Procedural World Generation – Hydraulic Erosion

Gabriel Mañeru

1 Introduction

Visually appealing landscapes are beginning to settle as the standard in most 3D exploration games. It helps immerse the player into the virtual world and gives a more realistic experience. An important part of the view is drawn from the terrain shape which can be achieved by a terrain sculping artist. However, not only this is highly time consuming, but is quite difficult to create realistic looking landscapes as each artist ends up apply his own style deviating from the original natural looking view. Also, as the human is used to terrain's traces, any small errors ends up being quite striking.

In order to simplify these processes, simulation algorithms have been introduced in order to reproduce as fair as possible the natural process that develops a terrain. However, we can't model the perfection of our planet inside a simulation. Instead, we try to combine our interpretation of the real process with a fair number of particles that follows the laws of the nature. Our resulting model takes up different input parameters and after a series of iterations, we obtain a natural realistic looking terrain.

Landscape showing erosion down the hills

One of the most important and obvious aspects of the terrain is the erosion trace left on it. Several processes are involved on it, but the most noticeable is the passage of water over sediment named Hydraulic Erosion.

2 Hydraulic Erosion

The hydraulic erosion is produced by the presence of water on a dissolvable material. The droplet absorbs some part and take it downhill. The quantity varies depending on the normal force exerted on the soil.

When the droplet first contacts the terrain it begins to flow down, speeding up and solving even more material. The overall density grows as the material's density is higher than the water's one. Also, the evaporation of the water decreases its volume and therefore it's absorption capacity.

The stable evolution between speed, volume and density allows to continue eroding the medium and altering those values. However, it exists a sweet spot: when transgressing this equilibrium, the droplet has no more the speed or volume enough to continue eroding. Instead, the droplet starts depositing part of the absorbed soil. This allows it to recover part of its original density, restoring it into the sweet spot.

However, the volume lost by evaporation can't be recovered. This is what bring the droplet's life to an end when it has lost its total volume.

3 Initial Statements

Some terms must be defined before going through the implementation

3.1 2D-Heightmap

A 2D-Heightmap is a bidimensional array containing one channel, usually interpreted as the height of the map in its 3D representation.

3.2 Terrain

For representing the terrain, I will use the previously defined heightmap where each location of the grid will represent the height of that 3D vertex. This can keep iterations simple as the erosion force that we will compute, will exclusively point downwards: so, no displacement will take place on the 2D-plane. This also allows to be easily converted into a 3D mesh for rendering it.

3.3 Particles

For representing the particles, I will use a custom data structure. A particle can be defined by a position, a direction, its speed, its water volume and its sediment volume. It will also be defined by a lifetime value which will be limited for ensuring stability purposes.

3.4 Input

The input I will receive will be an existing terrain. As I am no artist, I will procedurally generate a terrain adding multiple noise layers. However, as we would see later, the input doesn't specially need any feature: it doesn't matter if we receive an already realistic terrain, or a pure geometrical one. The erosion will end up giving a natural look anyway.

4 Particle based Implementation

This algorithm could have multiple approaches in which we have drops that are place on a map and run downhill moving material depending on a carry capacity and a speed.

My implementation will have that within a same iteration the drops do not interact with themselves and could have different step times, thus this is not a suitable for a visual representation of the fluid, it will just simulate the consequences of impacting the terrain.

4.1 Particles

The particle will start with a random position ($\mathrm{Pos}_{\mathrm{drop}}$) inside the map. I will assume it has just fall with a speed value of 1m/s, a water volume of 1m^3 and the direction on the 2D plane, the sediment volume and the lifetime are zero. As particle they will also be controlled by some external factors. Some of them are the $inertia$, the evaporation rate and a maximum lifetime. The last two only purposes are to limit the existence of the particles and the inertia defines how the drop react to the change of gradient.

