains, and implement the update steps for our simulation.

[1] Fast Hydraulic Erosion Simulation and Visualization on GPU

[2] Interactive Terrain Modeling Using Hydraulic Erosion

Video

Presentation Slides

Proposal

Final Milestone Proposal

Problem Description

We are interested in simulating erosion for the procedural generation of natural looking terrain. This application will be able to simulate natural erosion processes that are unobservable in real time scales. We plan on addressing the effects of water and wind on loosely packed material like soil and sand.

Goals and Deliverable

We plan to implement at least one erosive agent and one eroding material most likely wind and sand. With this application we plan to have basic interaction where the user is able to direct the erosive agent to affect the sand. We plan on implementing this with a platform independent online demo.

We hope to create an application that allows you to adjust time scales, as well as choose both the material being eroded and the erosive agent. We would like for it to be fully interactive, with the user having full control of all these parameters when simulating. Ideally, the user will be able to design their starting scene, pick parameters, and run the simulation for a certain amount of time. A successful implementation will ideally run online and allow users to interact with the application independent of their platform. A feature we hope to also implement is real-time rendering on the GPU.

Schedule

Week 1: Basic fluid simulation
    i: implement platform
Week 2: Implement erosion interactions
    i: Sand as small particles
Week 3: Create interactive application
    i: enable user specification of erosive agent
Week 4: Extras: more materials, more erosive agents, real-time GPU simulation

Resources

Link this video on your webpage.Link this video on your webpage.

Hybrid Fluid Simulation, APIC: https://www.math.ucla.edu/~jteran/papers/JSSTS15.pdf Adaptive Tearing and Cracking of Thin Sheets: http://graphics.berkeley.edu/papers/Pfaff-ATC-2014-07/Pfaff-ATC-2014-07.pdf Folding and Crumpling Adaptive Sheets: http://graphics.berkeley.edu/papers/Narain-FCA-2013-07/Narain-FCA-2013-07.pdf