Introduction to Computer Graphics
Project Part 1 - Terrain Generation

Group ?? - Florian Junker, Louis Séguy, Mathieu Monney

April 14, 2014

1 Overview

Our program uses procedural techniques to generate a terrain. It uses a noise function and fractal Brownian motion to compute the height at each point of the surface. It also computes the shading and a color depending on the height. See Figure 1 for a rendering.

2 Implementation

2.1 Display

We used the framework provided with Homework 6. Though, it seems it was not designed to be used with multiple include statements. When we try to include common.h in multiple files, there are multiple symbols and the linker can't produce an executable file. The problem is located in the `glfw_trackball.h`, `glfw_helpers.h` and `shader_helpers.h` files. There are some variable definitions in .h files instead of being in .cpp files. Since we didn't want to modify the framework for now, all our code is in a single `main.cpp` file, sorry !

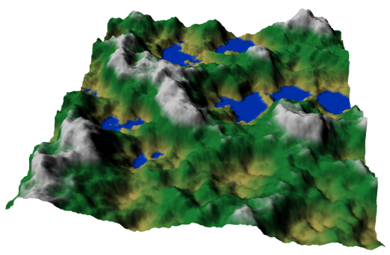

Figure 1: A render of our actual project stage.

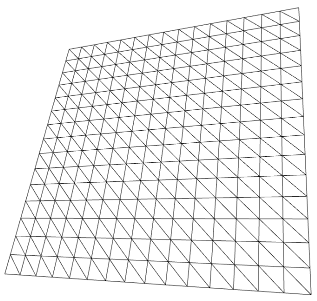

Figure 2: A rendering of the triangle grid.

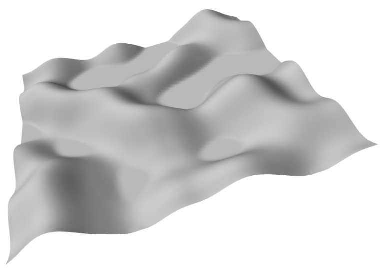

Figure 3: A rendering of our world shaded.

2.2 Triangle grid

We generate a 2D flat grid that will be displaced according to the height map. To avoid duplicate vertices, we use an indice buffer. To be able to draw the grid in a single glDrawElements call, we use the glPrimitiveRestartIndex instruction to specify an index that will indicate OpenGL we need to draw a new triangle strip.

2.3 Shading

We compute the shading for each pixel by doing the dot product between the sun direction vector and the surface normal. The sun distance is aproximate to infinity and so the sun direction vector is fix for all pixels. The normal for a specific position is compute by taking an arbitrary small area with this position as center. Knowing the coordinates of each corner of this area, the can do the cross product of two adjacent eadges.

2.4 Noise

The biggest problem we encountered while implementing Perlin noise was passing the arrays from the C++ code to the fragment shader. We accomplished this by creating a texture, loading the data in it and passing a sampler as uniform

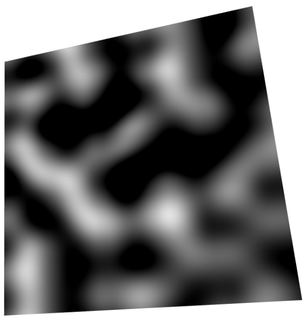

Figure 4: A simple perlin noise.

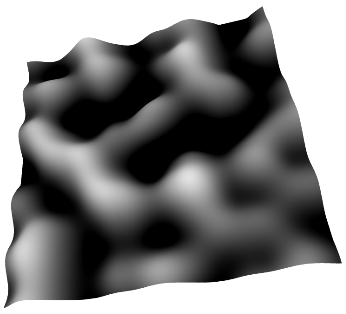

Figure 5: Our world after heightmap implementation.

to access the arrays in the shader. Once it was done implementing Pelin noise was straight forward, we followed the algorithm given in the project description (`http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter26.html`).

2.5 Heightmap

Creating the heightmap was pretty much straigthforward. A simple lookup in the texture computed by the perlin noise algorithm give the z value.

2.6 Fractional Brownian motion

Implementing fBm was mainly about searching for good parameters. We read about fBm on the internet and found multiple implementation, however, the idea was always the same. Our implementation is largy inspired by the following link.
`https://code.google.com/p/fractalterraingeneration/wiki/Fractional_Brownian_Motion`
You can see a sample output at this stage on Figure 6 and 7

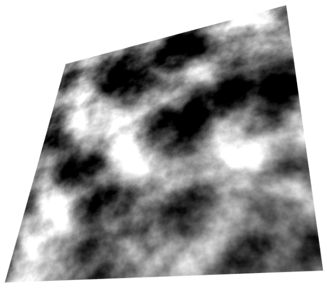

Figure 6: fractal Brownian motion.

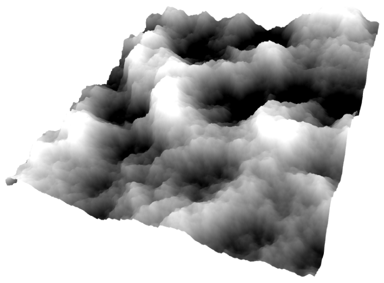

Figure 7: Our world with fractal Brownian motion.

3 Results

3.1 Triangle grid

The grid size influences heavily the performances. Indeed, more vertices means more triangles to draw. On the other hand, the grid coordinates have no effect on the performance, it just allows us to draw further in our world coordinates.

3.2 Shading

The only parameter we use to compute shading is the arbitrary offset named `small_offset`. Using a too large or too small value implies a unrealistic normal. We figured the value by rendering the image until we were happy about th result. You can see the results in Figure 3

3.3 Noise

The Noise computation is pretty fast (instantaneous), at least with our current parameters (512x512 texture to generate). Modifying the permutation array with a different permutation produces a different terrain. You can see the results in Figure 4

3.4 Heightmap

There are no parameter for this. Except maybe the sea level. The sealevel parameter let us define a height from which the terrain is plan. This let us draw water. You can see the result in Figure 5

3.5 Fractional Brownian motion

For fBm we use a frequency of 0.008. The frequency change the number of mountain per unit of area. If the frequency is very low then we get a "plan" area. We tried different values until we were satisfy of the resulting image. We used a lacunarity of 2.15. The lacunarity is what make the frequency grow. We choose 2.15 by experimenting the differents values. We began by 2 since almost everyone use similar values. Finnaly our amplitude is 1. The amplitude parameter is what fix the height of the montains. You can see the results in Figure 6 and 7

3.6 Turbulence

The only difference between Turbulence and fractional Brownian motion is that we sum over the absolut value of the noise function. The parameters behave as they do with fBm, the difference is that the lower value in the heighMap will be higher and so, if we want to have some lakes, we have to move the seaLevel up.