The goal is to move the drop along the path of least resistance from the current $pos_{old}$to a pos_{new} of local minimum height. For displacing the drop, we will extract the gradient of the heightmap at pos_{old}, this is computed by a simple bilinear interpolation taking the gradients of the four surrounding grid vertices:

$$\mathrm{gradient}(\mathrm{P}_{\mathrm{x,y}}) = \begin{pmatrix} \mathrm{P}_{\mathrm{x+1,y}} - \mathrm{P}_{\mathrm{x,y}} \\ \mathrm{P}_{\mathrm{x,y+1}} - \mathrm{P}_{\mathrm{x,y}} \end{pmatrix}$$
$$\mathrm{gradient}(\mathrm{P}_{\mathrm{x+1,y}}) = \begin{pmatrix} \mathrm{P}_{\mathrm{x+2,y}} - \mathrm{P}_{\mathrm{x+1,y}} \\ \mathrm{P}_{\mathrm{x+1,y+1}} - \mathrm{P}_{\mathrm{x+1,y}} \end{pmatrix}$$
$$\mathrm{gradient}(P_{x,y+1}) = \begin{pmatrix} P_{x+1,y+1} - P_{x,y+1} \\ P_{x,y+2} - P_{x,y+1} \end{pmatrix}$$
$$\mathrm{gradient}(\mathrm{P}_{\mathrm{x+1,y+1}}) = \begin{pmatrix} \mathrm{P}_{\mathrm{x+2,y+1}} - \mathrm{P}_{\mathrm{x+1,y+1}} \\ \mathrm{P}_{\mathrm{x+1,y+2}} - \mathrm{P}_{\mathrm{x+1,y+1}} \end{pmatrix}$$

We also compute the deviation Δx and Δy:

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = pos_{old} - floor(pos_{old})$$

From the four gradients we compute the bilinear interpolation:

$$\mathrm{gradient}(pos_{old}) = \begin{pmatrix} (P_{x+1,y} - P_{x,y}) \cdot (1 - \Delta x) + (P_{x+1,y+1} - P_{x,y+1}) \cdot \Delta x \\ (P_{x,y+1} - P_{x,y}) \cdot (1 - \Delta y) + (P_{x+1,y+1} - P_{x+1,y}) \cdot \Delta y \end{pmatrix}$$

The gradient is used to determine the new direction (dir_{new}) of the drop. It is computed by a linear interpolation between the dir_{old} and the just computed gradient using the $inertia$ as coefficient. Thus, if the $inertia$ is 0, dir_{old} is ignored, and the drop strictly follow the path of minimum resistance whereas if the $inertia$ is 1 (where the gradient is totally ignored) or the gradient is a zero vector (meaning the surface is flat), is leads the drop to stay at pos_{old} until complete evaporation.

$$\mathrm{di}r_{new} = \mathrm{di}r_{old} \cdot \mathrm{inertia} - \mathrm{gradient}(pos_{old}) \cdot (1 - inertia)$$

The new position (pos_{new}) of the drop is then computed by simply adding the normalized dir_{new} to pos_{old}. As the magnitude of a normalized vector is 1, the particle will move at a constant rate regardless of the $speed$. Using a magnitude higher than 1 will result in some cells being skipped in the path, if so, this will cause inconsistencies in the terrain as small hills that the drop should have been eroded or small pits where the drop should have deposit in. Using a magnitude lower than 1 could give more accurate results acting as if the map had more resolution however the differences aren't noticeable after a lot of iterations and ignoring it benefit performance.

The difference of height ($\Delta height$) is computed interpolating $height_{new}$ from the surrounding grid points (same as interpolating the gradient) and subtracting $height_{old}$.

$$\Delta height = height_{new} - height_{old}$$

This $\Delta height$ determines whether we are moving downhill or uphill. If it is positive, pos_{new} is higher than pos_{old} and sediment carried by the drop should be deposited at pos_{old} to fill the pit the drop is running through. If the drop carries enough sediment the pit could be filled, otherwise the drop will try to fill it as much as possible losing all its sediment. If it is negative, pos_{new} is lower than $pos_{old}$and we compute the capacity value using the $\Delta height$, the current $speed$, the water volume ($water$) and a $\mathrm{capacity}_{\mathrm{factor}}$. If $\Delta height$ tends to 0, the capacity will also be carried to 0 which leads to not eroding any flat surfaces, thus having a terrain with terrace looking plains. For this case, we use a minimum for $\Delta height$ ($minSlope$) preventing the capacity to fall too close to 0.

$$capacity = max(-\Delta height, minSlope) \cdot speed \cdot water \cdot capacity_{factor}$$

If the drop carries more sediment than the resulting $capacity$, it should drop some part of the extra sediment defined by $deposition_{factor}$ at pos_{old}. If the whole extra part is dropped, it would result in a spike and a sudden loss of speed.

$$\mathrm{t}o D\mathrm{eposit} = (sediment - capacity) \cdot deposition_{factor}$$

If the drop carries less sediment that the $capacity$ it should erode some part of the remaining extra capacity defined by $erosion_{factor}$ from pos_{old}. If the drop takes more sediment than $\Delta height$, the drop will be able to dig holes which is not wanted, so we limit the erosion with it.

$$\mathrm{t}o D\mathrm{eposit} = min\Big((capacity - sediment) \cdot erosion_{factor}, -\Delta height\Big)$$

Afterwards, we must adjust the $speed$ by computing the vector size of the sum between the speed and the gravity. Also, some part of the $water$ must be evaporated.

$$\text{speed} = \sqrt{\text{speed}^2 + \Delta height \cdot gravity}$$
$$\text{water} = \text{water} \cdot (1 - \text{evaporation})$$

This represent a single iteration of a drop. This process is repeated until the drop moves out of the map, stops at a flat surface or pit, it dries off or reaches the maximum lifetime.

4.2 Map application

We are using a discreet resolution to simulate this process, then we need to make all the changes as smooth as we could in order to avoid pitfalls and spikes. Then instead of eroding a single tile, we want to gradually take sediment from the surrounding tiles using a brush of a fixed radius. Every grid vertex inside the brush will lose sediment according to its $weight$ which linearly decrease with the distance from the drop position. All weight is normalize using the overall sum to ensure the exact same amount of sediment is eroded. We could play with having different weights influences depending on the material properties (ex: sand, grass, solid rock) however we will keep things simple and set an overall material density for every tile.

For distributing the deposit of sediment, we don’t need that much smoothness, we could simply release it on the four surrounding tiles using bilinear interpolation.

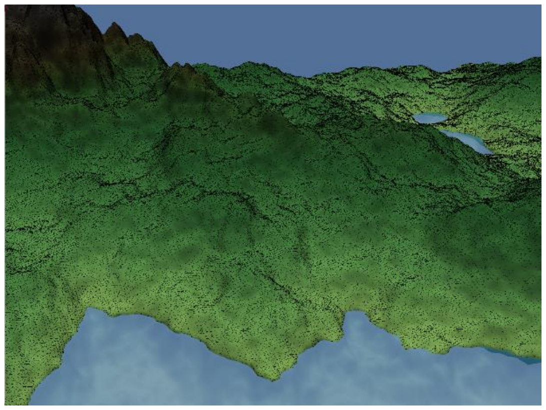
4.3 Extra tools

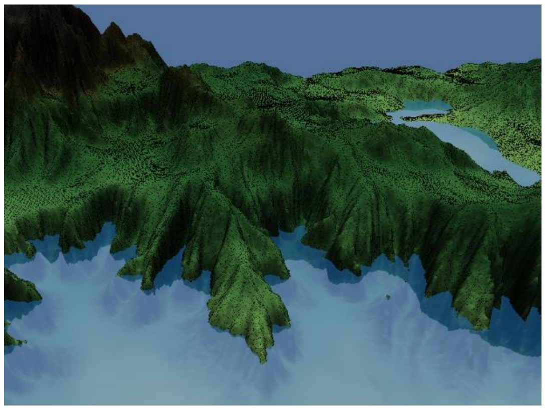
Depending on the number of iterations and the input, the map could degenerate into pitfalls or thin ravines/cliffs. In order to avoid it or control it, it’s convenient to add some blurring option to our map. We could either blur it at some rate after a set number of iterations to ensure the ravines/cliffs remain thick enough. Also, we could use it after running the algorithm to remove some pits and spikes that could be formed. However, in any of those cases you should never blur completely the map, or you will lose the erosion texture. Then, we will linearly interpolate the result between the unblurred and blurred values using a $blurForce$ parameter.

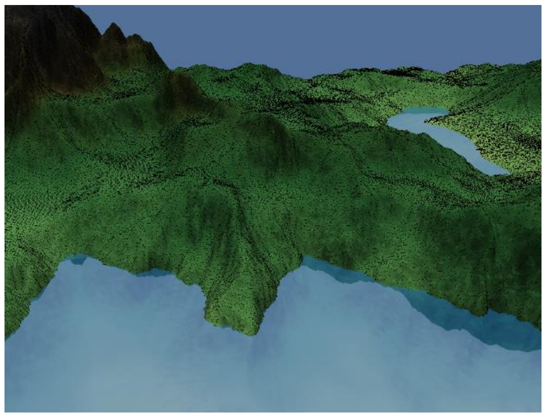
5 Parameter showcases

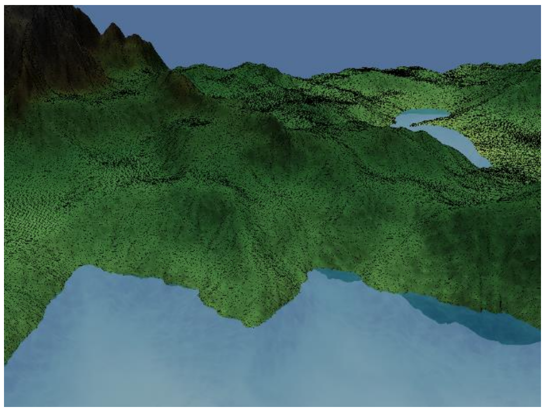
In the following part all relevant parameters will be explained showcase their effect on the terrain.

5.1 Inertia


(1) initial terrain


(2) $inertia = 0.025$

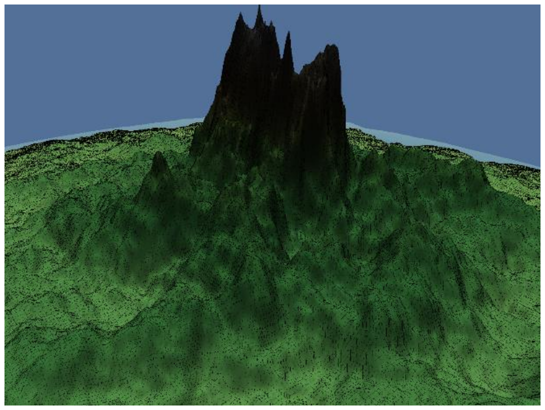

(3) $inertia = 0.1$

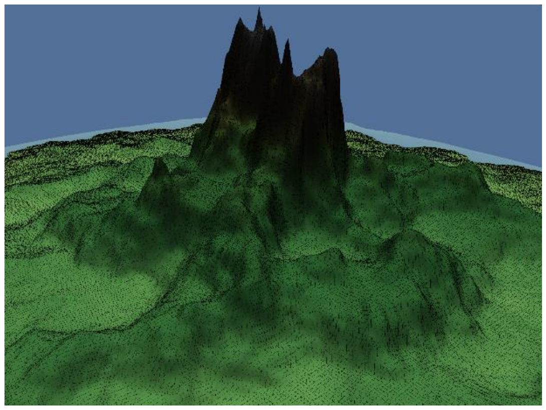

(4) $inertia = 0.4$

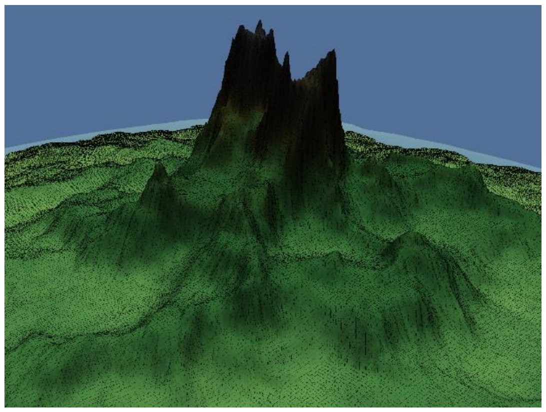
Example terrain eroded with different values of $inertia$. The terrain has resolution of 512x512. In (2),(3) and (4), 500,000 drops were simulated.

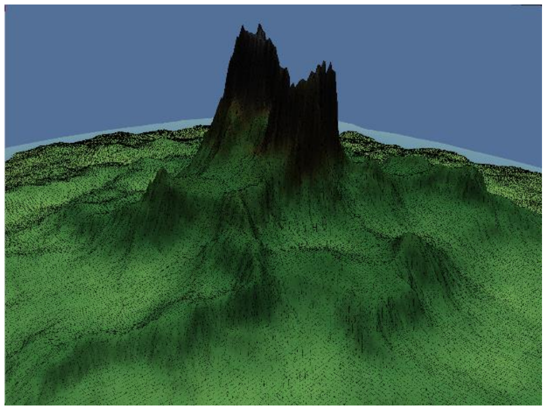
The parameter $inertia$ is bound between 0 and 1. The less inertia the droplet has, it tends with more intensity towards pits and ravines digging those valleys even deeper. That's why any small deviations from the flatness of a surface will degenerate into a ravine. This also intensifies the sharpness of the slope's terrain flattening the lower parts. However, a higher inertia will have more bounciness and will affect less the terrain needing a higher number of iterations on average to produce a similar result than with lesser inertia.

5.2 Capacity Factor


(1) initial terrain


(2) $capacity_{factor} = 2$

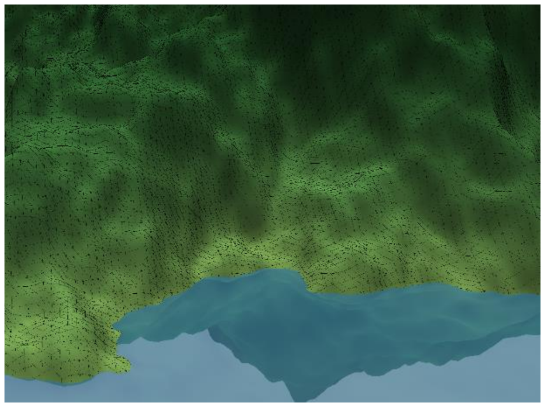

(3) $capacity_{factor} = 8$

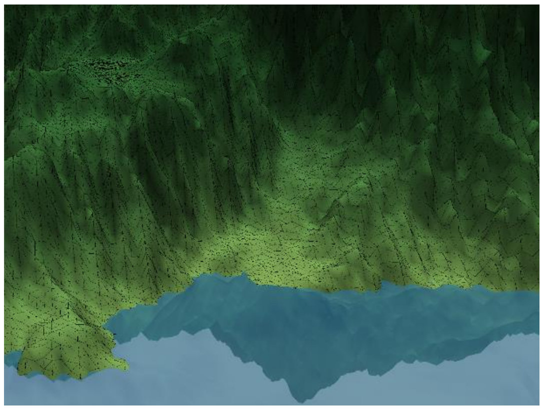

(4) $capacity_{factor} = 32$

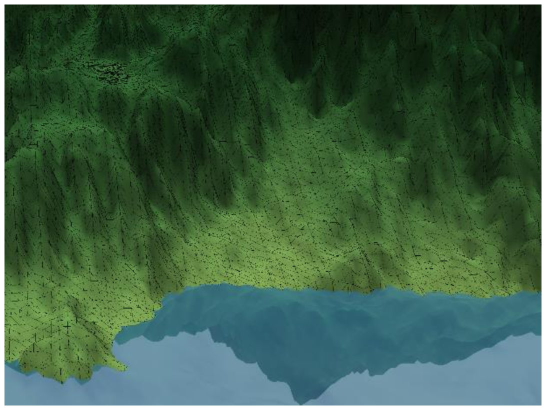
Example terrain eroded with different values of $capacity_{factor}$. The terrain has resolution of 512x512. In (2),(3) and (4), 500,000 drops were simulated.

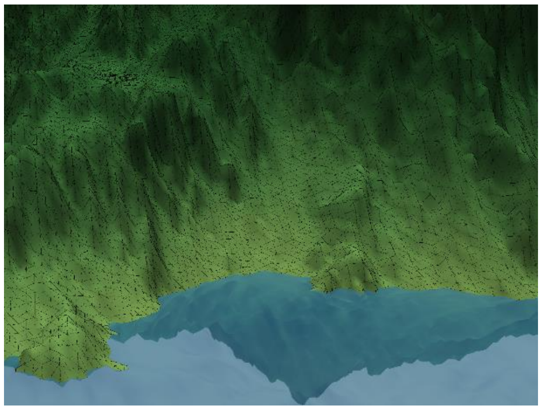
The $capacity_{factor}$ determines the quantity of sediment a drop can carry relating the speed and its water volume. The higher the value is, more sediment can be displaced from steeper ground to lower regions per droplet. Then each drop that start on a higher position will have more impact on the result leading to a slimmer terrain with more ravines. Having a lower value will give a smoother result but will need more droplets to reach the same erosion level.

5.3 Deposit Factor


(1) initial terrain


(2) $deposit_{factor} = 0.01$

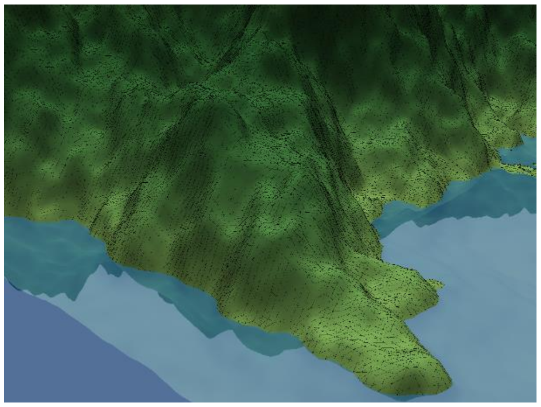

(3) $deposit_{factor} = 0.1$

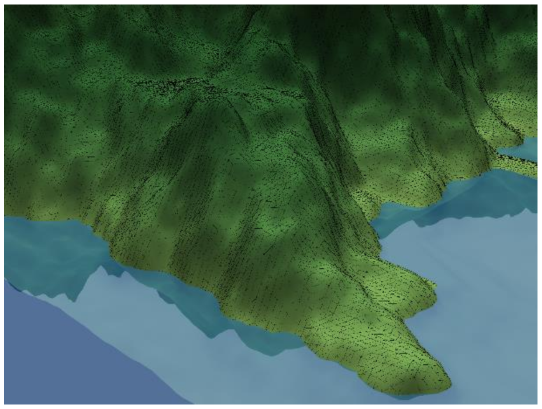

(4) $deposit_{factor} = 1$

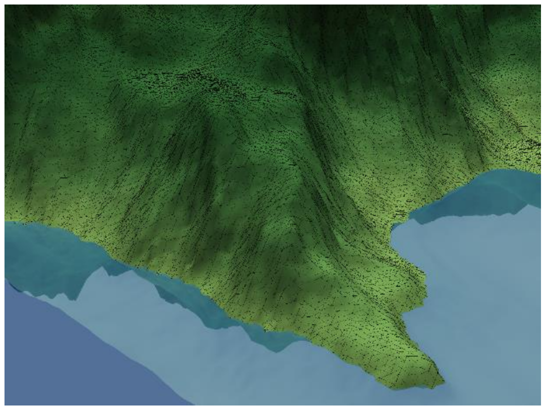
Example terrain eroded with different values of $deposit_{factor}$ and a low $inertia$ in order to force ravines. The terrain has resolution of 512x512. In (2),(3) and (4), 500,000 drops were simulated.

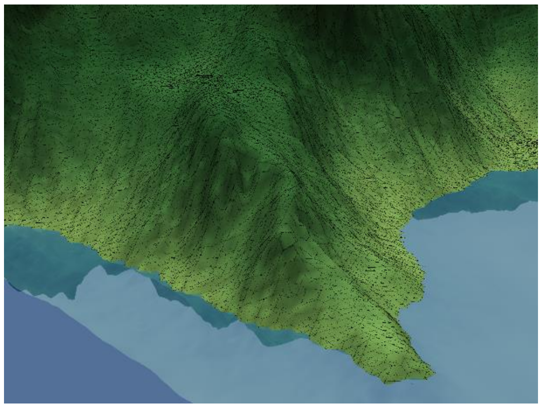
The $deposit_{factor}$determines the amount of sediment surplus that is dropped when it exists, and it is responsible of giving the texture and shape inside the ravines. The value is bound between 0 and 1, where 0 means no sediment is deposed and 1 the entire surplus is deposed at once. The differences are less noticeable than other parameters as we can see comparing the three images(2),(3) and (4).
When the factor tends to 0, that means it almost never deposit soil while descending the ravine, that makes that the ravine has less shape, less texture and is longer. When the factor is close to 1, as soon as it takes a surplus it would depose it, leaving a spike on the ravine. Having a lot of iterations will end up bringing the spikes material to the lower ground and removing them. It seems that that a value close to 0.1 gives a stable shape without noticeable spikes and a realistic enough texture.

5.4 Erosion Factor


(1) initial terrain


(2) $\text{erosion}_{factor} = 0.01$


(3) $\text{erosion}_{factor} = 0.1$


(4) $\text{erosion}_{factor} = 1$

Example terrain eroded with different values of erosion_{factor}. The terrain has resolution of 512x512. In (2),(3) and (4), 500,000 drops were simulated.

The $erosion_{factor}$ determines the amount of free capacity that can be filled with sediment. The value is bound between 0 and 1, where 0 means no sediment is eroded and 1 the whole capacity is filled and carved from the initial position. Like the $deposit_{factor}$, it gives a stable shape for the value 0,1. When it tends to 0 less erosion takes place and when it tends to 1 more pitfalls are carved.

5.5 Other parameters

The $evaporation$ determines how fast a drop loses $water$ and dries off. Again, its value is bound between 0 and 1, where 0 means the drop will never dry off and 1 that it would do it on the first iteration. It is a natural consistency parameter; however, it doesn’t change much the result unless bringing it to the extreme points.

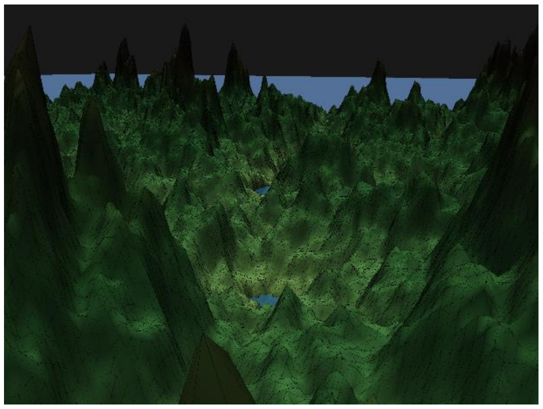
The $erosionRadius$ determines the intensity of each drop. The smallest the radius, each droplet's path should be deeper and recognizable at the end of the algorithm. However, as we want a terrain that has been totally affected by the erosion, we need so many particles that you can't notice the individual participation. Therefore, we won't see any major changes when varying the radius.

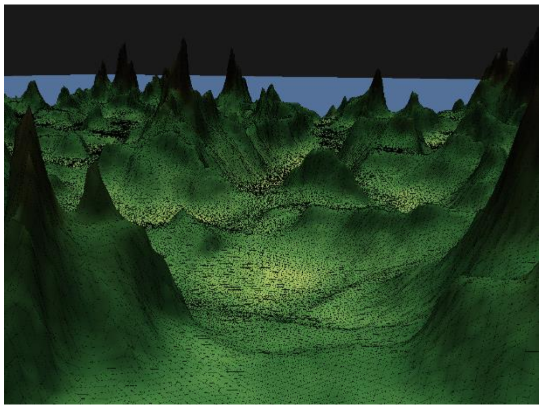
The $minSlope$ is the minimum value for $\Delta height$ that is taken for the $capacity$ computation. The value ensures the drop doesn't stop on a flat surface. Same as we saw with the variance of $capacity$, it controls the ravine formation dragging material from the steeper cells.

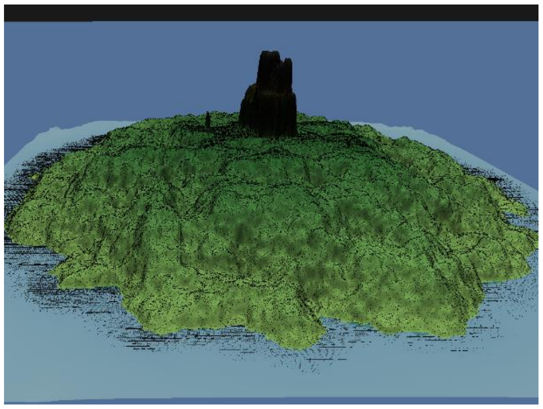
The $maxLifetime$ determines how much a drop can live. For enhancing the realism of the simulation, every droplet should dry off by simple evaporation, the only purpose of the $lifetime$ is ensuring stability to the program.

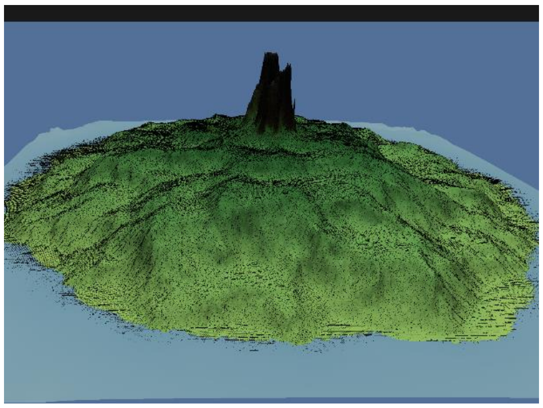
6 Examples

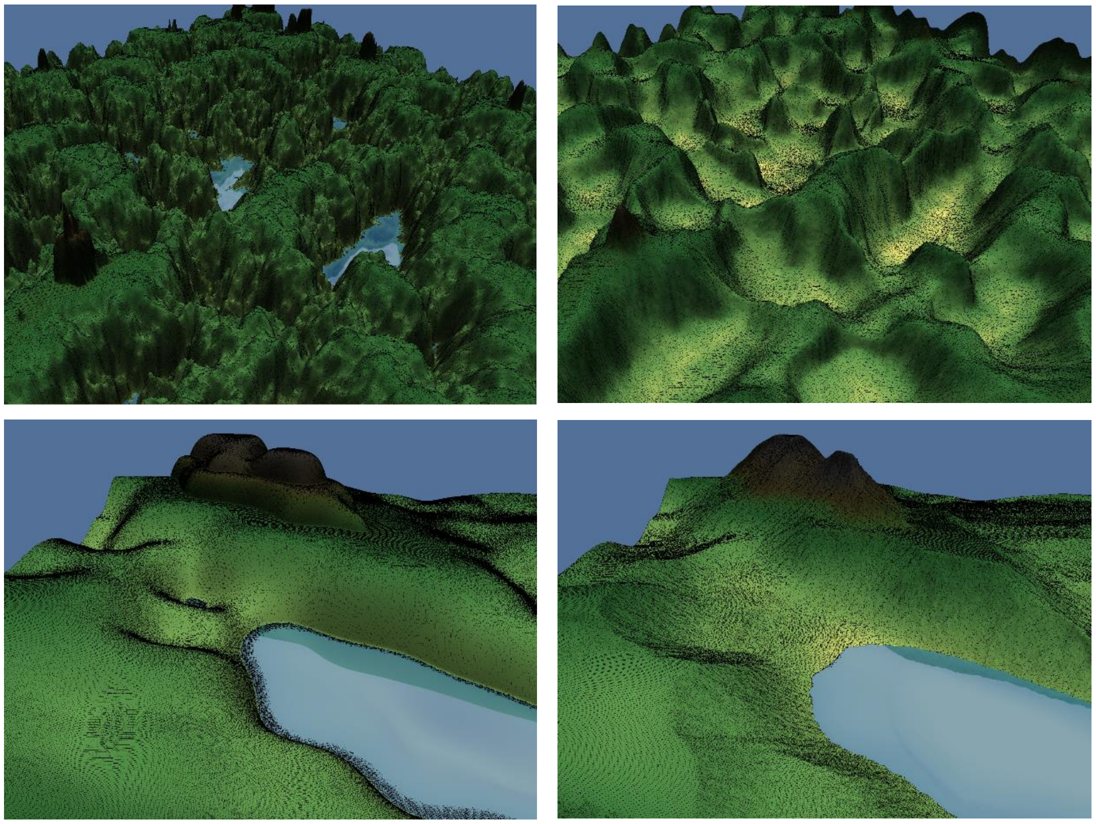
Those are some examples showing what the algorithm can do.











7 Conclusion

This particle-based algorithm simplifies natural processes in order to reproduce as fair as possible the natural development of a terrain. It simulates the hydraulic erosion where particles try to mimic water running on the terrain and distributing the sediment. The algorithm is highly adaptable to different types of input terrain and the result is adjustable through a wide range of parameters.

8 References

[E-DOG] Water erosion on heightmap terrain:
http://ranmantaru.com/blog/2011/10/08/water-erosion-on-heightmap-terrain/

[O. Šťava, B. Beneš, M. Brisbin, and J. Kˇrivánek] Interactive Terrain Modeling Using Hydraulic Erosion:
http://hpcg.purdue.edu/papers/Stava08SCA.pdf

[Xing Mei, Philippe Decaudin, Bao-Gang Hu] Fast Hydraulic Erosion Simulation and Visualization on GPU:
https://hal.inria.fr/inria-00402079/document

[Sebastian Lague] Demo of Hydraulic erosion on a procedural terrain:
https://sebastian.itch.io/hydraulic-erosion

[Hans Theobald Beyer] Implementation of a method for hydraulic erosion:
https://www.firespark.de/resources/downloads/implementation%20of%20a%20methode%20for%20hydraulic%20erosion.pdf